Università degli Studi di Salerno
Corso di Ingegneria del Software

HairBeautyNow
ODD: Object Design Document
Versione 2.0

Data: 09/02/2025

Progetto: HairBeautyNow	Versione: 2.0
Documento: Object Design Document	Data: 09/02/2025

Coordinatore del progetto:

Nome	Matricola

Partecipanti:

Nome	Matricola
Aniello Cirillo	0512117775
Christian Pio Lanziero	0512117931

Scritto da:	

Revision History

Data	Versione	Descrizione	Autore
27/11/2024	0.1	Creazione del documento	Tutto il team
06/12/2024	0.2	Scopo del sistema, Linee guida e definizione dei packages	Tutto il team
12/12/2024	0.3	Packaging del sistema, dei layer e dei singoli sottoinsiemi	Tutto il team
16/12/2024	1.0	Package data, interfaccia delle classi e prima release del documento	Tutto il team
09/02/2025	2.0	Revisione finale del documento, ristrutturazione dell'utilizzo dello Strategy Pattern, revisione dei Package, rivisitazione dell'interfaccia delle classi ed aggiornamento del Class diagram.	Tutto il team

Indice

1. Introduzione

Il presente **Object Design Document** ha l'obiettivo di approfondire in modo dettagliato gli aspetti relativi all'implementazione del sistema **HairBeautyNow**, integrando e ampliando quanto definito nei documenti precedenti, che si sono focalizzati principalmente sull'architettura e sulla progettazione generale del sistema. Questo documento offre una descrizione tecnica e completa delle decisioni progettuali maturate durante le fasi di analisi e design, evidenziando i principali compromessi valutati, le linee guida per la documentazione delle interfacce e l'adozione dei **Design Pattern** più adeguati.

Nel corso del documento verranno inoltre definiti i **packages**, le interfacce delle classi e i diagrammi UML che rappresentano le principali decisioni implementative del sistema. Sarà dedicata particolare attenzione alla descrizione dettagliata delle operazioni, dei parametri e delle firme delle varie interfacce e classi, garantendo coerenza con i sottosistemi definiti nel **System Design Document (SDD_HairBeautyNow)** e con i requisiti funzionali e non funzionali indicati nel **Requirements Analysis Document (RAD_HairBeautyNow)**.

Lo scopo principale dell'**Object Design Document** è fornire un modello chiaro e dettagliato per implementare tutte le funzionalità identificate, favorendo una transizione senza ostacoli verso la fase di sviluppo e assicurando che il sistema mantenga elevati standard di manutenibilità ed espandibilità.

1.1 Scopo del sistema

Lo scopo del sistema è facilitare la gestione centralizzata e la prenotazione di appuntamenti per una catena di barbieri e saloni di bellezza, offrendo una piattaforma digitale intuitiva e funzionale. In un contesto in cui la cura personale è sempre più importante, il sistema consente agli utenti di

prenotare comodamente i loro appuntamenti, visualizzare la disponibilità in tempo reale, e scegliere il servizio o il professionista desiderato presso una qualsiasi delle sedi della catena. I gestori della sede possono amministrare facilmente le prenotazioni a livello di singola sede, gestire i servizi offerti, e migliorare la comunicazione con i clienti, ottimizzando così l'efficienza operativa e l'esperienza complessiva del brand.

Il sistema è una web-app pensata sia per i clienti che vogliono prenotare servizi presso una delle sedi della catena di barbieri e saloni di bellezza, sia per i gestori delle singole sedi e per la gestione centrale della catena. Gli utenti possono creare profili personali, visualizzare la disponibilità dei professionisti in tempo reale, e prenotare appuntamenti per diversi servizi come taglio, colore, manicure, e altri trattamenti estetici presso la sede preferita.

I gestori delle singole sedi possono utilizzare la piattaforma per amministrare le prenotazioni, e monitorare l'andamento delle attività giornaliere nella propria sede. Inoltre, la gestione centrale del franchise permette di monitorare e gestire le operazioni di tutte le sedi in modo unificato, coordinando risorse per migliorare la performance complessiva del brand. Il sistema permette di personalizzare i servizi offerti, consentendo agli utenti di scegliere specifici professionisti e servizi, favorendo così un'esperienza ottimizzata e uniforme per i clienti di tutta la catena.

1.2. Object Trade-off considerati

1. Rapidità di Sviluppo e Funzionalità

Per rispettare i tempi di consegna prefissati, alcune funzionalità avanzate, come la possibilità di modificare una prenotazione, modificare una sede oppure la gestione di promozioni, non sono state incluse nella versione iniziale del sistema. Questa scelta ha permesso di fornire rapidamente le funzionalità principali, pianificando l'aggiunta di funzionalità secondarie in aggiornamenti futuri.

2. Usabilità e Affidabilità

Le password degli utenti sono crittografate per prevenire accessi non autorizzati, è stato implementato un sistema di autenticazione sicuro con crittografia nel database. Tuttavia, per mantenere un'esperienza utente fluida, non è stata inclusa la verifica in due passaggi nella versione iniziale del sistema, riducendo leggermente il livello di sicurezza.

1.3 Linee guida per la documentazione dell'interfaccia

Nomi dei Package

I nomi dei package nel sistema **HairBeautyNow** devono essere scritti in **minuscolo** e non devono contenere spazi o caratteri speciali. In caso di nomi composti da più parole, è necessario utilizzare il formato **snake_case**.
Esempio: gestione_servizi, gestione_sede

Nomi delle Classi

I nomi delle classi devono seguire il formato **PascalCase**, iniziando con una lettera maiuscola. Devono essere descrittivi e riflettere chiaramente l'entità o la funzionalità implementata.
Esempio: Utente, Prenotazione.

Classi DAO

Le classi DAO devono terminare con il suffisso **DAO** per indicare il loro ruolo di accesso ai dati.
Esempio: UtenteDAO,

Classi Bean

Le classi Bean devono terminare con il suffisso **Bean**, utilizzato per rappresentare entità del dominio.
Esempio: UtenteBean, ServizioBean

Classi di Gestione

Le classi di gestione devono indicare chiaramente la funzionalità principale svolta.
Esempio: PrenotazioniService, SaloneService

Nomi delle Servlet

I nomi delle Servlet non utilizzate per la logica di business devono seguire il formato **PascalCase** e terminare con il suffisso **Servlet**.
Esempio: PrenotazioneServlet

Nomi dei Metodi

I metodi devono avere nomi descrittivi che riflettano chiaramente l'operazione svolta. I nomi devono iniziare con una lettera minuscola e seguire il formato **camelCase**.
Esempio: addPrenotazione(), creaGestore()

Nomi delle Variabili

I nomi delle variabili devono essere descrittivi e seguire il formato **camelCase**, iniziando con una lettera minuscola. Per i nomi composti, ogni parola interna deve iniziare con una lettera maiuscola.
Esempio: nCarta, dataScadenza.
È possibile utilizzare variabili temporanee abbreviate (ad esempio, **i**), ma solo in contesti limitati come cicli for o altri ambiti ristretti.

Nomi delle JSP

I file JSP devono seguire il formato **camelCase** e riflettere chiaramente il contenuto o la funzionalità della pagina.
Esempio: rimozioneProfessionisti.jsp

Organizzazione delle Componenti:

● **Classi e Package:** Tutte le classi che realizzano un sottosistema devono essere racchiuse nello stesso package Java, organizzate in modo logico in base alla loro funzione in modo da mantenere un sistema coeso.

● **Risorse Statiche:** Fogli di stile, script e immagini devono essere organizzati nella directory static, con sottocartelle per ciascun tipo di file (es. static/style, static/js, static/images).

1.4 Design Pattern

Per implementare le funzionalità del sistema **HairBeautyNow**, sono stati adottati due design pattern principali: **Singleton Pattern** e **Strategy Pattern**. Di seguito vengono illustrate le motivazioni che hanno portato all'utilizzo di tali pattern nel contesto dell'applicazione.

Singleton Pattern

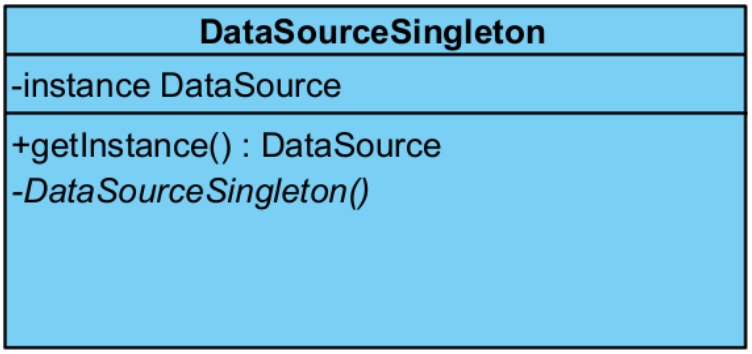
L'applicazione richiede un controllo centralizzato per l'accesso alla risorsa di connessione al database, essendo quest'ultima una risorsa condivisa. L'adozione del **Singleton Pattern** si rivela quindi utile per limitare e gestire l'accesso concorrente alla risorsa **DataSource**.

L'implementazione del **Singleton Pattern** garantisce che esista una sola istanza di **DataSource** durante l'intero ciclo di vita dell'applicazione. In questo modo:

- Si previene la creazione simultanea di connessioni ridondanti.
- Si ottimizza il ciclo di vita della connessione condivisa.
- Si semplifica il coordinamento e la gestione dell'accesso ai dati.

