Università degli Studi di Salerno
Corso di Ingegneria del Software

CineNow
ODD: Object Design Document
Versione 2.0

Data: 07/01/2025

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
20/12/2024	1.1	Ridefinizione interfacce dei sottosistemi.	Tutto il team
29/12/2024	1.2	Definizione del class diagram delle entità del sistema, ottimizzato.	Tutto il team
07/01/2025	2.0	Revisione documento per 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).

Le funzionalità che sono state considerate in questo documento si riferiscono soltanto ai requisiti funzionali con priorità alta presenti nel documento RAD_CineNow e saranno implementate nella prima versione del sistema per rispettare i tempi di consegna.

1.1. Object design trade-offs

Funzionalità Avanzate vs. Tempi di Sviluppo:
Come descritto nel RAD e nell'SDD, alcune funzionalità secondarie, non saranno implementate nella prima versione del sistema per rispettare i tempi di consegna. L'attenzione sarà focalizzata sulle funzionalità essenziali. 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 email e password (crittografata), 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.

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: Seguono il formato *PascalCase* e devono terminare con il suffisso **DAO** per indicare il loro ruolo di accesso ai dati (ad esempio, **PrenotazioneDAO**, **UtenteDAO**).

Classi che forniscono servizi: Seguono il formato *PascalCase* (ad esempio, **SalaService**).

Nomi delle Servlet: I nomi delle Servlet, 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 seguire il formato *camelCase*(ad esempio, **calcolaTotale()**, **aggiungiProiezione()**).

Nomi delle Variabili: I nomi delle variabili devono essere descrittivi e seguire il formato *camelCase* (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 interfacce: Le interfacce devono seguire il formato *PascalCase*. Devono rappresentare chiaramente il comportamento che definiscono(ad esempio, **ValidatorStrategy**).

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

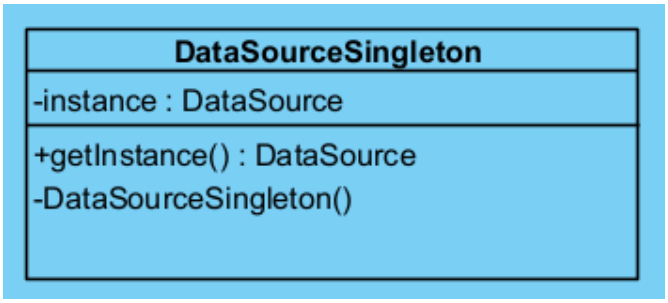
Nomi delle classi che implementano Strategy: Le classi che implementano il Pattern Strategy, devono seguire il formato *PascalCase ,* e devono terminare con la parola **Validator** (ad esempio, **PasswordValidator**)

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

1.3. Design Pattern

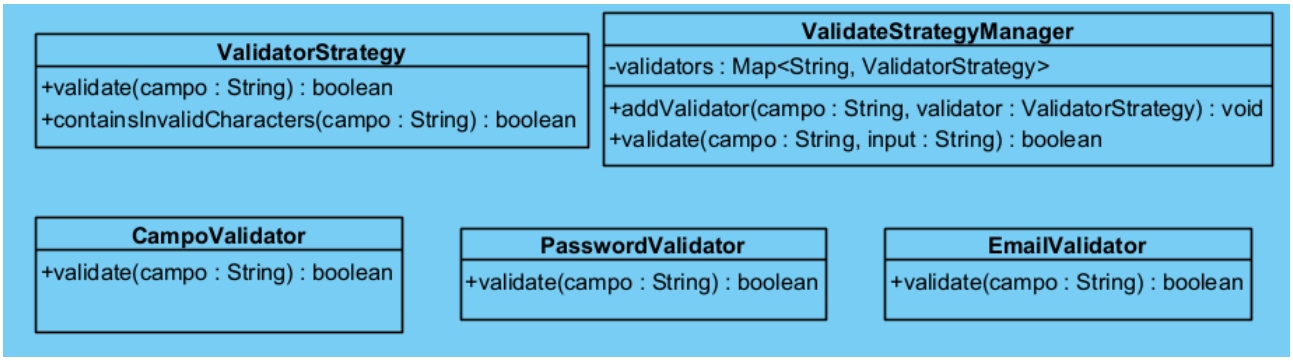
Per implementare le funzionalità del sistema CineNow, sono stati adottati due design pattern: **Singleton Pattern** e **Strategy Pattern**. Di seguito vengono riportate le motivazioni che hanno portato all'adozione dei suddetti pattern nel contesto dell'applicazione.

