Università degli Studi di Salerno
Corso di Ingegneria del Software

CineNow
ODD: Object Design Document
Versione 1.0

Data: 15/12/2024

Coordinatore del progetto:

Nome	Matricola

Partecipanti:

Nome	Matricola
Emanuele Iovane	0512120565
Armando Vigliotti	0512117739
Antonio Caiazzo	0512117751

Scritto da:	Caiazzo Antonio, Iovane Emanuele, Vigliotti Armando

Revision History

Data	Versione	Descrizione	Autore
02/12/2024	0.1	Creazione dell'Object Design Document	Tutto il team
03/12/2024	0.2	Definizione del Packaging CineNow	Emanuele Iovane
07/12/2024	0.3	Creazione dell'Introduzione, Object design trade-offs, Interface documentation guidelines, Design Pattern.	Antonio Caiazzo
13/12/2024	0.4	Definizione contratti delle classi del sistema.	Armando Vigliotti
15/12/2024	1.0	Revisione per primo rilascio.	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 CineNow, completando e ampliando quanto definito nei precedenti documenti, che si sono concentrati prevalentemente sull'architettura e progettazione generale del sistema. Questo documento fornisce una descrizione tecnica e completa delle decisioni progettuali prese durante le fasi di analisi e design, specificando i principali trade-off considerati, le linee guida per la documentazione delle interfacce e l'identificazione dei Design Pattern adottati.

Nel corso del documento verranno inoltre definiti i packages, le interfacce delle classi e i diagrammi delle classi che rappresentano le principali decisioni implementative del sistema. Particolare attenzione sarà dedicata alla descrizione dettagliata delle operazioni, dei parametri e delle firme delle varie interfacce e classi, assicurando una coerenza con i sottosistemi individuati nel System Design Document (SDD_CineNow) e con i requisiti funzionali e non funzionali descritti nel Requirements Analysis Document (RAD_CineNow).

La funzionalità dell'Object Design Document è fornire un modello preciso e utilizzabile per implementare tutte le funzionalità identificate, favorendo una transizione fluida verso la fase di implementazione e garantendo la manutenibilità e l'espandibilità del sistema.

1.1. Object design trade-offs

Funzionalità Avanzate vs. Tempi di Sviluppo:

Come descritto nel RAD e nell'SDD, alcune funzionalità secondarie, come la cancellazione delle prenotazioni oppure l'inserimento del codice (OTP) di registrazione, non saranno implementate nella prima versione del sistema per rispettare i tempi di consegna. L'attenzione sarà focalizzata sulle funzionalità essenziali, quali la gestione delle prenotazioni, l'autenticazione degli utenti e la visualizzazione delle proiezioni. Le funzionalità avanzate verranno considerate in fasi di sviluppo successive.

Sicurezza vs. Efficienza:

Considerando i tempi stretti di sviluppo, verranno implementati sistemi di sicurezza essenziali, come l'autenticazione basata su username e password crittografati, e una gestione degli accessi in base ai ruoli definiti nel RAD (cliente, gestore sede, gestore catena). Tuttavia, per mantenere l'efficienza dello sviluppo, non saranno inclusi meccanismi avanzati come l'autenticazione a due fattori. Questo compromesso garantisce un livello di sicurezza adeguato per proteggere i dati sensibili, bilanciando al contempo la velocità e la semplicità di implementazione per rispettare le scadenze.

Comprensibilità vs. Tempo:

A causa dei tempi di sviluppo limitati, non verrà data priorità alla creazione di commenti esaustivi per ogni metodo o funzione. Tuttavia, saranno forniti commenti per le principali classi, interfacce e metodi critici al fine di garantire una comprensione di alto livello del sistema. Questa scelta consente di accelerare il processo di implementazione, mantenendo comunque una documentazione sufficiente per supportare le attività di sviluppo e manutenzione. Il compromesso consente di rispettare le scadenze stabilite sacrificando una documentazione approfondita, ma senza compromettere la leggibilità del codice per le parti essenziali del sistema.

1.2. Interface documentation guidelines

Nomi dei Package: I nomi dei package nel sistema CineNow 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* (es. **gestione_sale**, **gestione_utenti**).

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

Classi DAO: Devono terminare con il suffisso **DAO** per indicare il loro ruolo di accesso ai dati (ad esempio, **PrenotazioneDAO**, **UtenteDAO**).

Classi Bean: Devono terminare con il suffisso Bean, utilizzato per rappresentare entità del dominio (ad esempio, **UtenteBean**, **FilmBean**).

Classi di gestione: Devono indicare chiaramente la funzionalità principale(ad esempio, **GestionePrenotazioniControl**).

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** (ad esempio, **CatalogoServlet)**.

Nomi dei Metodi: I metodi devono avere nomi descrittivi che rappresentino chiaramente l'operazione eseguita. I nomi devono iniziare con una lettera minuscola e seguire il formato *camelCase*(ad esempio, **calcolaTotale()**, **aggiungiProiezione()**).

Nomi delle Variabili: I nomi delle variabili devono essere descrittivi e seguire il formato *camelCase*. Devono iniziare con una lettera minuscola. Per nomi composti, ogni parola interna deve iniziare con una lettera maiuscola (es. **numeroPosti**, **nomeFilm**). Possono essere utilizzate variabili temporanee ed esse possono essere abbreviate (es. **i**), ma devono essere limitate al contesto in cui sono utilizzate come ad esempio in un ciclo **for**.

Nomi delle Costanti: I nomi delle costanti devono essere scritti in maiuscolo, con le parole separate da underscore (_) (ad esempio, **POSTI_DISPONIBILI=10).**

Nomi delle Eccezioni: Le eccezioni devono avere nomi descrittivi che indichino chiaramente il problema che segnalano. Devono terminare con il suffisso **Exception**(ad esempio, **PostoNonDisponibileException**, **UtenteNonAutorizzatoException)**.

Nomi delle interfacce: Le interfacce devono seguire il formato *PascalCase*, come le classi, e terminare con il suffisso **Interface**. Devono rappresentare chiaramente il comportamento che definiscono(ad esempio, **PrenotazioneInterface**).

Nomi delle JSP: I file JSP devono seguire il formato *camelCase* e riflettere chiaramente il contenuto della pagina(ad esempio, **loginView.jsp**, **storicoOrdiniView.jsp)**.

Nomi delle classi che implementano Strategy: Le classi che implementano il Pattern Strategy, devono seguire il formato *PascalCase ,* e devono terminare con la parola **Validator** (ad esempio, **PasswordValidator**)

Organizzazione delle Componenti:

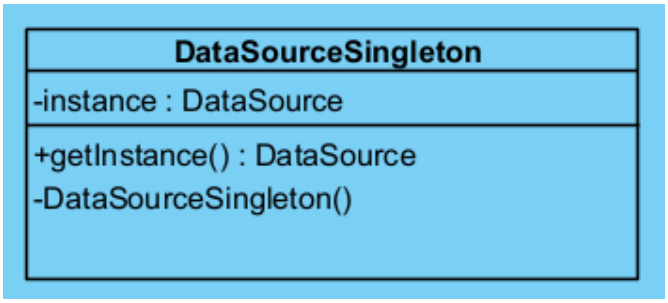
- **Classi e Package:** Tutte le classi che realizzano un sottosistema devono essere racchiuse nello stesso package Java, organizzate in modo logico in base alla loro funzione in modo da mantenere un sistema coeso.
- **Risorse Statiche**: Fogli di stile, script e immagini devono essere organizzati nella directory **resources**, con sottocartelle per ciascun tipo di file (es. r**esources/css**, **resources/js**, **resources/images**).

1.3. Design Pattern

Per implementare le funzionalità del sistema CineNow, sono stati adottati due design pattern: **Singleton Pattern** e **Strategy Pattern**. Di seguito vengono riportate le motivazioni che hanno portato all'adozione dei suddetti pattern nel contesto dell'applicazione.

Singleton Pattern
L'applicazione necessita di un controllo centralizzato dell'accesso alla risorsa di connessione al database essendo una risorsa condivisa. L'utilizzo di un Singleton Pattern quindi risulta essere utile, per la limitazione,e l'accesso concorrente alla risorsa DataSource. L'utilizzo del pattern Singleton garantisce quindi l'esistenza di un'unica istanza di DataSource. In questo modo quindi: si previene la creazione simultanea di più connessioni ridondanti,si gestisce con maggior chiarezza il ciclo di vita della connessione condivisa e si semplifica il coordinamento dell'accesso ai dati.