Nel contesto del sistema **HairBeautyNow**, la classe **DatabaseSourceSingleton** rappresenta una specializzazione del pattern Singleton. Questa classe include un attributo statico, di tipo **DataSource**, denominato `instance`.

L'accesso al **DataSource** avviene tramite il metodo **getConnection()**, che restituisce l'istanza unica della connessione. La gestione di tale connessione è integrata nel contesto dell'applicazione web tramite la classe **MainContext**, che centralizza l'uso della risorsa condivisa.



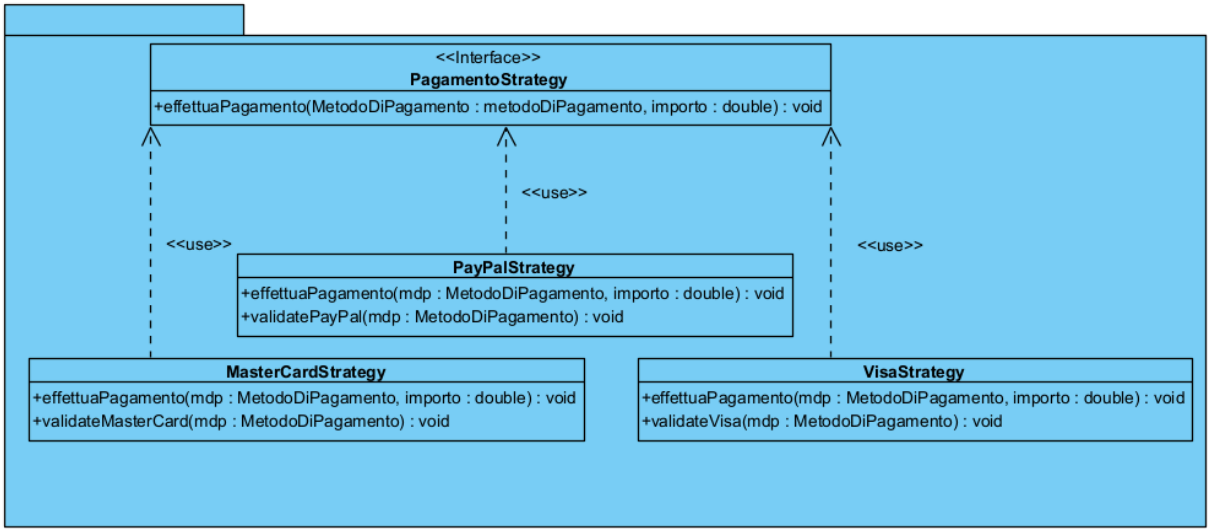
Strategy Pattern

Nel sistema **HairBeautyNow**, il form di pagamento deve essere dinamico perché ogni metodo di pagamento richiede informazioni diverse. Ad esempio, le carte di credito necessitano del numero della carta, del CVV e della data di scadenza, mentre PayPal richiede un'email.

Per gestire questa variabilità senza rendere il codice rigido e difficile da mantenere, è stato adottato lo **Strategy Pattern**. Questo approccio permette di definire una struttura flessibile in cui ogni metodo di pagamento ha una propria strategia di validazione, separata dalle altre.

Per il sistema **HairBeautyNow**, sono stati definiti i seguenti componenti:

- **PagamentoStrategy (Interfaccia Generale)**
  Questa interfaccia stabilisce un contratto comune per tutti i metodi di pagamento, dichiarando un metodo che permette di validare i dati inseriti. Qualsiasi metodo di pagamento deve implementare questa interfaccia per essere compatibile con il sistema.
- **VisaStrategy**
  Questa classe rappresenta la strategia di validazione per le carte **Visa**. Controlla che il numero della carta inizi con il prefisso corretto, che il CVV sia lungo tre cifre e che la data di scadenza sia in un formato valido.
- **MasterCardStrategy**
  Simile alla strategia per Visa, ma con regole specifiche per le carte **MasterCard**, come un prefisso numerico diverso.
- **PayPalStrategy**
  Questa classe si occupa della validazione per i pagamenti tramite **PayPal**, verificando che l'email inserita dall'utente sia in un formato corretto (ad esempio, che contenga "@" e un dominio valido).



1.5 Definizione, acronimi e abbreviazioni

- **HairBeautyNow:** Nome della piattaforma che verrà sviluppata per la gestione delle prenotazioni e delle operazioni nei saloni di bellezza e dei barbieri del franchising, con funzionalità per la gestione centralizzata delle sedi.
- **Dashboard:** Un'interfaccia grafica dove l'utente può visualizzare e/o gestire i dati in base ai suoi permessi.
- **Utente Generale:** Utente che può visitare il sito ed esplorare i servizi offerti dal franchising. Tuttavia, per prenotare un servizio, l'Utente Generale deve registrarsi.
- **Utente Acquirente:** Un utente registrato su HairBeautyNow che può prenotare servizi, selezionare la sede, il professionista e l'orario preferito, oltre a visualizzare in tempo reale le offerte promozionali disponibili.
- **Utente Gestore Sede(UGS):** Un utente registrato su HairBeautyNow che ha il ruolo di gestire una singola sede del franchising. L'Utente Gestore Sede può amministrare le prenotazioni, gestire le postazioni e i professionisti della sede, monitorare la disponibilità delle risorse e promuovere offerte e sconti per la sede specifica. L'Utente Gestore Sede è unico per ogni sede, ed è necessario affinché la singola sede possa essere operativa.
- **Utente Gestore Catena(UGC):** Un utente registrato su HairBeautyNow che ha il compito di gestire più sedi di una catena di saloni. L'Utente Gestore Catena può monitorare e amministrare centralmente le attività di tutte le sedi, visualizzare le performance complessive della catena e gestire le promozioni a livello multi-sede.

1.6 Riferimenti

Libro: Object-Oriented Software Engineering - Using UML, Patterns and Java.
Autori: Bernd Bruegge & Allen H. Dutoit
Documenti:

- RAD_HairBeautyNow : Sono descritte le funzionalità individuate in fase di analisi.
- SDD_HairBeautyNow : Contiene una descrizione completa dell'architettura del sistema, inclusi i sottosistemi individuati ed i servizi forniti da ciascun sottosistema.

2.Packages

2.1 Panoramica

In questa sezione verrà prima mostrata una panoramica generale del packaging completo del sistema HairBeautyNow. Dopodichè nella sezione 2.2 si entrerà nel dettaglio. Mentre nella sezione 2.4 si analizzeranno i singoli package e le interfacce delle classi che sono al suo interno. Il progetto sarà organizzato in modo modulare per garantire una gestione chiara del codice sorgente.

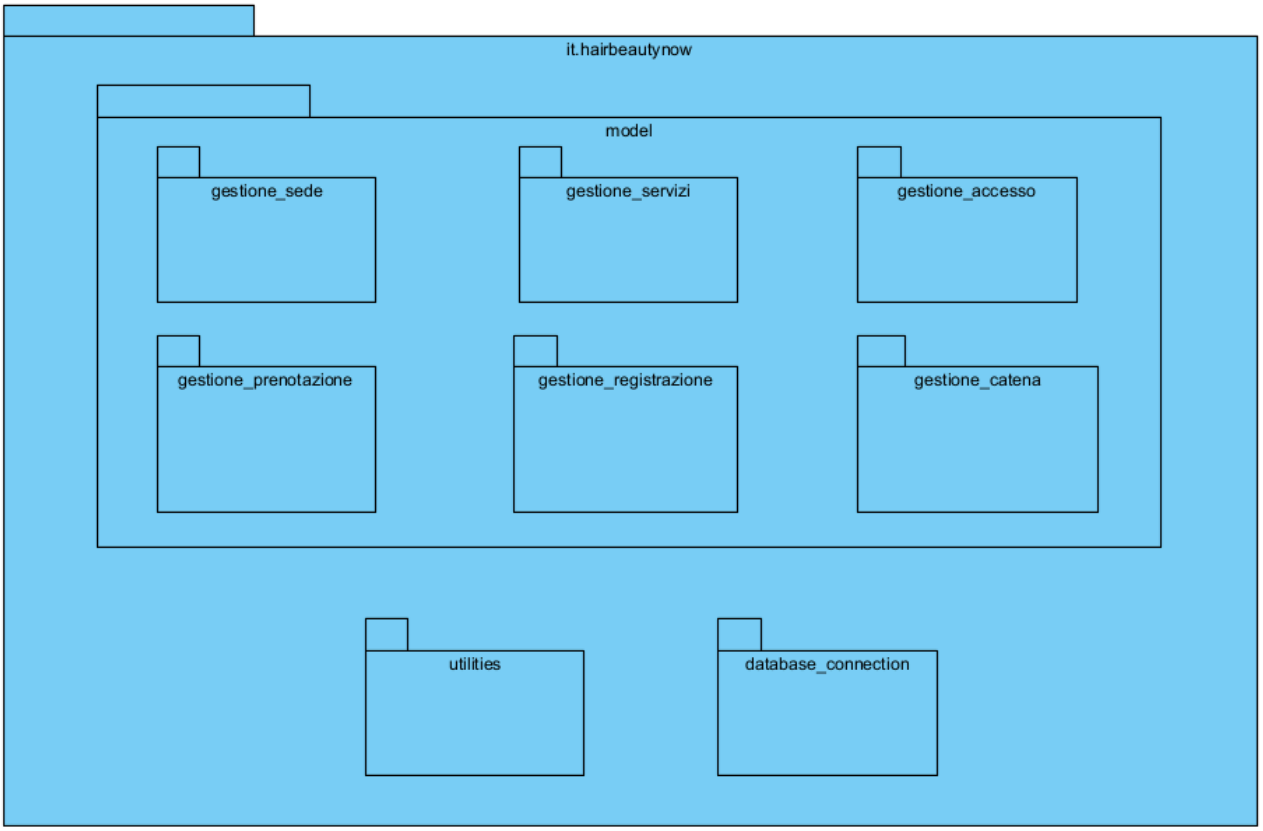
Struttura:

- **src**/
- │
- ├── **main**/
- │ ├── **java**/
- │ │ ├── it.hairbeautynow/
- │ │ │ ├── **sottosistemi**/
- │ │ │ │ ├── **gestione_sede**/
- │ │ │ │ │ ├── ◆view
- │ │ │ │ │ ├── ◆service
- │ │ │ │ ├── **gestione_servizi**/
- │ │ │ │ │ ├── ◆view
- │ │ │ │ │ ├── ◆service
- │ │ │ │ ├── **gestione_accesso**/
- │ │ │ │ │ ├── ◆view
- │ │ │ │ │ ├── ◆service
- │ │ │ │ ├── **gestione_registrazione**/
- │ │ │ │ │ ├── ◆view
- │ │ │ │ │ ├── ◆service
- │ │ │ │ ├── **gestione_prenotazione**/
- │ │ │ │ │ ├── ◆view
- │ │ │ │ │ ├── ◆service
- │ │ │ │ ├── **gestione_catena**/
- │ │ │ │ │ ├── ◆view
- │ │ │ │ │ ├── ◆service
- │ │ │ │
- │ │ │ ├── **model**/
- │ │ │ │ ├── **entity**: Contiene le entità del sistema
- │ │ │ │ ├── **dao**:Contiene le classi responsabili all'accesso ai dati
- │ │ │ ├── **database_connection**/
- │ │ │ │ ├── Classi per la connessione al database

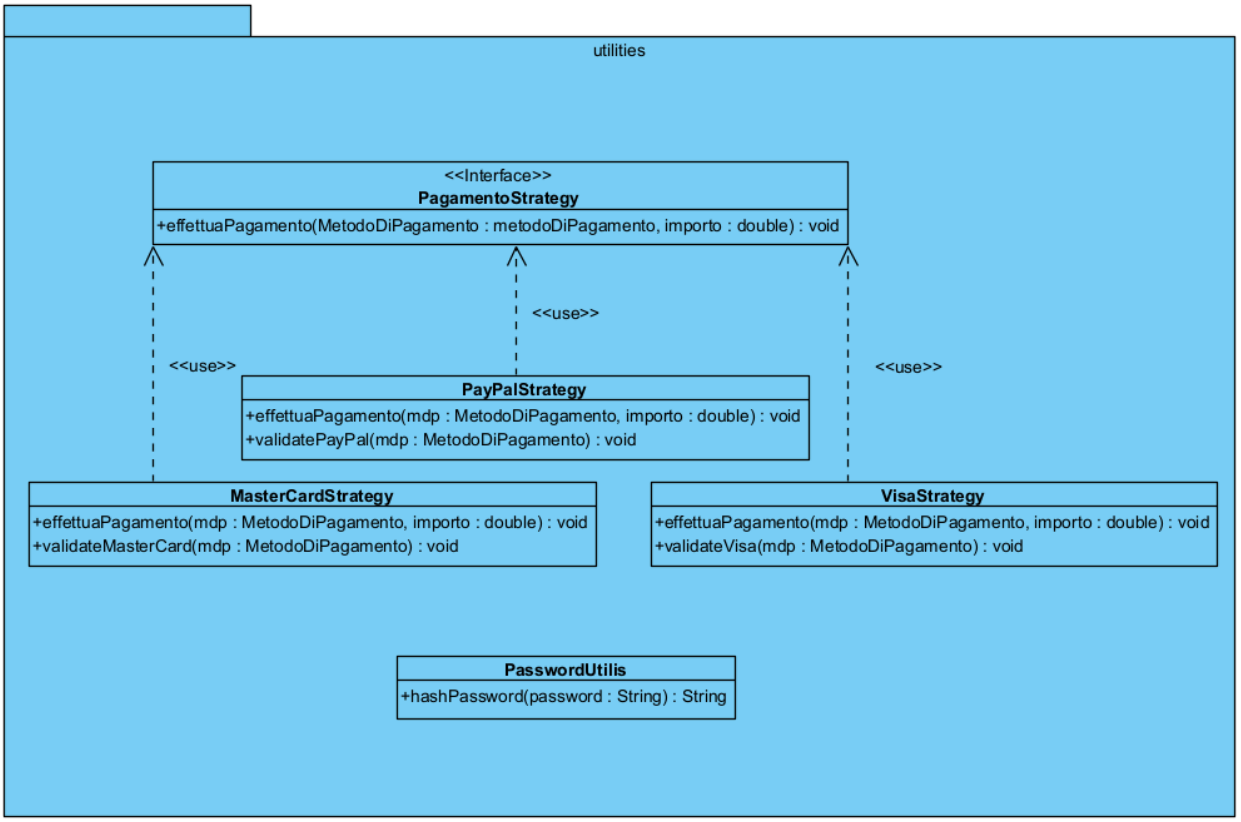
- │ │ │ ├── **utilities**/
- │ │ │ │ ├── Classi per la validazione di campi
- │ ├── **webapp**/
- │ │ ├── **META-INF**/
- │ │ ├── **WEB-INF**/
- │ │ │ ├── **jsp**/
- │ │ │ ├── **static**/
- │ │ │ │ ├──images
- │ │ │ │ ├──js
- │ │ │ │ ├──style
- └── **test**/
- │ ├── **java**/
- │ │ ├── **integration**: Classi del test di integrazione
- │ │ ├── **unit**: Classi del test di unità
- │ │ ├──**system:** Classi del test di sistema

I package identificati con “view”, nei sottosistemi, contengono le servlet per la logica dipresentazione. I “service”, invece, offrono i servizi dei sottosistemi.