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 **getInstance().**



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

RAD: Requirements Analysis Document.
SDD: System Design Document.
ODD: Object Design Document.
UTC: Utente Cliente.
UGS: Utente Gestore Sede.
UGC: Utente Gestore Catena.
OCL: Object Constraints Language

1.5. Riferimenti

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

- RAD_CineNow 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.

2. STRUTTURA DEL SISTEMA

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 Struttura del progetto

Il progetto sarà organizzato in modo modulare per garantire una gestione chiara del codice sorgente e delle risorse, la struttura definita a partire dal percorso application/src è :

***main**

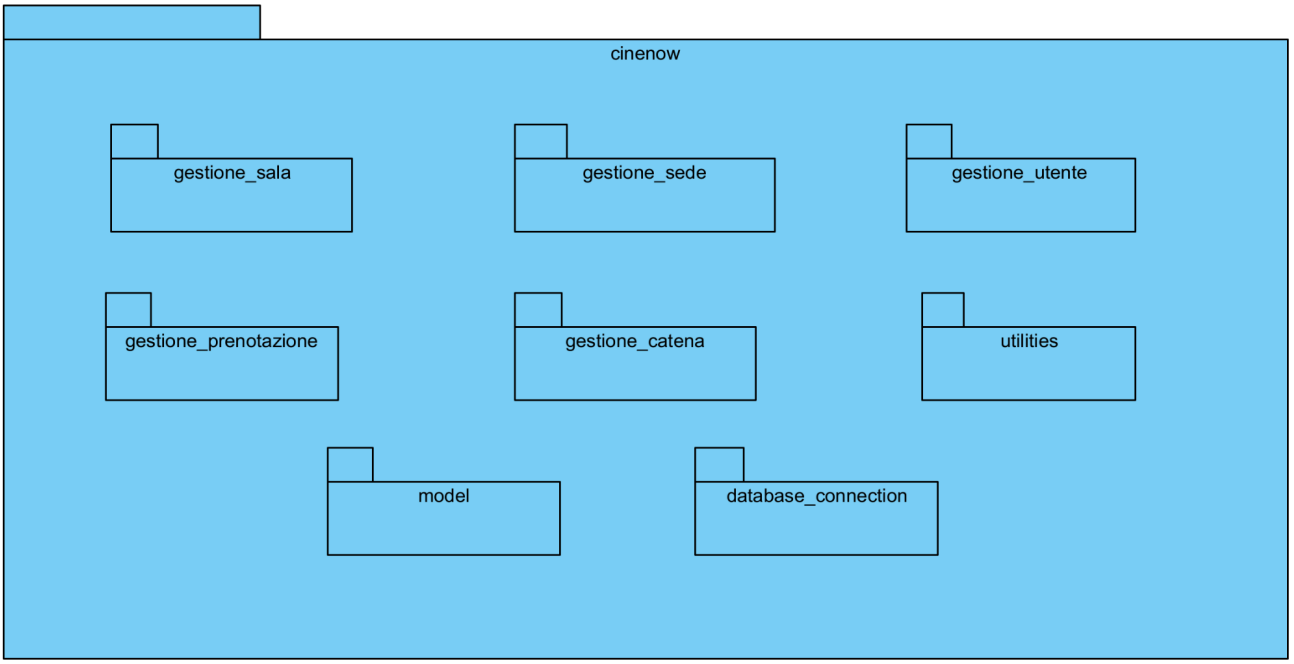
- **java**:Contiene i file sorgente Java organizzati in pacchetti.
  - it.unisa.application:
  - **sottosistemi:**
    - **gestione_sala:**
      - **view**
      - **service**
    - **gestione_catena:**
      - **view**
      - **service**
    - **gestione_prenotazione:**
      - **view**
      - **service**
    - **gestione_utente:**
      - **view**
      - **service**
    - **gestione_sede:**
      - **view**
      - **service**
  - **model**:
    - **entity:** Contiene le entità del sistema
    - **dao:** Contiene le classi responsabili dell'accesso ai dati (DAO)
  - **database_connection**: Contiene le classi per la connessione al database.
  - **utilities:** Contiene le classi per la validazione dei campi e i filtri.
- **webapp**
  - **META-INF**
  - **WEB-INF**
    - **jsp**
  - **static**:
    - **images**
    - **js**
    - **style**