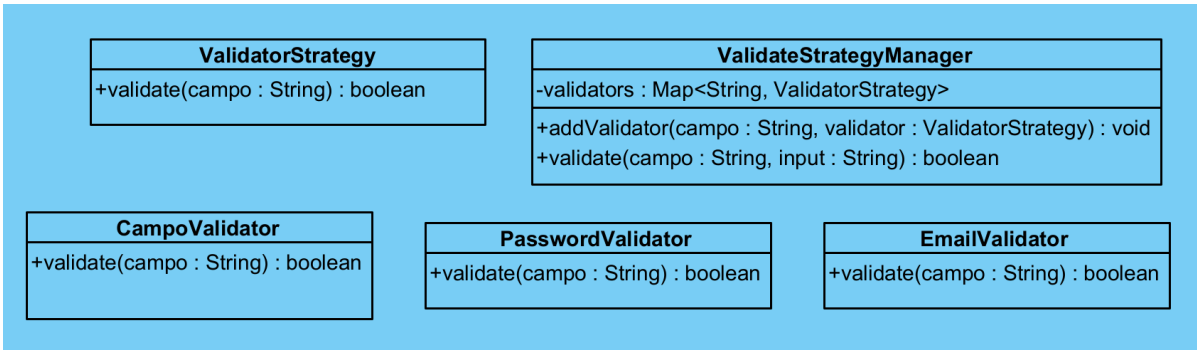
La classe DatabaseSourceSingleton è la specializzazione del pattern Singleton per l'applicazione CineNow. presenta un attributo statico, di tipo DataSource: instance. L'accesso al DataSource avviene quindi ottenendo l'attributo instance tramite il metodo getConnection(). La connessione verrà poi inserita nel contesto dell'applicazione web tramite l'uso della classe MainContext.



Strategy Pattern

I requisiti di validazione dei campi input, come ad esempio email, password o altri tipi di dati, possono variare spesso e soprattutto richiedere frequenti modifiche o ampliamenti. Si è optato quindi per l'introduzione dello Strategy Pattern. Esso consente di definire un'interfaccia comune e di implementare differenti strategie di validazione, facilmente intercambiabili. Grazie a questo approccio quindi è possibile gestire con maggiore flessibilità e modularità l'introduzione di nuove logiche di validazione senza alterare l'architettura esistente, e riutilizzando quella necessaria per più istanze di classi che ne richiedono una, semplificando inoltre la manutenibilità e il testing. Per la nostra applicazione, sono stati definiti quindi:

- ValidatorStrategy : interfaccia che espone il metodo per la validazione del campo
- ValidationManager: Classe che gestisce ogni Validator che implementa l'interfaccia e che può essere utilizzato.
- Classi per l'implementazione dei Validator su tipi di dati differenti.



1.4. Definizioni, acronimi e abbreviazioni

FR: Requisito Funzionale.
NFR: Requisito Non Funzionale.
UTC: Utente Cliente.
UGS: Utente Gestore Sede.
UGC: Utente Gestore Catena.

1.5. Riferimenti

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

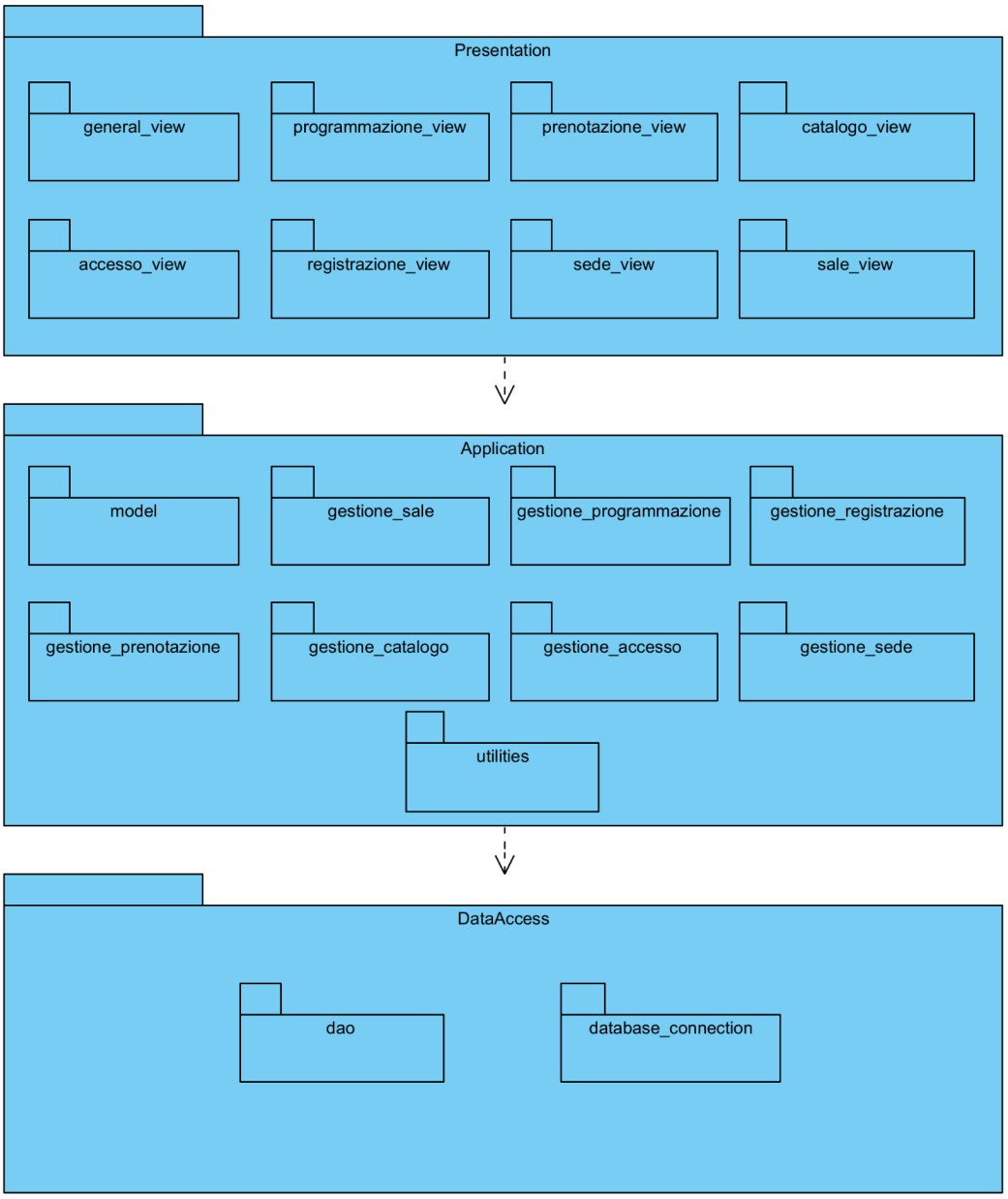
- RAD_CineNow (Requirements Analysis Document): Sono descritte le funzionalità individuate in fase di analisi.
- SDD_CineNow (System Design Document): Contiene una descrizione completa dell'architettura del sistema, inclusi i sottosistemi individuati ed i servizi forniti da ciascun sottosistema.
- ClassDiagram_CineNow: Mostra il class diagram, per il sistema CineNow.