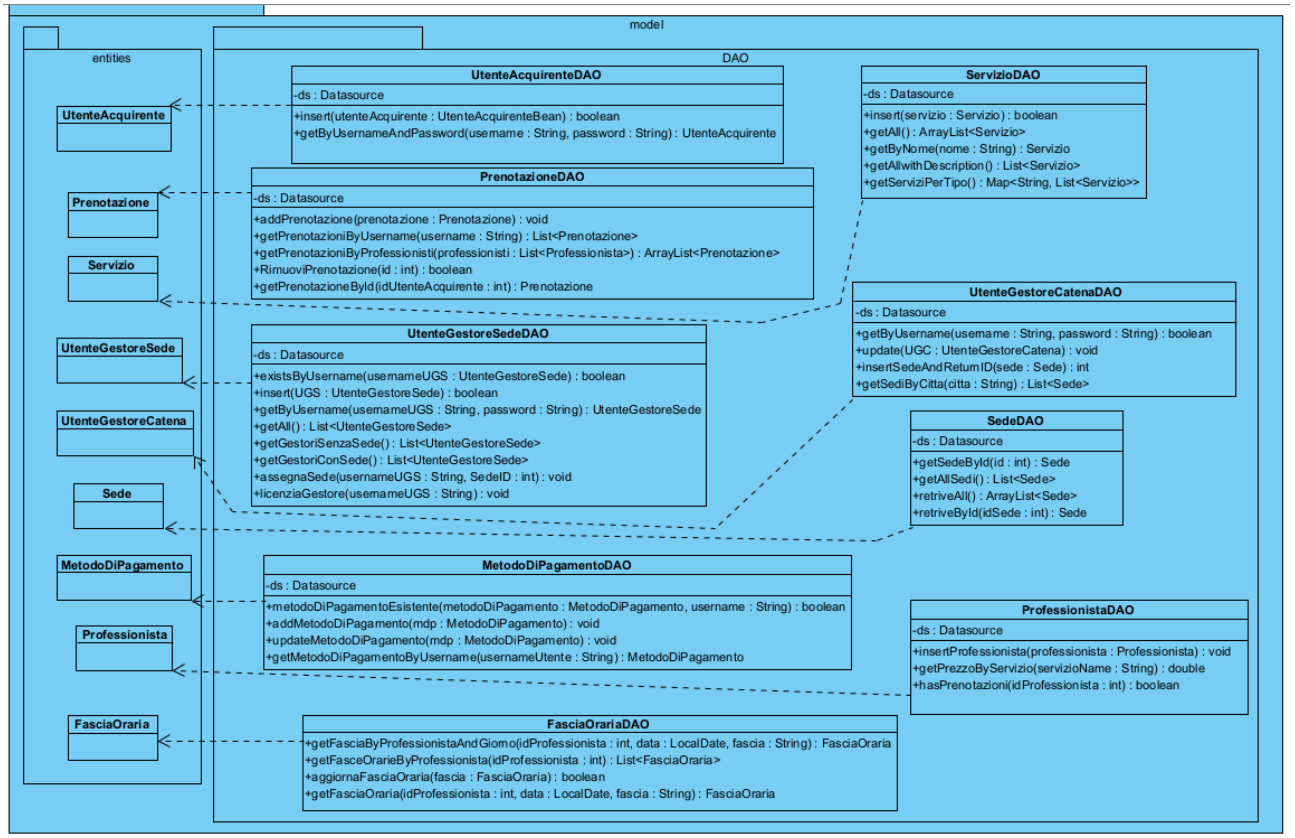
2.2 Packaging del sistema



2.3 Package utilities

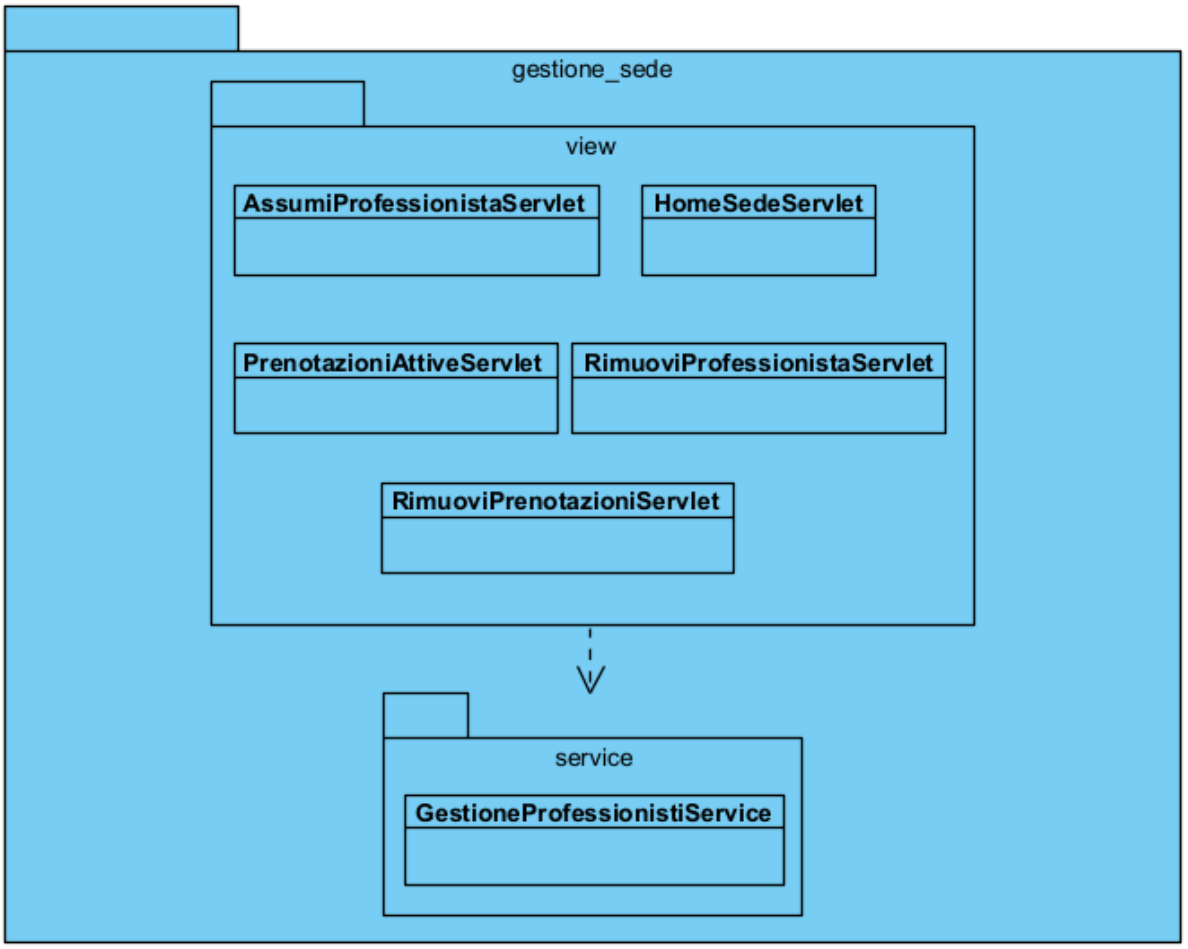

utilities
<<Interface>>
PagamentoStrategy
+effettuaPagamento(MetodoDiPagamento : metodoDiPagamento, importo : double) : void
<<use>>
<<use>>
<<use>>
PayPalStrategy
+effettuaPagamento(mdp : MetodoDiPagamento, importo : double) : void
+validatePayPal(mdp : MetodoDiPagamento) : void
MasterCardStrategy
+effettuaPagamento(mdp : MetodoDiPagamento, importo : double) : void
+validateMasterCard(mdp : MetodoDiPagamento) : void
VisaStrategy
+effettuaPagamento(mdp : MetodoDiPagamento, importo : double) : void
+validateVisa(mdp : MetodoDiPagamento) : void
PasswordUtilis
+hashPassword(password : String) : String