***test**

- **java**: Contiene le classi di test.
  - **integration**: Contiene le classi dei test di integrazione.
  - **unit**: Contiene le classi dei test di unità.

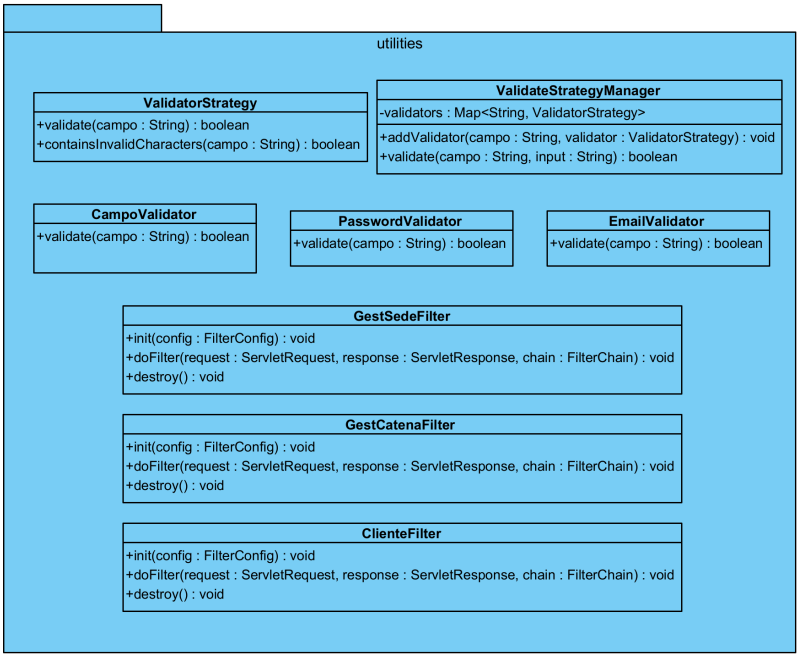
I package identificati con "*view",* nei sottosistemi, contengono le servlet per la logica di presentazione. I "*service",* invece, offrono i servizi dei sottosistemi.