2. PACKAGES

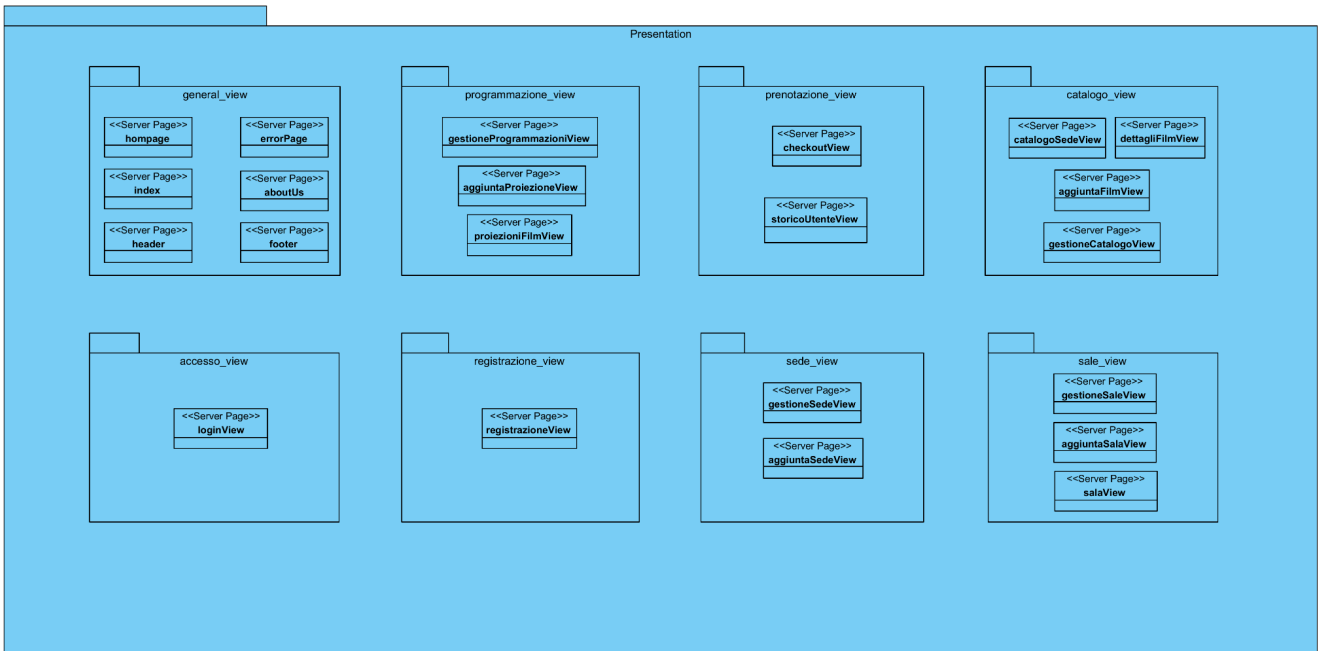
2.1 Panoramica

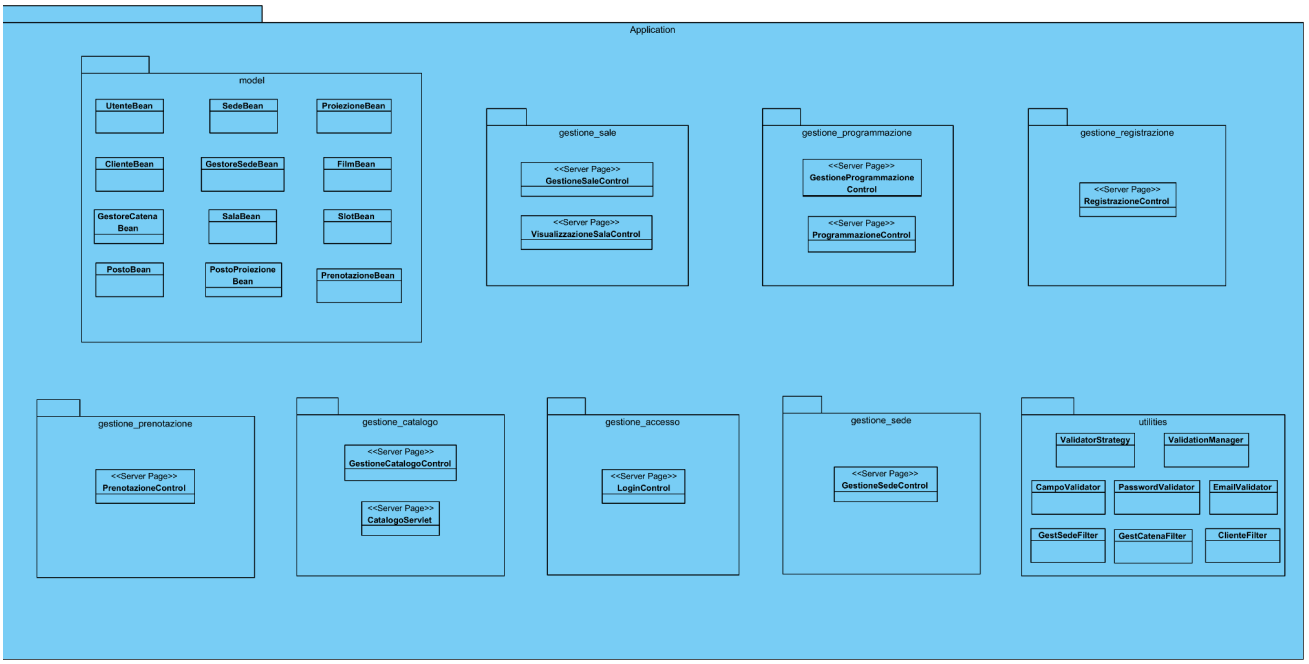
In questa sezione, verrà prima mostrata una panoramica generale del packaging completo del sistema CineNow. Dopodiché nella sezione 2.3 si entrerà più nel dettaglio, dei package dei layer Application, DataAccess e Presentation. Nella sezione 2.4 si analizzano i singoli package e le interfacce delle classi in esse contenute.