2.4 Package model

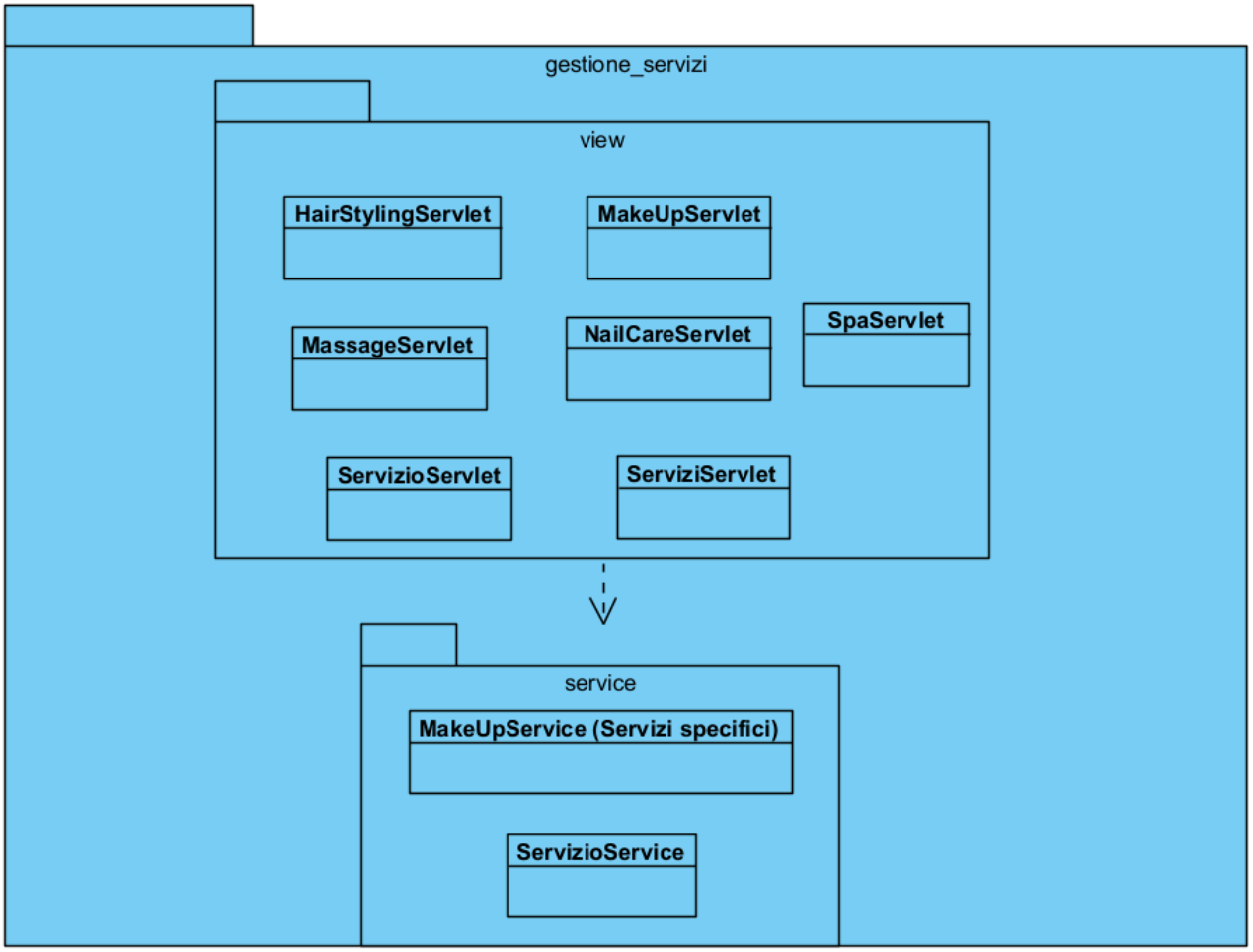


2.5 Package di sottosistema

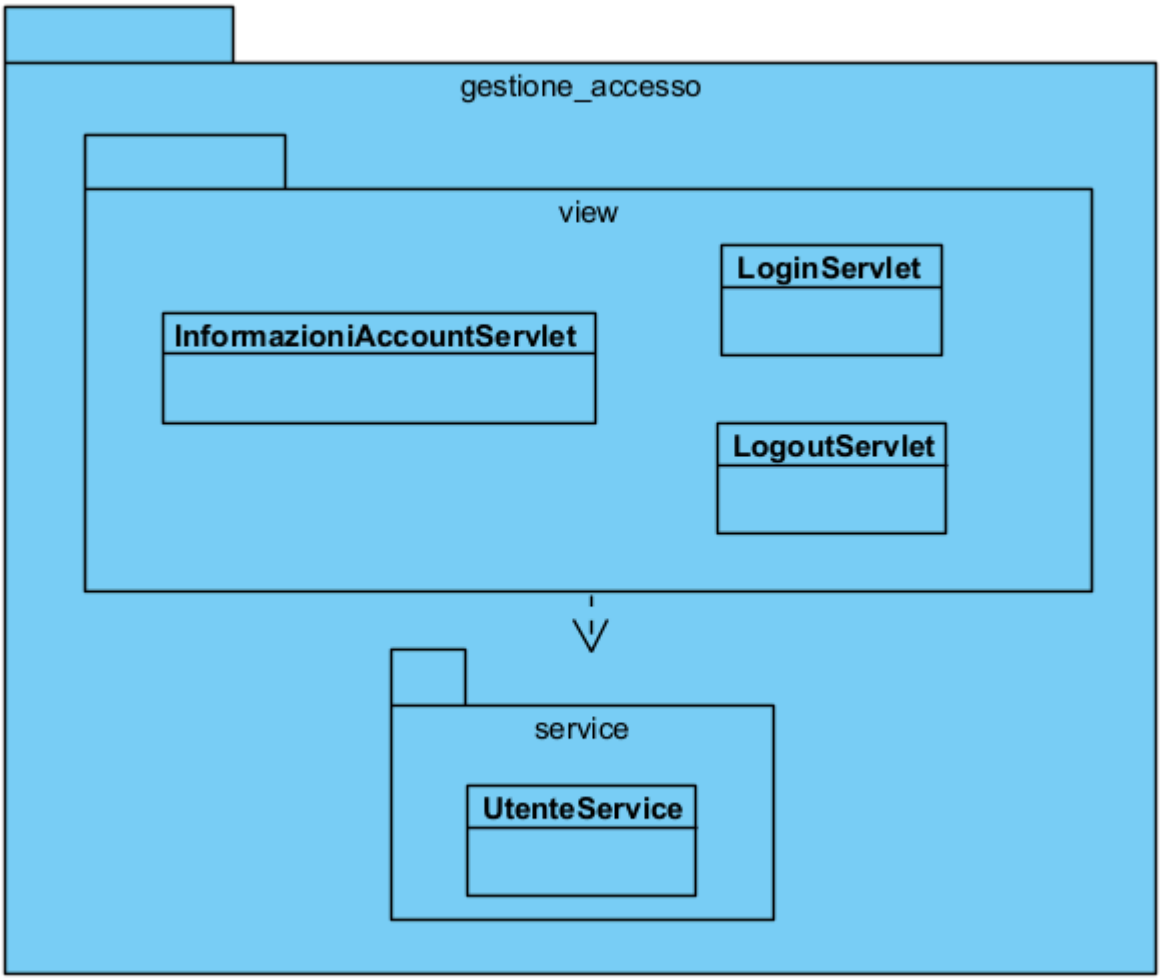
2.5.1 Package gestione_sede



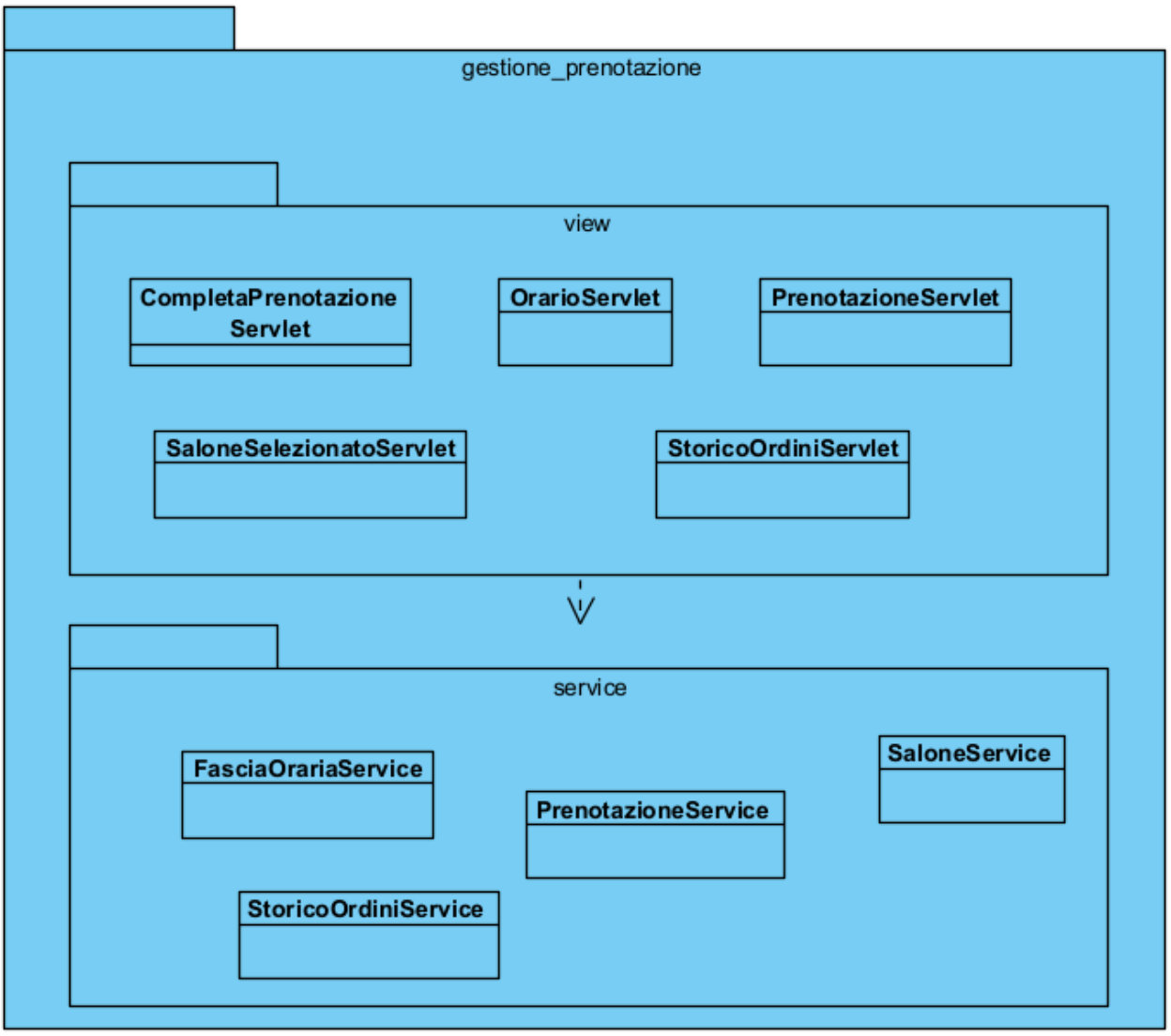
2.5.2 Package gestione_servizi



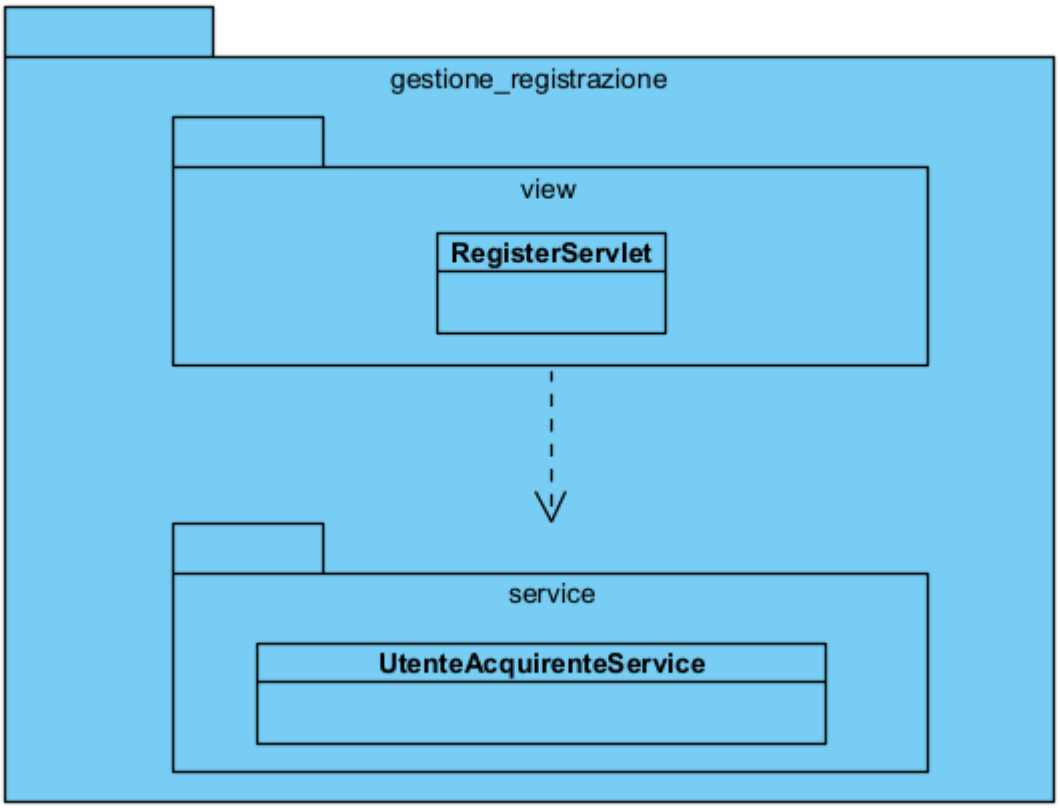
2.5.3 Package gestione_accesso



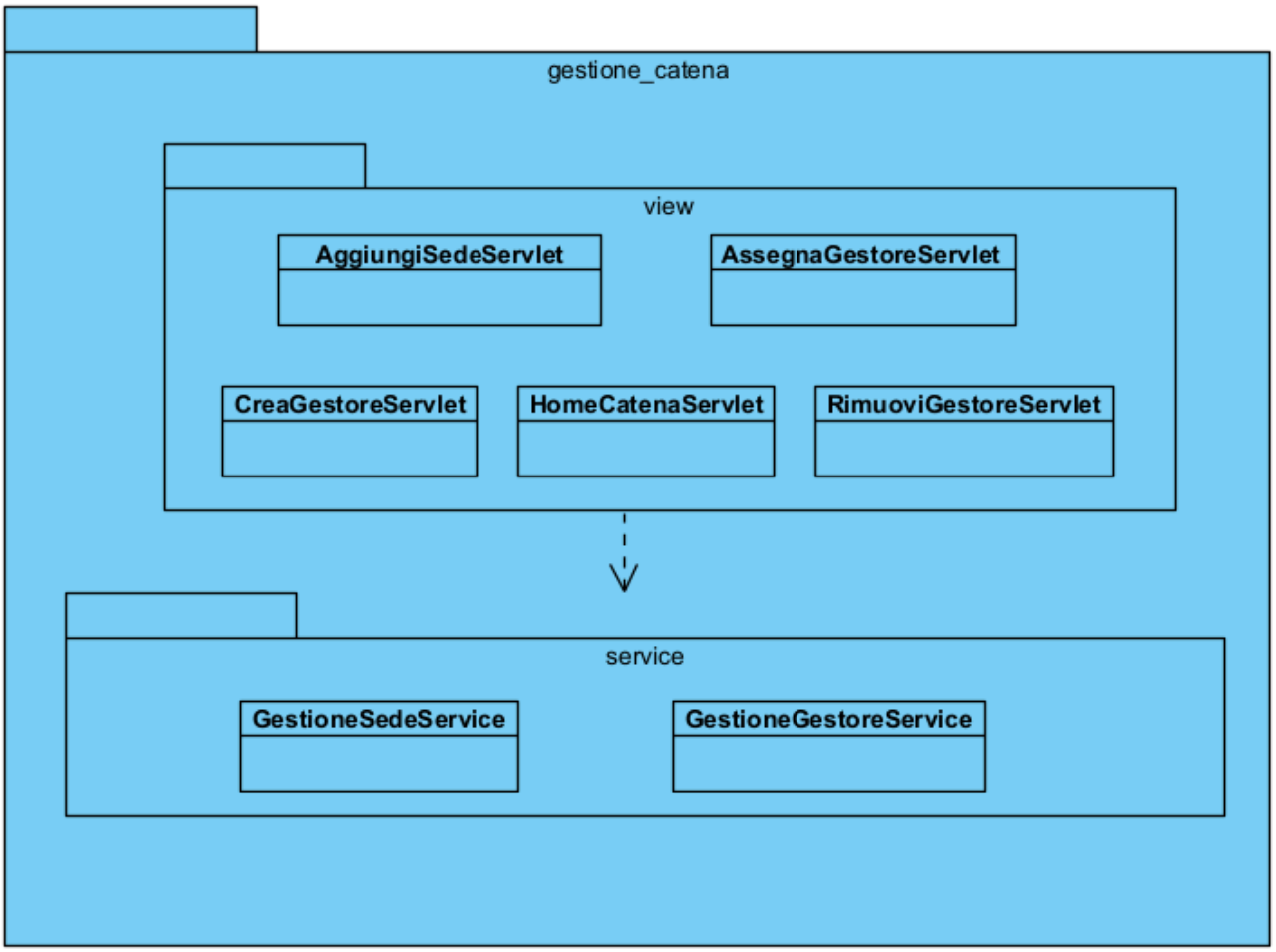
2.5.4 Package gestione_prenotazione



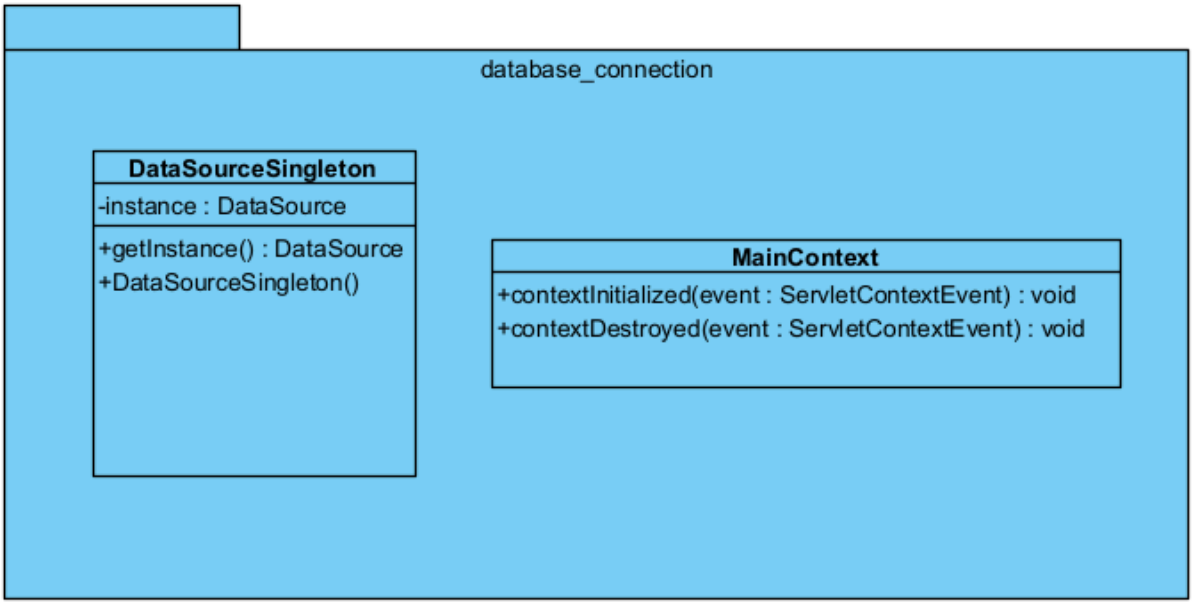
2.5.5 Package gestione_registrazione



2.5.6 Package gestione_catena



2.6 Package database_connection



3. Interfaccia delle classi

La terza sezione, **Interfacce delle Classe**, fornisce una descrizione delle classi e delle relative interfacce pubbliche. Questa sezione include una panoramica di ciascuna classe, evidenziando le dipendenze con altre classi e package, gli attributi pubblici, le operazioni disponibili e le eventuali eccezioni che possono essere generate.

Gestione_Accesso

NOME CLASSE	**UtenteService**
METODI	+login(username : String, password : String) : Object +logout(request : HttpServletRequest) : void
INVARIANTI	//

NOME METODO	+login(username : String, password: String) : Object
DESCRIZIONE	Questo metodo permette ad un UtenteGuest di loggarsi.
PRE-CONDIZIONE	**context:** UtenteService::login(username:String, password:String) **pre:** email <> null, AND Email.isValid() AND password<> null AND password.isValid()
POST-CONDIZIONE	**context:** UtenteService::login(username:String, password:String) **post:** result <> null AND result.email = email AND result.password = password

NOME METODO	+logout(request : HttpServletRequest) : void
DESCRIZIONE	Questo metodo permette ad un Utente

	Autenticato di effettuare la disconnessione.