2.3 Packaging del sistema

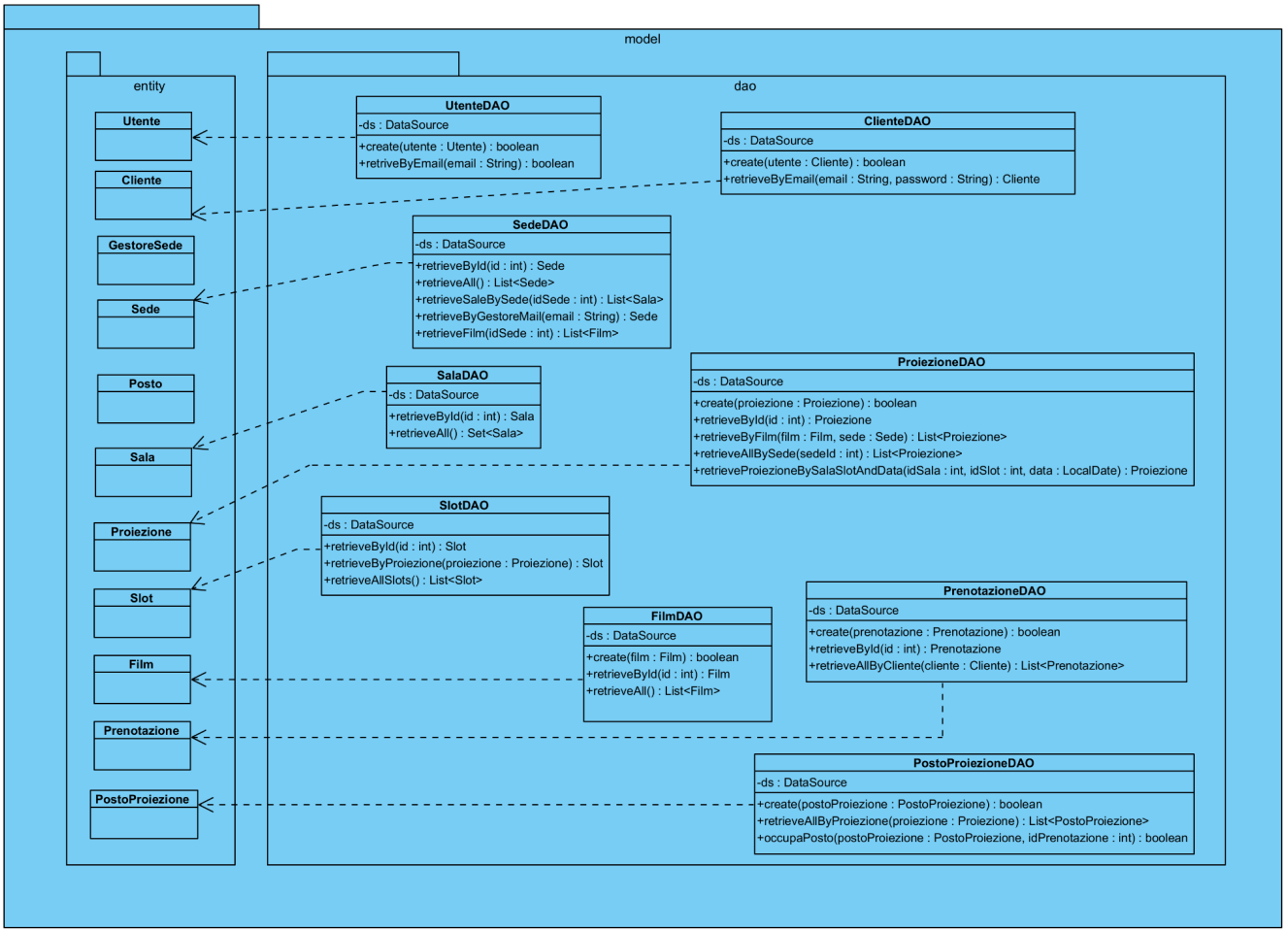


2.4 Dettaglio dei singoli package

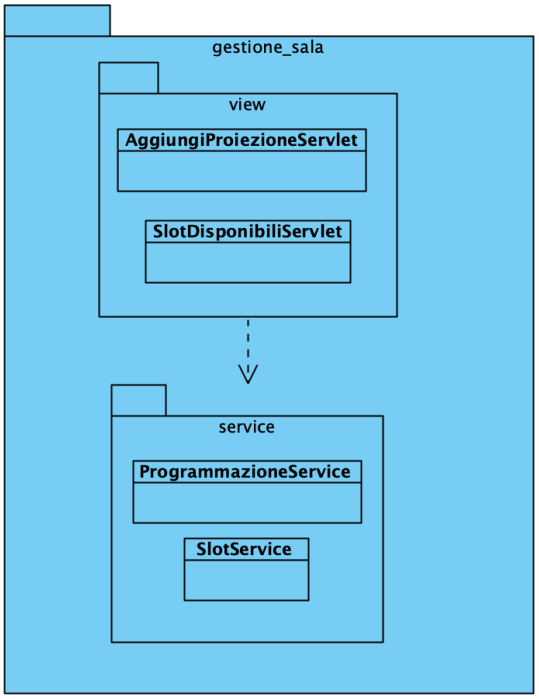
2.4.1 Package utilities