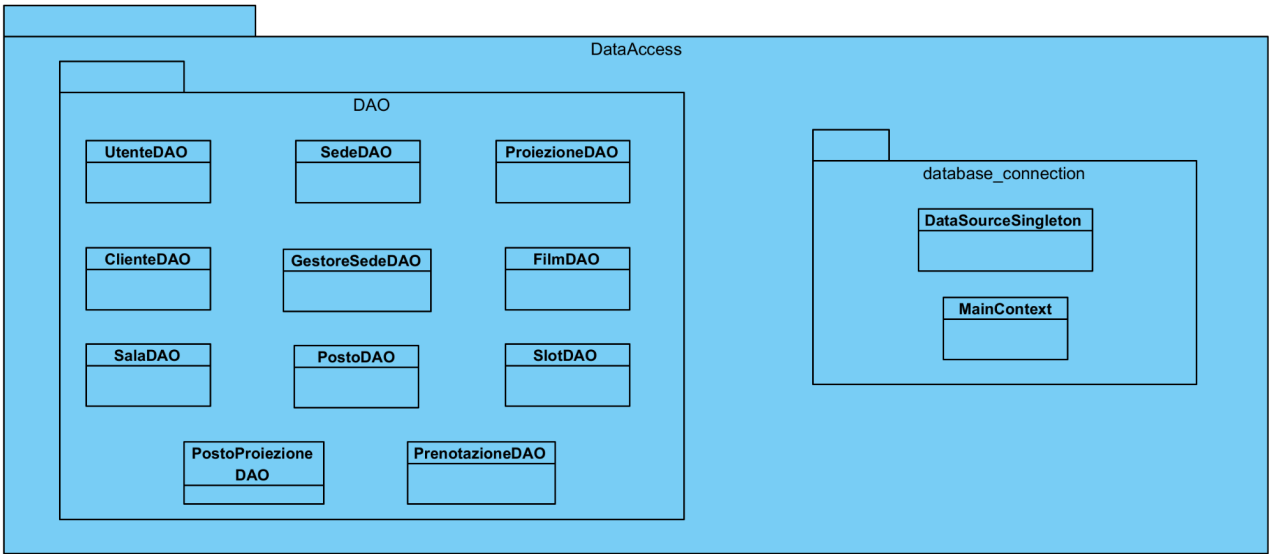
2.2 Packaging del sistema



2.3 Packaging dei layer



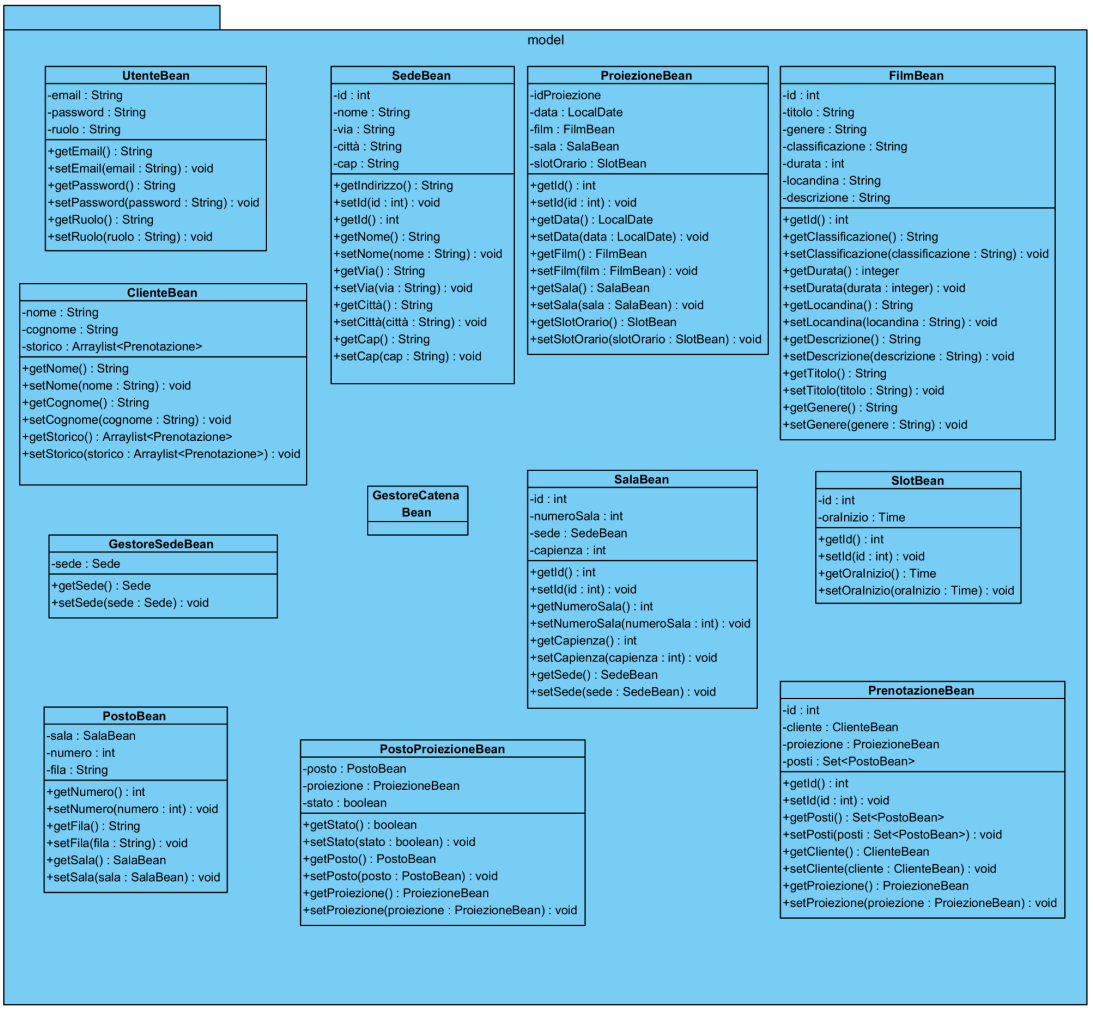



DataAccess
DAO
UtenteDAO
SedeDAO
ProiezioneDAO
ClienteDAO
GestoreSedeDAO
FilmDAO
SalaDAO
PostoDAO
SlotDAO
PostoProiezione DAO
PrenotazioneDAO
database_connection
DataSourceSingleton
MainContext

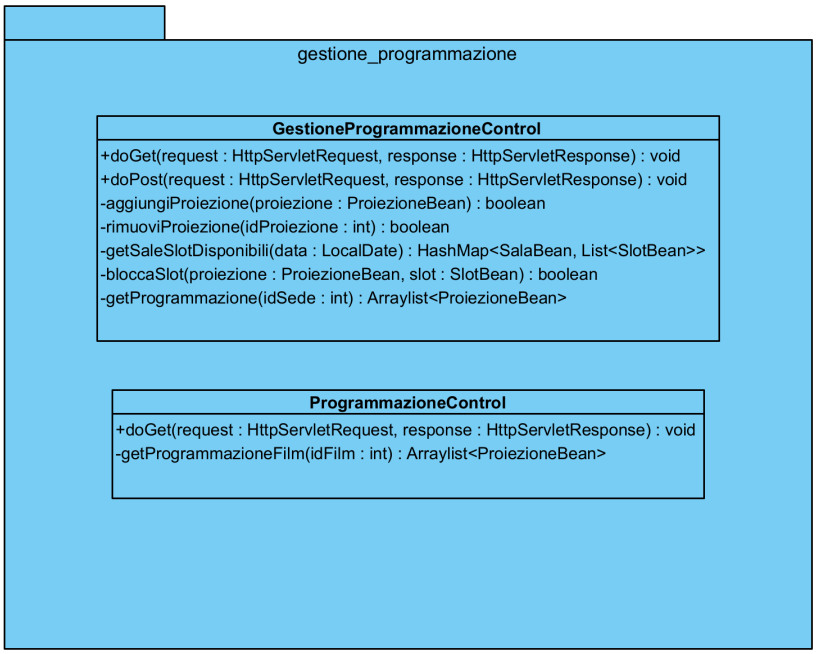
2.4 Dettaglio dei singoli package

2.4.1 Package Application

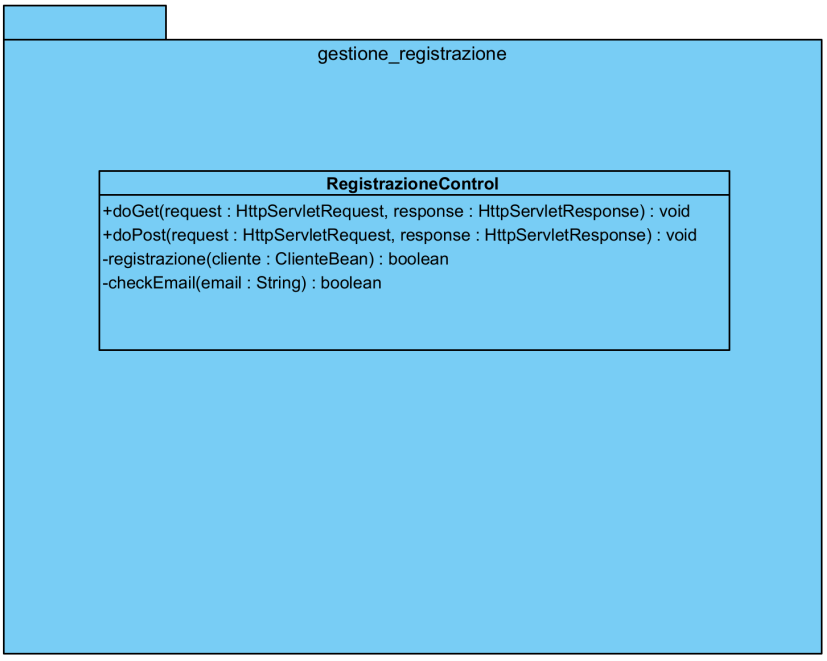
Package Application: model



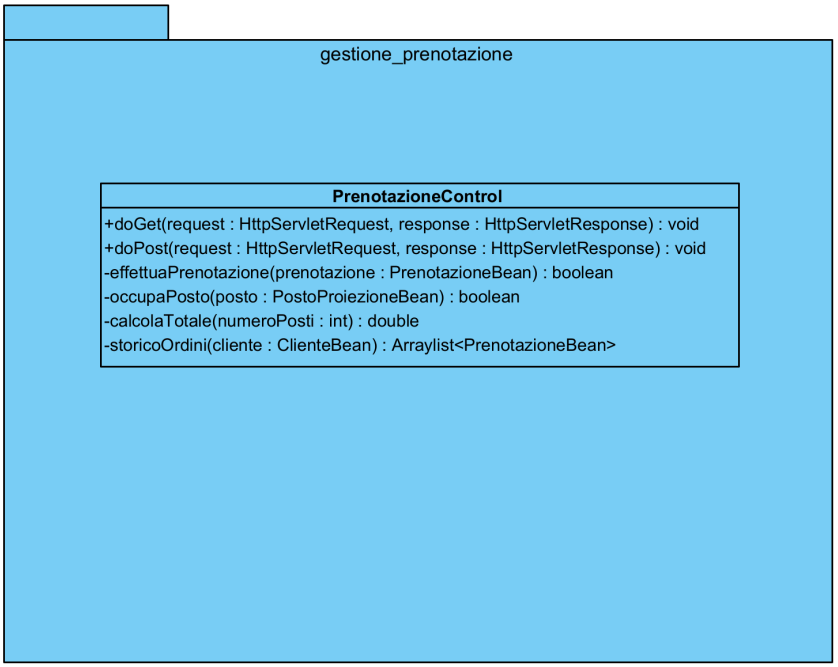
Package Application: gestione_programmazione



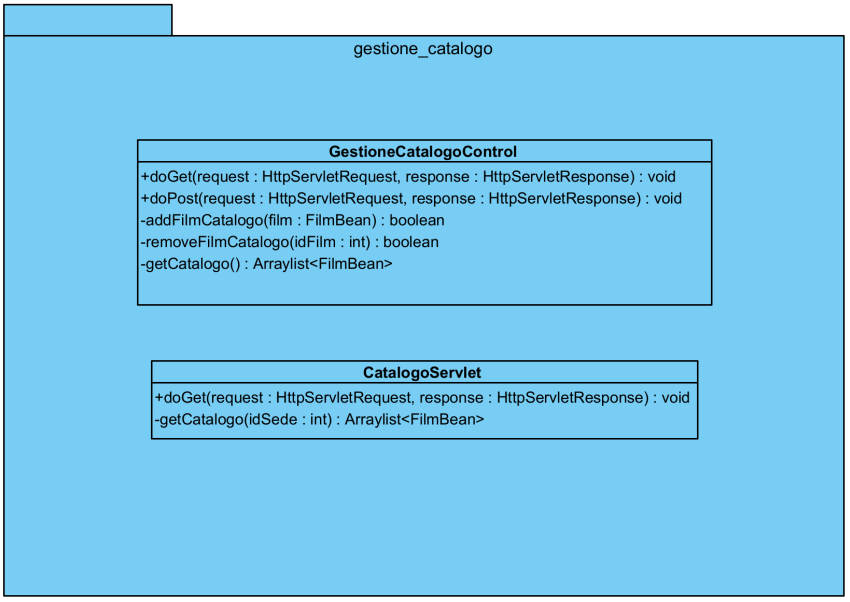
Package Application: gestione_registrazione