PRE-CONDIZIONE	**context:** UtenteService::logout(request : HttpServletRequest) **pre:** request <> null
POST-CONDIZIONE	**context:** UtenteService::logout(request : HttpServletRequest) **post:** request NULL.

Gestione_Registrazione

NOME CLASSE	**UtenteAcquirenteService**
METODI	+createUser(utenteAcquirente : UtenteAcquirente) : boolean
INVARIANTI	//

NOME METODO	+createUser(utenteAcquirente : UtenteAcquirente) : boolean
DESCRIZIONE	Questo metodo effettua la registrazione utente.
PRE-CONDIZIONE	**context:** UtenteAcquirenteService::createUser(utenteAcquirente : UtenteAcquirente) **pre:** UtenteAcquirente <> NULL
POST-CONDIZIONE	**context:** UtenteAcquirenteService::createUser(utenteAcquirente : UtenteAcquirente) **post:** UtenteAcquirente.exists() == TRUE

Gestione_servizi

NOME CLASSE	**MakeUpService**
METODI	+getAllServiziWithDescription() : List<Servizio> +getServiziPerTipo() : Map<String, List<Servizio>> +getPrezzoByNome(nome : String) : double
INVARIANTI	//

NOME METODO	+getAllServiziWithDescription() : List<Servizio>
DESCRIZIONE	Questo metodo prende tutti i servizi esistenti con la descrizione annessa.
PRE-CONDIZIONE	**context:** MakeUpService::getAllServiziWithDescription() **pre:** Servizio.exists(TRUE)
POST-CONDIZIONE	**context:** MakeUpService::getAllServiziWithDescription() **post:** List<Servizio>.isEmpty() = FALSE

NOME METODO	+getServiziPerTipo() : Map<String,

	List<Servizio>>
DESCRIZIONE	Questo metodo seleziona il servizio specifico in base alla categoria della richiesta.
PRE-CONDIZIONE	**context:** MakeUpService::getServiziPerTipo() **pre:** List<Servizio>.exists() = TRUE
POST-CONDIZIONE	**context:** MakeUpService::getServiziPerTipo() **post:** Map<String, List<Servizio>>.isEmpty() = FALSE

NOME METODO	+getPrezzoByNome(nome : String) : double
DESCRIZIONE	Questo metodo ricava il prezzo di ogni servizio.
PRE-CONDIZIONE	**context:** MakeUpService::getPrezzoByNome(nome : String) **pre:** nome <> NULL
POST-CONDIZIONE	**context:** MakeUpService::getPrezzoByNome(nome : String) **post:** result = double Servizio.prezzo

NOME CLASSE	ServizioService
METODI	+getAllServizi() : List<Servizio> +getPrezzoByNome(nome : String) +getServiziPerTipo() : Map<String, List<Servizio>>
INVARIANTI	//

NOME METODO	+getAllServizi() : List<Servizio>
DESCRIZIONE	Questo metodo ricava tutti i servizi da mostrare all'utente nella pagina generale dei servizi.
PRE-CONDIZIONE	**context:** ServizioService::getAllServizi() **pre:** Servizio.exists() = TRUE
POST-CONDIZIONE	**context:** ServizioService::getAllServizi() **post:** List<Servizio>.isEmpty = FALSE

Gestione_sede

NOME CLASSE	GestioneProfessionistaService
METODI	+getSedeById(SedeId : int) : Sede +assumiProfessionista(professionista : Professionista) : void +rimuoviProfessionista(professionistaID : int) : String
INVARIANTI	//

NOME METODO	+assumiProfessionista(professionista : Professionista) : void
DESCRIZIONE	Questo metodo permette l'assunzione dei professionisti.
PRE-CONDIZIONE	**context:** GestioneGestoreService::assumiProfessionista(professionista : Professionista) : void **pre:** professionista <> NULL
POST-CONDIZIONE	**context:** GestioneGestoreService:: assumiProfessionista(professionista : Professionista) **post:** Professionista.exists() = TRUE

NOME METODO	+rimuoviProfessionista(professionistaID : int) : String
DESCRIZIONE	Questo metodo permette la rimozione di un professionista
PRE-CONDIZIONE	**context:** GestioneGestoreService::rimuoviProfessionista(professionistaID : int) : String **pre:** professionistaID <> NULL
POST-CONDIZIONE	**context:** GestioneGestoreService::rimuoviProfessionista(professionistaID : int) : String **post:** result = "successo" Or result =

	“Errore durante la rimozione del professionista”

Gestione_Prenotazione

NOME CLASSE	**FasciaOrariaService**
METODI	+getFasciaOraria(professionistaID : int, data : LocalDate, orario : String) : FasciaOraria +updateFasciaOraria(fasciaOraria : FasciaOraria) : void
INVARIANTI	//

NOME METODO	+getFasciaOraria(professionistaID : int, data : LocalDate, orario : String) : Fascia Oraria
DESCRIZIONE	Questo metodo ricava la fascia oraria di ogni professionista, in base alla data selezionata.
PRE-CONDIZIONE	**context:** FasciaOrariaService::getFasciaOraria(professionistaID : int, data : Localdate, orario : String) : FasciaOraria **pre:** professionistaID <> NULL AND orario <> NULL AND data <> NULL
POST-CONDIZIONE	**context:** FasciaOrariaService::getFasciaOraria(professionistaID : int, data : Localdate, orario : String) : FasciaOraria

	post: FasciaOraria.data = data AND FasciaOraria.orario = orario

NOME METODO	+updateFasciaOraria(fasciaOraria : FasciaOraria) void
DESCRIZIONE	Questo metodo permette l'aggiornamento delle fasce orarie occupate.
PRE-CONDIZIONE	**context:** FasciaOrariaService::updateFasciaOraria(fasciaOraria : FasciaOraria) : void **pre:** FasciaOraria <> NULL
POST-CONDIZIONE	**context:** FasciaOrariaService::updateFasciaOraria(fasciaOraria : FasciaOraria) : void **post:** FasciaOrariaResult != fasciaOraria

NOME CLASSE	**PrenotazioneService**
METODI	+getSediByCitta(citta : String) : List<Sede> +getPrenotazioniAttive(SedeId : int) : List<Prenotazione> +getCittaDisponibili(sedi : List<Sede>) : Set<String> +getAllSedi() : List<Sede> +addPrenotazione(prenotazione : Prenotazione) : void +rimuoviPrenotazione(prenotazioneID : int) : String

INVARIANTI	//

NOME METODO	+getPrenotazioniAttive(SedeId : int) : List<Prenotazione>
DESCRIZIONE	Questo metodo mostra le sedi in base alla città selezionata.
PRE-CONDIZIONE	**context:** PrenotazioneService::getSediByCitta(citta : String) : List<Sede> **pre:** citta <> NULL
POST-CONDIZIONE	**context:** PrenotazioneService::getSediByCitta(citta : String) : List<Sede> **post:** result : List<Sede.isEmpty = FALSE

NOME METODO	+getSediByCitta(citta : String) : List<Sede>
DESCRIZIONE	Questo metodo mostra le sedi in base alla città selezionata.
PRE-CONDIZIONE	**context:** PrenotazioneService::getSediByCitta(citta : String) : List<Sede> **pre:** citta <> NULL
POST-CONDIZIONE	**context:** PrenotazioneService::getSediByCitta(citta : String) : List<Sede>

	post: result : List<Sede.isEmpty = FALSE

NOME METODO	+getCittaDisponibili(sedi : List<Sede>) : Set<String>
DESCRIZIONE	Questa metodo ricava tutte le città disponibili con una sede attiva.
PRE-CONDIZIONE	**context:** PrenotazioneService::getCittaDisponibili(sedi : List<Sede>) : Set<String> **pre:** sedi.isEmpty() = FALSE
POST-CONDIZIONE	**context:** PrenotazioneService::getCittaDisponibili(sedi : List<Sede>) : Set<String> **post:** Set<String> <> NULL

NOME METODO	+getAllSedi() : List<Sede>
DESCRIZIONE	Questo metodo ricava tutte le sedi attive
PRE-CONDIZIONE	**context:** PrenotazioneService::getAllSedi() : List<Sede> **pre:** Sede.Exists() = TRUE
POST-CONDIZIONE	**context:** PrenotazioneService::getAllSedi()

	: List<Sede> **post:** result.List<Sede>.isEmpty() = FALSE

NOME METODO	+addPrenotazione(prenotazione : Prenotazione) : void
DESCRIZIONE	Questo metodo permette l'aggiunta della prenotazione alla lista delle prenotazioni attive
PRE-CONDIZIONE	**context:** PrenotazioneService::addPrenotazione(prenotazione : Prenotazione) **pre:** Prenotazione <> NULL
POST-CONDIZIONE	**context:** PrenotazioneService::addPrenotazione(prenotazione : Prenotazione) **post:** List<Prenotazione>.isEmpty() = FALSE

NOME METODO	+rimuoviPrenotazione(prenotazioneID : int) : String
DESCRIZIONE	Questo metodo rimuove la prenotazione dalla lista delle prenotazioni attive di una sede.
PRE-CONDIZIONE	**context:**

	PrenotazioneService::rimuoviPrenotazione(prenotazioneID : int) **pre:** prenotazioneID <> NULL
POST-CONDIZIONE	**context:** PrenotazioneService::rimuoviPrenotazione(prenotazione : Prenotazione) **post:** List<Prenotazione>.isEmpty() = FALSE

NOME CLASSE	**SaloneService**
METODI	+getProfessionistiBySalone(saloneID : int) : List<Professionista> +getFasceOrarieByProfessionista(professionistaID : int) : Map<LocalDate, List<String>
INVARIANTI	//

NOME METODO	+getProfessionistiBySalone(saloneID : int) : List<Professionista>
DESCRIZIONE	Questo metodo ricerca tutti i professionisti per un dato salone.
PRE-CONDIZIONE	**context:** SaloneService::getProfessionistiBySalone(SaloneId : int) **pre:** SaloneID <> NULL
POST-CONDIZIONE	**context:**

	SaloneService::getProfessionistiBySalone(SaloneId : int) **post:** List<Professionista>.isEmpty() = FALSE

NOME METODO	+getFasceOrarieByProfessionista(ProfessionistaID : int) : Map<LocalDate, List<String>
DESCRIZIONE	Questo metodo recupera ed associa ad ogni professionista dato (dal metodo precedente) la sua fascia oraria lavorativa.
PRE-CONDIZIONE	**context:** SaloneService::getFasceOrarieByProfessionista(ProfessionistaID : int **pre:** ProfessionistaID <> NULL
POST-CONDIZIONE	**context:** SaloneService::getFasceOrarieByProfessionista(ProfessionistaID : int **post:** result.Map<LocalDate, List<String>.isEmpty() = FALSE

NOME CLASSE	**StoricoOrdiniService**
METODI	+getPrenotazioniByUsername(username : String) : List<Prenotazione> +getPrezzoByServizio(servizio : String) : double +getIndirizzoBySedeId(sedeID : Int) : String

INVARIANTI	//

NOME METODO	+getPrenotazioniByUsername(username : String) : List<Prenotazioni>
DESCRIZIONE	Questo metodo recupera tutte le prenotazioni di un dato utente.
PRE-CONDIZIONE	**context:** StoricoOrdiniService::getPrenotazioniByUsername(username : String) **pre:** username <> NULL
POST-CONDIZIONE	**context:** StoricoOrdiniService::getPrenotazioniByUsername(username : String) **post:** List<Prenotazioni>.isEmpty() = FALSE