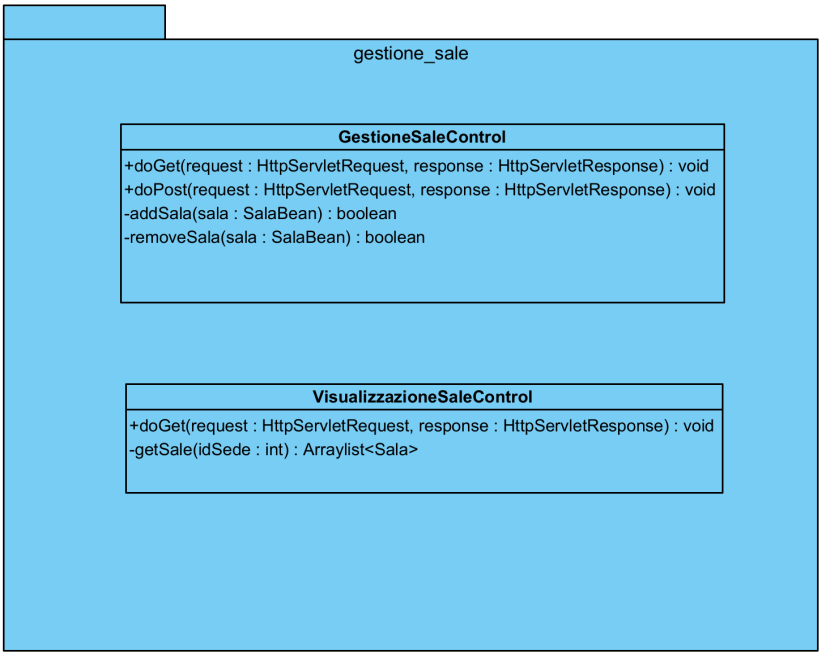
Package Application: gestione_prenotazione