NOME METODO	+getPrezzoByServizio(servizio : String) : double
DESCRIZIONE	Questo metodo recupera i prezzi in base al servizio indicato nella prenotazione.
PRE-CONDIZIONE	**context:** StoricoOrdiniService::getPrezzoByServizio(servizio : String) **pre:** servizio <> NULL
POST-CONDIZIONE	**context:** StoricoOrdiniService::getPrezzoByServizio

	(servizio : String) **post:** result.prezzo = servizio.prezzo)

NOME METODO	+getIndirizzoBySedeId(sedeID : int) : String
DESCRIZIONE	Ricava l'indirizzo della sede in cui c'è stata la prenotazione da mostrare nello storico ordini
PRE-CONDIZIONE	**context:** StoricoOrdiniService::getIndirizzoBySedeId(sedeID : int) **pre:** sedeID <> NULL
POST-CONDIZIONE	**context:** StoricoOrdiniService::getIndirizzoBySedeId(sedeID : int) **post:** resultString = sede.Indirizzo

Gestione_Catena

NOME CLASSE	**GestioneGestoreService**
METODI	+creaGestore(Ugs : UtenteGestoreSede) : boolean +assegnaSede(usernameUGS : String, sedeID : int) : void +licenziaGestore(usernameUGS : String) : void +getGestoriSenzaSede() : List<UtenteGestoreSede> +getGestoriConSede() : List <UtenteGestoreSede>

INVARIANTI	//

NOME METODO	+creaGestore(UGS : UtenteGestoreSede) : boolean
DESCRIZIONE	Questo metodo crea un Utente Gestore Sede.
PRE-CONDIZIONE	**context:** GestioneGestoreService::creaGestore(UGS : UtenteGestoreSede) **pre:** UGS <> NULL
POST-CONDIZIONE	**context:** GestioneGestoreService::creaGestore(UGS : UtenteGestoreSede) **post:** List<UtenteGestoreSede>.isEmpty() = FALSE

NOME METODO	+assegnaSede(usernameUGS : String, SedeID : int) : void
DESCRIZIONE	Questo metodo assegna un Utente Gestore Sede ad una data sede.
PRE-CONDIZIONE	**context:** GestioneGestoreService::assegnaSede(user nameUGS : String, sedeID : int) **pre:** usernameUGS <> NULL AND sedeID <> NULL
POST-CONDIZIONE	**context:** GestioneGestoreService::assegnaSede(user nameUGS : String, sedeID : int) **post:** UtenteGestoreSede.sedeID = sede.ID

NOME METODO	+licenziaGestore(usernameUGS : String)
DESCRIZIONE	Questo metodo rimuove un UtenteGestoreSede
PRE-CONDIZIONE	**context:** GestioneGestoreService::licenziaGestore(usernameUGS : String) **pre:** usernameUGS <> NULL
POST-CONDIZIONE	**context:** GestioneGestoreService::licenziaGestore(usernameUGS : String) **post:** UtenteGestoreSede.Exists() = FALSE

NOME METODO	+getGestoriSenzaSede() : List<UtenteGestoreSede>
DESCRIZIONE	Questo metodo ricava tutti i gestori ai quali non è stata assegnata ancora una sede.
PRE-CONDIZIONE	**context:** GestioneGestoreService::getGestoriSenzaSede() **pre:** List<UtenteGestoreSede>.isEmpty() = FALSE
POST-CONDIZIONE	**context:** GestioneGestoreService::getGestoriSenzaSede() **post:** result.List<UtenteGestoreSede>.get(UtenteGestoreSede.SedeID) == FALSE

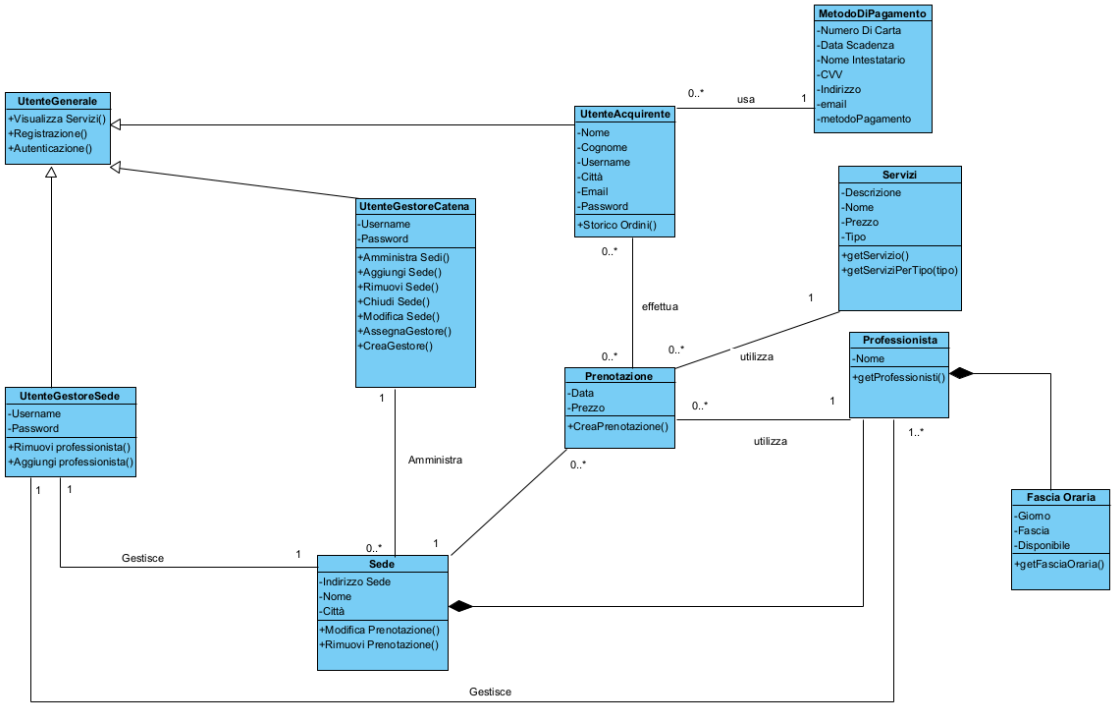
NOME METODO	+getGestoriConSede() : List<UtenteGestoreSede>
DESCRIZIONE	Questo metodo ricava tutti i gestori ai quali è stata assegnata una sede.
PRE-CONDIZIONE	**context:** GestioneGestoreService::getGestoriConSede() **pre:** List<UtenteGestoreSede>.isEmpty() = FALSE
POST-CONDIZIONE	**context:** GestioneGestoreService::getGestoriSenzaSede() **post:** result.List<UtenteGestoreSede>.get(UtenteGestoreSede.SedeID) == TRUE

NOME CLASSE	**GestioneSedeService**
METODI	+creaSede(sede : Sede) : int
INVARIANTI	//

NOME METODO	+getGestoriConSede() : List<UtenteGestoreSede>
DESCRIZIONE	Questo metodo crea una sede disponibile alle prenotazioni.
PRE-CONDIZIONE	**context:** GestioneSedeService::creaSede(sede : Sede) **pre:** Sede <> NULL.

POST-CONDIZIONE	**context:** GestioneSedeService::creaSede(sede : Sede) **post:** result = 1;

4. Class Diagram Ottimizzato



Come già anticipato nell'SDD, ci sono stati sostanziali cambiamenti nello schema progettato inizialmente, cambiamenti che sono stati guidati dalla necessità di semplificare l'implementazione successiva dell'applicativo:

- Rimossa l'associazione tra MetodoDiPagamento e Prenotazione, poiché quest'ultima puo' accedere alle informazioni riguardo il metodo di pagamento tramite l'UtenteAcquirente.
- Rimossa l'entità Catena, utilizzata come aggregatore di sede, poiché i metodi in essa contenuta sono metodi che sono utilzzati dall'UtenteGestoreCatena, e di conseguenza

vengono spostati in quest'ultima. Anche tutti le associazioni di Catena vengono trasferite all'UGC.

- Aggiunta l'entità FasciaOraria, associata al professionista, poiché permette una gestione semplificata degli orari assegnati ad un professionista.
- Associati (tramite l'associazione container), l'entità Professionista e l'entità Sede, con la Sede container, poiché un professionista non può esistere senza la sua sede.
- Rimozione dell'entità Promozione, non implementata per L'object Trade Off considerato (Rapidità di sviluppo e Funzionalità)
- Piccoli cambiamenti (rimozione di alcuni attributi ridondanti es: durata dall'entità Servizio, aggiunta di altri come 'e-mail' a MetodoDiPagamento per la gestione del pagamento con paypal, ecc..)

5. Glossario

Utente Generale	Rappresenta un utente non autenticato nel sistema, sono presenti quindi restrizioni nell'utilizzo di quest'ultimo.
Utente Acquirente	Rappresenta un utente autenticato nel sistema, che quindi puo' sfruttare le funzionalità riguardanti le prenotazioni e lo storico degli ordini.
Utente Gestore Sede	Rappresenta un utente registrato con il compito di gestire una singola sede dell'intera catena, esso può gestire le prenotazioni e i professionisti in salone.
Utente Gestore Catena	Rappresenta un utente registrato, che gestisce l'intera catena HairBeautyNow, ha il compito di amministrare le sedi, gli Utenti Gestori Sede e di creare e attivare Promozioni in tutta la catena.
Prenotazione	Rappresenta l'acquisto di un servizio con

	relative informazioni riguardo la sede, il professionista scelto e il servizio.
Franchise	Un sistema di collaborazione tra un produttore e un distributore, ove il primo cede al secondo la facoltà di distribuire il servizio con alcune definizioni contrattuali.