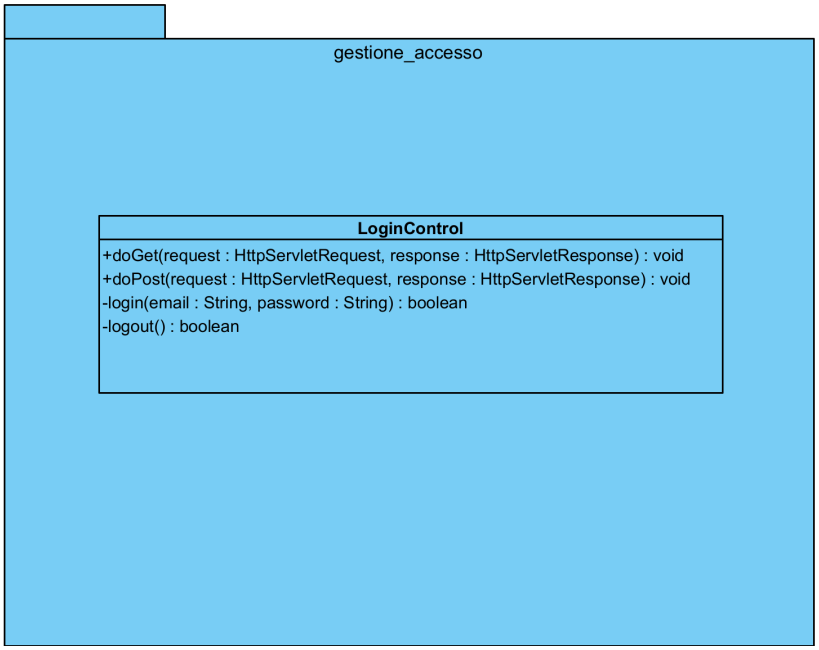
Package Application: gestione_catalogo



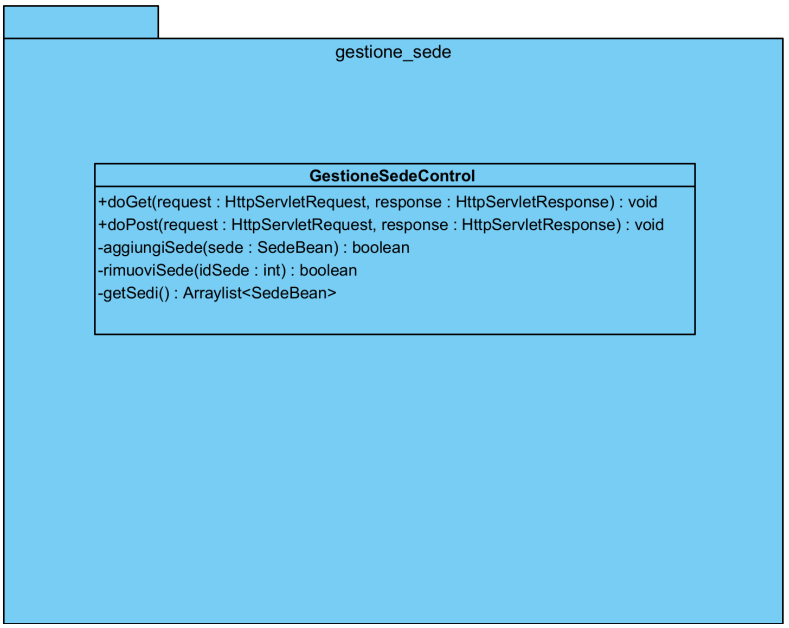
Package Application: gestione_sale



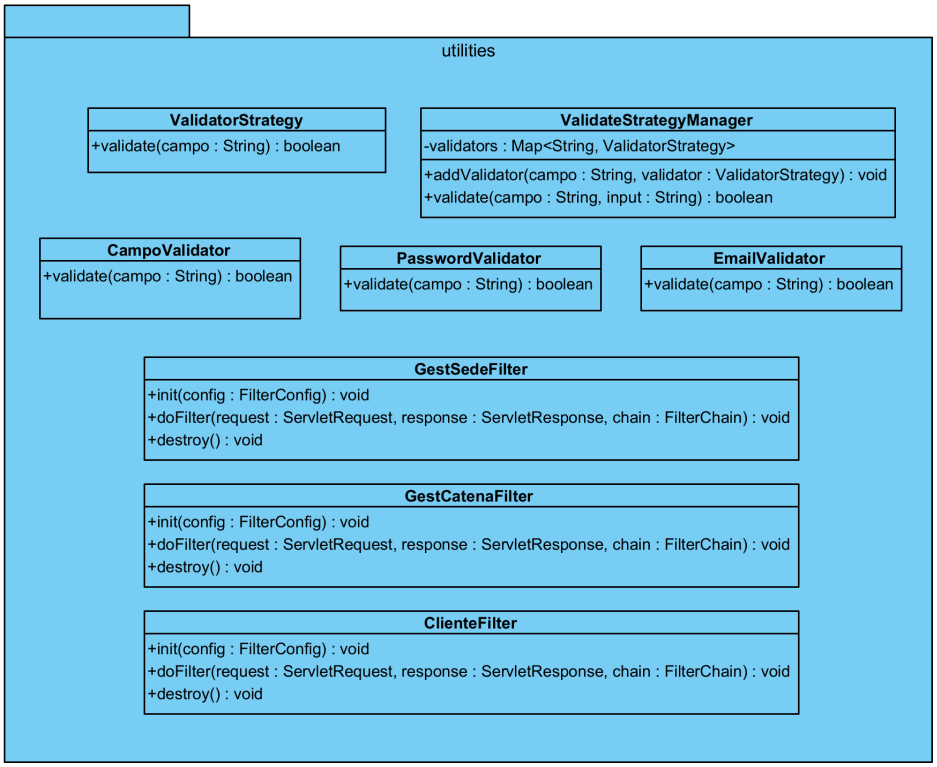
Package Application: gestione_accesso



Package Application: gestione_sede

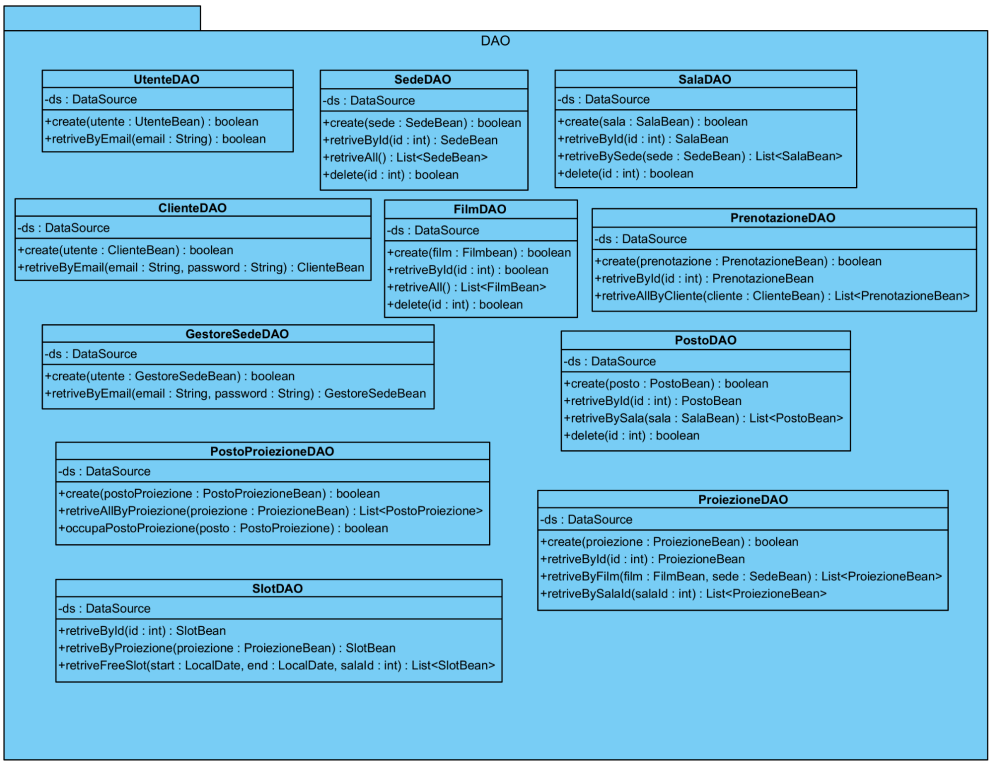


Package Application: utilities

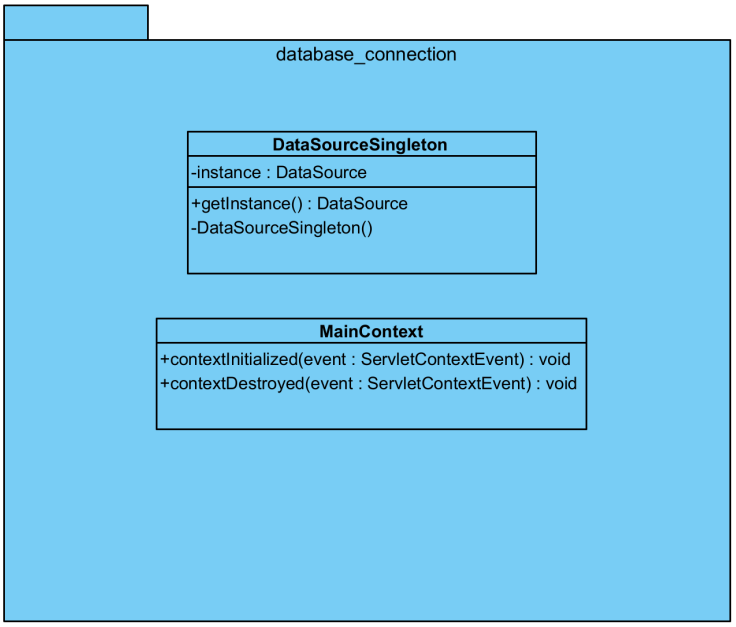


2.4.2 Package DataAccess

Package DataAccess: DAO



Package DataAccess: database_connection



3. CLASS INTERFACES

3.1 Panoramica

Di seguito, in dettaglio, vengono mostrate le classi e i loro metodi, nonché i contratti definiti tramite OCL. Per il class diagram completo, visitare: ClassDiagram_CineNow .

3.2 DataAccess

Package: DAO

NOME CLASSE	**UtenteDAO**
METODI	+create(utente : UtenteBean) : boolean +retriveByEmail(email : String) : UtenteBean
INVARIANTI	

NOME METODO	+create(utente : UtenteBean) : boolean
DESCRIZIONE	Questo metodo consente di inserire un nuovo utente
PRE-CONDIZIONE	**context:** UtenteDAO::create(utente: UtenteBean) **pre:** utente!= NULL AND utente.email != NULL AND utente.password!=null AND Database.utente->!exists(Utente | Utente.email == utente.email)
POST-CONDIZIONE	**context:** UtenteDAO::create(utente: UtenteBean) **post:** Database.Utente->include(utente)

NOME METODO	+retriveByEmail(email : String) : UtenteBean
DESCRIZIONE	Questo metodo permette di ottenere l'utente associato alla email
PRE-CONDIZIONE	**context:** UtenteDAO::retriveByEmail(email: String): UtenteBean **pre:** email != NULL
POST-CONDIZIONE	**context:** UtenteDAO::retriveByEmail(email: String): UtenteBean **post:** result = Database.Utente->select(Utente | Utente.email == email)

NOME CLASSE	**SedeDAO**
METODI	+create(sede: SedeBean) : boolean +retriveById(id : int) : SedeBean +retriveAll() : List<SedeBean> +delete(id : int) : boolean
INVARIANTI	

NOME METODO	+create(sede: SedeBean) : boolean
DESCRIZIONE	Questo metodo permette di aggiungere una nuova sede
PRE-CONDIZIONE	**context:** SedeDAO::create(sede: SedeBean): boolean **pre:** sede!=NULL AND sede.nome != NULL AND sede.nome != "" AND sede.via != NULL AND

	sede.via != “” AND sede.cap != NULL AND sede.cap.length() == 5 AND sede.città != NULL AND sede.città != “”
POST-CONDIZIONE	**context:** SedeDAO::create(sede: SedeBean): boolean **post:** Database.Sede->include(sede)

NOME METODO	+retriveById(id : int) : SedeBean
DESCRIZIONE	Questo metodo permette di ottenere una sede in base all’id
PRE-CONDIZIONE	**context:** SedeDAO::retriveById(id : int) : SedeBean **pre:** id != NULL AND id>0
POST-CONDIZIONE	**context:** SedeDAO::retriveById(id : int) : SedeBean **post:** result = Database.Sede->select(Sede | Sede.id == id) AND result!=null

NOME METODO	+retriveAll() : List<SedeBean>
DESCRIZIONE	Questo metodo permette di ottenere tutte le sedi presenti nel database
PRE-CONDIZIONE	**context:** SedeBean::retriveAll() : List<SedeBean> **pre:**
POST-CONDIZIONE	**context:** SedeBean::retriveAll() : List<SedeBean> **post:** result = Database.Sede

NOME METODO	+delete(id : int) : boolean
DESCRIZIONE	Questa operazione permette al Gestore Catena di eliminare una sede
PRE-CONDIZIONE	**context:** SedeDAO::delete(id : int) : boolean **pre:** Database.Sede->exists(Sede | Sede.id == id)
POST-CONDIZIONE	**context:** SedeDAO::delete(id : int) : boolean **post:** Database.Sede->!exists(Sede | Sede.id == id)

NOME CLASSE	**SalaDAO**
METODI	+create(sala: SalaBean) : boolean +retriveById(id: int) : SalaBean +retiveBySede(SedeId: int) : List<SalaBean> +delete(id : int) : boolean
INVARIANTI	

NOME METODO	+create(sala: SalaBean) : boolean
DESCRIZIONE	Questo metodo permette l'aggiunta di una nuova sala dal Gestore Sede
PRE-CONDIZIONE	**context:** SalaDAO::create(sala: SalaBean) : boolean **pre:** sala.numero != NULL AND sala.capienza != NULL AND sala.capienza > 0 sala.id_sede != NULL
POST-CONDIZIONE	**context:** SalaDAO::create(sala: SalaBean) : boolean **post:** Database.Sala->include(sala)

NOME METODO	+retriveById(id: int) : SalaBean
DESCRIZIONE	Questo metodo permette di ottenere una sala tramite il suo id
PRE-CONDIZIONE	**context:** SalaDAO::retriveById(id: int) : SalaBean **pre:** id != NULL AND id>0
POST-CONDIZIONE	**context:** SalaDAO::retriveById(id: int) : SalaBean **post:** result = Database.Sala->select(Sala | Sala.id == id)

NOME METODO	+retiveBySede(SedeId: int) : List<SalaBean>
DESCRIZIONE	Questo metodo permette di ottenere tutte le sale di un sede
PRE-CONDIZIONE	**context:** retiveBySede(SedeId: int) : List<SalaBean> **pre:** SedeId != NULL AND SedeId>0
POST-CONDIZIONE	**context:** retiveBySede(SedeId: int) : List<SalaBean> **post:** result = Database.Sala->select(Sala| sala.id_sede = SedeId)

NOME METODO	+delete(id : int) : boolean
DESCRIZIONE	Questo metodo permette al Gestore Sede di eliminare una sala
PRE-CONDIZIONE	**context:** delete(id : int) : boolean **pre:** Database.Sala->exists(Sala | Sala.id == id)
POST-CONDIZIONE	**context:** delete(id : int) : boolean

	post: Database.Sala->!exists(Sala | Sala.id == id)

NOME CLASSE	**ClienteDAO**
METODI	+create(utente: ClienteBean) : boolean +retriveByEmail(email: String): ClienteBean
INVARIANTI	

NOME METODO	+create(utente: ClienteBean) : boolean
DESCRIZIONE	Questo metodo consente di inserire un nuovo cliente
PRE-CONDIZIONE	**context:** ClienteDAO::create(utente: ClienteBean) : boolean **pre:** cliente!= NULL AND cliente.email != NULL AND cliente.password!=null AND cliente.email.validate() AND cliente.password.validate() AND Database.Cliente->!exists(Cliente | Cliente.email == cliente.email)
POST-CONDIZIONE	**context:** ClienteDAO::create(utente: ClienteBean) : boolean **post:** Database.Cliente->exists(Cliente)

NOME METODO	+retriveByEmail(email: String): ClienteBean
DESCRIZIONE	Questo metodo permette di ottenere il cliente associato all'email
PRE-CONDIZIONE	**context:** ClienteDAO:: retriveByEmail(email: String): ClienteBean

	pre: email != NULL
POST-CONDIZIONE	**context:** ClienteDAO::retriveByEmail(email: String): ClienteBeanb **post:** result = Database.Cliente->select(Cliente | Cliente.email = email)

NOME CLASSE	**FilmDAO**
METODI	+create(film: FilmBean) : boolean +retriveById(id: int) : boolean +retriveAll(): List<FilmBean> +delete(id: int) : boolean
INVARIANTI	

NOME METODO	+create(film: FilmBean) : boolean
DESCRIZIONE	Questo metodo permette all'utente Gestore Catena di aggiungere un film al catalogo
PRE-CONDIZIONE	**context:** FilmDAO::create(film: FilmBean) : boolean **pre:** film != NULL AND film.titolo != NULL AND film.titolo != "" AND film.genere != NULL AND film.genere != "" AND film.classificazione != NULL AND film.classificazione != "" AND film.durata != NULL AND film.durata > 52 (minuti) AND film.locandina != NULL AND film.locandina != "" AND film.descrizione != NULL film.descrizione != ""

POST-CONDIZIONE	**context:** FilmDAO::create(film: FilmBean) : boolean **post:** Database.Film->include(film)

NOME METODO	+retriveById(id: int) : boolean
DESCRIZIONE	Questo metodo permette di ottenere il film in base all'id
PRE-CONDIZIONE	**contex:** FilmDAO::retriveById(id: int) : boolean **pre:** id!= NULL AND id > 0
POST-CONDIZIONE	**contex:** FilmDAO::retriveById(id: int) : boolean **post:** result = Database.Film->select(Film | Film.id = id)

NOME METODO	+retriveAll(): List<FilmBean>
DESCRIZIONE	Questo metodo permette di ottenere tutta la lista di Film presenti nel Database
PRE-CONDIZIONE	**context:** FilmDAO::retriveAll(): List<FilmBean> **pre:**
POST-CONDIZIONE	**context:** FilmDAO::retriveAll(): List<FilmBean> **post:** result = Database.Film

NOME METODO	+delete(id: int) : boolean
DESCRIZIONE	Questo metodo permette al Gestore Catena di eliminare un film dal catalogo
PRE-CONDIZIONE	**context:** FilmDAO::delete(id: int) : boolean

	pre: Database.Film->exists(Film| Film.id = id)
POST-CONDIZIONE	**context:** FilmDAO::delete(id: int) : boolean **post:**Database.Film->!exists(Film|Film.id = id)

NOME CLASSE	**PrenotazioneDAO**
METODI	+create(prenotazione: PrenotazioneBean) : boolean +retriveById(id: int): prenotazioneBean +retriveAllByCliente(clienteId: int): List<PrenotazioneBean>
INVARIANTI	

NOME METODO	+create(prenotazione: PrenotazioneBean) : boolean
DESCRIZIONE	Questo metodo permette di creare una nuova prenotazione
PRE-CONDIZIONE	**context:** PrenotazioneDAO::create(prenotazione: PrenotazioneBean) : boolean **pre:** prenotazione != NULL AND prenotazione.email_cliente != NULL AND prenotazione.id_proiezione != NULL
POST-CONDIZIONE	**context:** PrenotazioneDAO::create(prenotazione: PrenotazioneBean) : boolean **post:** Database.Prenotazione->include(prenotazione)

NOME METODO	+retriveById(id: int): prenotazioneBean

DESCRIZIONE	Questo metodo permette di ottenere una prenotazione in base all'id
PRE-CONDIZIONE	**context:** PrenotazioneDAO::retriveById(id: int): prenotazioneBean **pre:** id != NULL AND id > 0
POST-CONDIZIONE	**context:** PrenotazioneDAO::retriveById(id: int): prenotazioneBean **post:** Database.Prenotazione->select(Prenotazione|Prenotazione.id = id)

NOME METODO	+retriveAllByCliente(clienteId: int): List<PrenotazioneBean>
DESCRIZIONE	Questo metodo permette di ottenere tutte le prenotazioni effettuate dal cliente
PRE-CONDIZIONE	**context:** PrenotazioneDAO::retriveAllByCliente(cliente Id: int): List<PrenotazioneBean> **pre:** DA CAMBIARE
POST-CONDIZIONE	**context:** PrenotazioneDAO::retriveAllByCliente(email_ cliente: int): List<PrenotazioneBean> **post:** result = Database.Prenotazione(Prenotazione | Prenotazione.email_cliente = email_cliente)

NOME CLASSE	**GestoreSedeDAO**
METODI	+create(utente: GestoreSedeBean) : boolean +retriveByEmail(email: String): GestioneSedeBean
INVARIANTI	

NOME METODO	+create(utente: GestoreSedeBean) : boolean
DESCRIZIONE	Questo metodo permette di inserire un nuovo Gestore Sede
PRE-CONDIZIONE	**context:** GestoreSedeDAO::create(utente: GestoreSedeBean) : boolean **pre:** utente != NULL AND utente.email != NULL AND utente.email.validate() AND utente.id_sede != NULL
POST-CONDIZIONE	**context:** GestoreSedeDAO::create(utente: GestoreSedeBean) : boolean **post:** Database.Gest_Sede->include(utente)

NOME METODO	+retriveByEmail(email: String): GestioneSedeBean
DESCRIZIONE	Questo metodo permette di ottenere il Gestore Sede associato alla email
PRE-CONDIZIONE	**context:** GestoreSedeDAO::retriveByEmail(email: String): GestioneSedeBean **pre:** emil != NULL
POST-CONDIZIONE	**context:** GestoreSedeDAO::retriveByEmail(email: String): GestioneSedeBean **post:** result = Database.Gest_Sede->select(Gest_Sede | Gest_Sede.email = email)

NOME CLASSE	**ProiezioneDAO**

METODI	+create(proiezione: ProiezioneBean) : boolean +retriveById(id: int) : ProiezioneBean +retriveByFilmId(filmId: int) : List<ProiezioneBean> +retriveBySalaId(salaId: int) : List<ProiezioneBean>
INVARIANTI	

NOME METODO	+create(proiezione: ProiezioneBean) : boolean
DESCRIZIONE	Questo metodo permette al Gestore Sede di aggiungere una nuova proiezione
PRE-CONDIZIONE	**context:** ProiezioneDAO::create(proiezione: ProiezioneBean) : boolean **pre:** proiezione != NULL AND proiezione.data != NULL AND proiezione.id_film != NULL AND proiezione.id_sala != NULL AND proiezione.id_orario != NULL
POST-CONDIZIONE	**context:** create(proiezione: ProiezioneBean) : boolean **post:** Database.Proiezione->include(proiezione)

NOME METODO	+retriveById(id: int) : ProiezioneBean
DESCRIZIONE	Questo metodo permette di ottenere la proiezione in base all'id
PRE-CONDIZIONE	**context:** ProiezioneDAO::retriveById(id: int) : ProiezioneBean **pre:** id != NULL AND id > 0
POST-CONDIZIONE	**context:** ProiezioneDAO::retriveById(id: int) : ProiezioneBean

	post: result = Database.Proiezione->select(Proiezione | Proiezione.id = id)

NOME METODO	+retriveByFilmId(filmId: int) : List<ProiezioneBean>
DESCRIZIONE	Questo metodo permette di ottenere una lista di proiezioni del film
PRE-CONDIZIONE	**context:** ProiezioneDAO::retriveByFilmId(filmId: int) : List<ProiezioneBean> **pre:** filmId != NULL AND filmId > 0
POST-CONDIZIONE	**context:** ProiezioneDAO::retriveByFilmId(filmId: int) : List<ProiezioneBean> **post:** result = Database.Proiezione(Proiezione | Proiezione.id_film = filmId)

NOME METODO	+retriveBySalaId(salaId: int) : List<ProiezioneBean>
DESCRIZIONE	Questo metodo permette di ottenere la lista di proiezione di quella determinata sala
PRE-CONDIZIONE	**context:** PrenotazioneDAO::retriveBySalaId(salaId: int) : List<ProiezioneBean> **pre:** salaId != NULL AND salaId > 0
POST-CONDIZIONE	**context:** PrenotazioneDAO::retriveBySalaId(salaId: int) : List<ProiezioneBean> **post:** result = Database.Proiezione->select(Proiezione | Proiezione.id_sala = salaId)

NOME CLASSE	PostoDAO
METODI	+create(posto: PostoBean): boolean +retriveById(id: int): SalaBean +retriveBySala(idSala): List<PostoBean> +delete(id: int): boolean
INVARIANTI	

NOME METODO	+create(posto: PostoBean): boolean
DESCRIZIONE	Questo metodo permette di creare un posto all'interno della sala
PRE-CONDIZIONE	**context:** PostoDAO::create(posto: PostoBean): boolean **pre:** posto != NULL AND posto.id_sala != NULL AND posto.id_sala > 0 NAD posto.numero != NULL AND posto.fila != NULL AND Database.Posto->!exists(Posto | Posto.numero = posto.numero AND Posto.fila = posto.fila)
POST-CONDIZIONE	**context:** create(posto: PostoBean): boolean **post:** Database.Posto->include(posto)

NOME METODO	+retriveById(id: int): SalaBean
DESCRIZIONE	Questo metodo restituisce la sala in cui è situato il posto
PRE-CONDIZIONE	**context:** PostoDAO::retriveById(id: int): SalaBean **pre:** id != NULL AND id > 0
POST-CONDIZIONE	**context:** retriveById(id: int): SalaBean

	post: result = Database.Sala->select(Sala | Sala.id = id)

NOME METODO	+retriveBySala(idSala): List<PostoBean>
DESCRIZIONE	Questo metodo restituisce una lista di posti della sala
PRE-CONDIZIONE	**context:** PostoDAO::retriveBySala(idSala): List<PostoBean> **pre:** idSala != NULL AND idSala > 0
POST-CONDIZIONE	**context:** PostoDAO::retriveBySala(idSala): List<PostoBean> **post:** result = Database.Posto->select(Posto | Posto.id_sala = idSala)

NOME METODO	+delete(id: int): boolean
DESCRIZIONE	Questo metodo permette al Gestore Sede di eliminare il posto
PRE-CONDIZIONE	**context:** Posto::delete(id: int): boolean **pre:** id != NULL AND Database.Posto->exists(Posto | Posto.id_sala = id)
POST-CONDIZIONE	**context:** Posto::delete(id: int): boolean **post:** Database.Posto->!exists(Posto | Posto.id_sala = id)

NOME CLASSE	**PostoProiezioneDAO**
METODI	+create(postoProiezione: PostoProiezioneBean): boolean +retriveAllByProiezione(proiezioneId: int) :

	List<PostoProiezione> +occupaPostoProiezione(posto: PostoProiezione): boolean
INVARIANTI	

NOME METODO	+create(postoProiezione:PostoProiezioneBean): boolean
DESCRIZIONE	Questo metodo permette di inserire il posto della prenotazione
PRE-CONDIZIONE	**context:** PostoProiezioneDAO::create(postoProiezione: PostoProiezioneBean): boolean **pre:** PostoProiezione != NULL AND PostoProiezione.id_sala != NULL AND Database.Sala->exists(Sala | Sala.id_sala = id_sala) AND PostoProiezione.fila != NULL AND Database.Posto->exists(Posto | Posto.fila = fila) AND PostoProiezione.numero_posto != NULL AND Database.Posto->exists(Posto | Posto.numero = numero_posto) AND PostoProiezione.id_proiezione != NULL PostoProiezione.stato = False
POST-CONDIZIONE	**context:** PostoProiezioneDAO::create(postoProiezione: PostoProiezioneBean): boolean **post:** Database.Posto_Proiezione->include(postoProiezione)

NOME METODO	+retriveAllByProiezione(proiezioneId: int) : List<PostoProiezione>

DESCRIZIONE	Questo metodo restituisce la lista dei posti per quella prenotazione
PRE-CONDIZIONE	**context:** PostoProiezioneDAO::retriveAllByProiezione(proiezioneId: int) : List<PostoProiezione> **pre:** proiezioneId != NULL AND Database.Posto_Proiezione->exists(PostoPr | PostoPr.id_proiezione = proiezioneId)
POST-CONDIZIONE	**context:** PostoProiezioneDAO::retriveAllByProiezione(proiezioneId: int) : List<PostoProiezione> **post:** result = Database.Posto_Proiezione->select(PostoPr | PostoPr.id_proiezione = proiezioneId)

NOME METODO	+occupaPostoProiezione(posto: PostoProiezione): boolean
DESCRIZIONE	Questo metodo permette di occupare un posto
PRE-CONDIZIONE	**context:** PostoProiezioneDAO::occupaPostoProiezione(posto: PostoProiezione): boolean **pre:** posto != NULL
POST-CONDIZIONE	**context:** PostoProiezioneDAO::occupaPostoProiezione(posto: PostoProiezione): boolean **post:** posto.stato = True

NOME CLASSE	**SlotDAO**
METODI	+retriveById(id: int): SlotBean

	+retriveByProiezione(proiezioneId: int) : SlotBean +retriveFreeSlot(start: LocalDate, end: LocalDate) : List<SlotBeam>
INVARIANTI	

NOME METODO	+retriveById(id: int): SlotBean
DESCRIZIONE	Questo metodo permette di ottenere lo slot tramite l'id
PRE-CONDIZIONE	**context:** SlotDAO::retriveById(id: int): SlotBean **pre:** id != NULL AND Database.Slot->exists(Slot | Slot.id = id)
POST-CONDIZIONE	**context:** SlotDAO::retriveById(id: int): SlotBean **post:** result = Database.Slot->select(Slot | Slot.id = id)

NOME METODO	+retriveByProiezione(proiezioneId: int) : SlotBean
DESCRIZIONE	Questo metodo permette di ottenere lo slot per quella determinata proiezione
PRE-CONDIZIONE	**context:** SlotDAO::retriveByProiezione(proiezioneId: int) : SlotBean **pre:** Database.Proeizione->existst(Proiezione | Proiezione.id = proiezioneId)
POST-CONDIZIONE	**context:** SlotDAO::retriveByProiezione(proiezioneId: int) : SlotBean

	post: Database.Proiezione->select(Proiezione | Proiezione.id_orario)

NOME METODO	+retriveFreeSlot(start: LocalDate, end: LocalDate) : List<SlotBeam>
DESCRIZIONE	Questo metodo permette di ottenere una lista di slot liberi in una determinata fascia oraria
PRE-CONDIZIONE	**context:** SlotDAO::retriveFreeSlot(start: LocalDate, end: LocalDate) : List<SlotBeam> **pre:**
POST-CONDIZIONE	**context:** SlotDAO::retriveFreeSlot(start: LocalDate, end: LocalDate) : List<SlotBeam> **post:** DA VEDERE BENE

4. GLOSSARIO

Prenotazione	Rappresenta l'acquisto di uno o più tickets da parte di un cliente registrato del sistema
Checkout	Inserimento dei dati per effettuare l'acquisto
Catalogo	Elenco di film disponibili in una sede
Cliente (UTC)	Utente registrato, utilizza il sistema per prenotare i biglietti
Gestore Sede(UGS)	Utente registrato, che gestisce una sede specifica della catena Movieplex. Egli gestisce le proiezioni della sua sede e le sale
Gestore Catena(UGC)	Utente registrato, che gestisce la catena Movieplex. Egli gestisce i film proiettabili e le sedi della catena

Piantina sala	Visualizzazione grafica che modella la sala fisica della sede.
Proiezione	Specifico spettacolo di un film, a una data e orario definiti, associato a una sala.
Programmazione	Insieme delle proiezioni di un cinema.