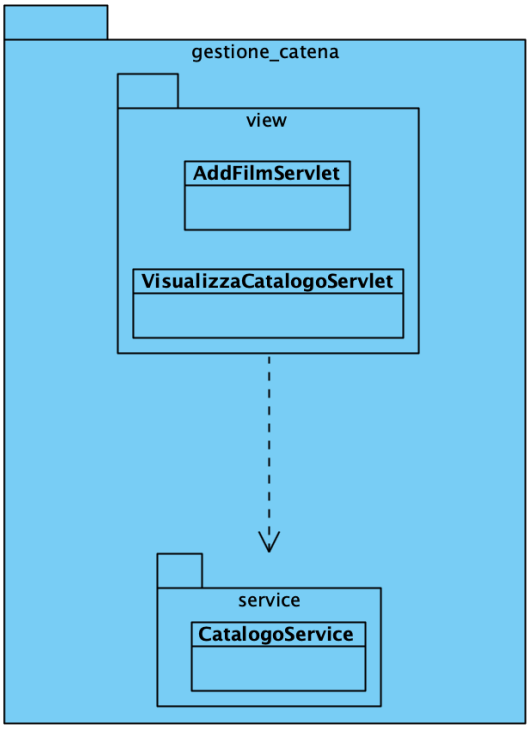
2.4.2 Package model



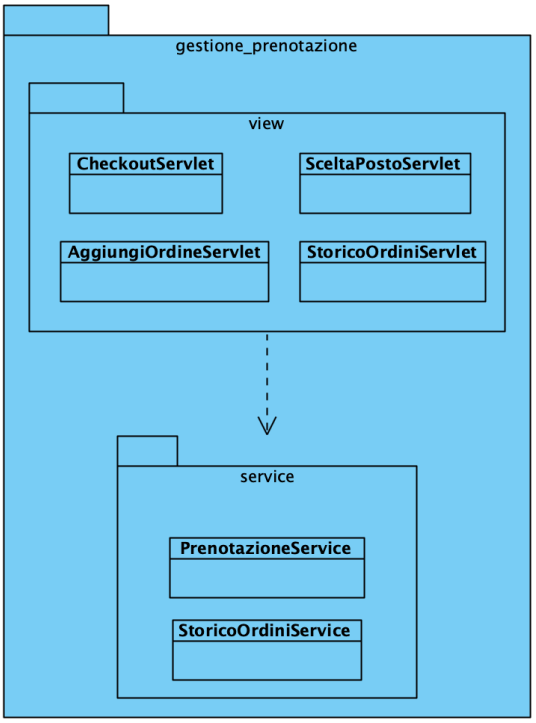
2.4.3 Package gestione_sala



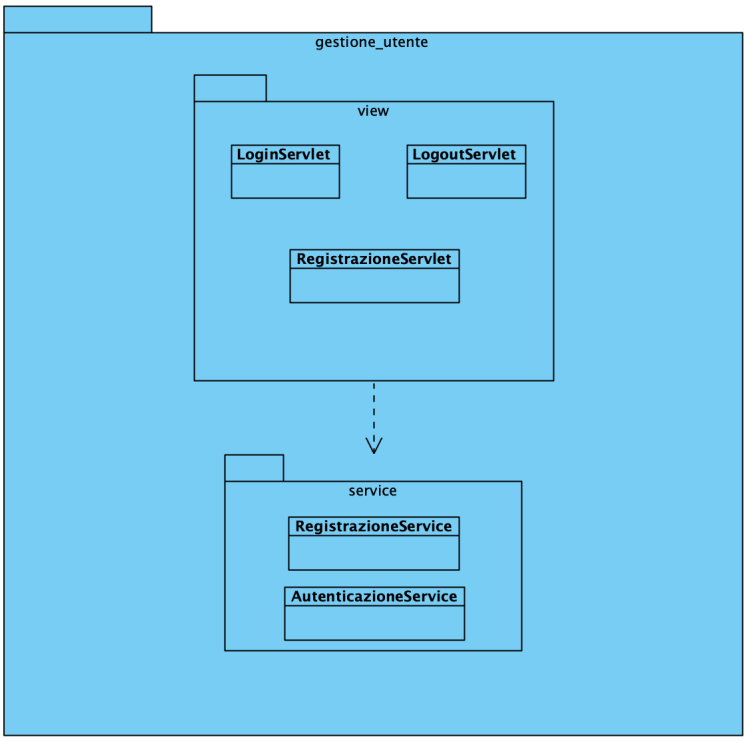
2.4.4 Package gestione_catena



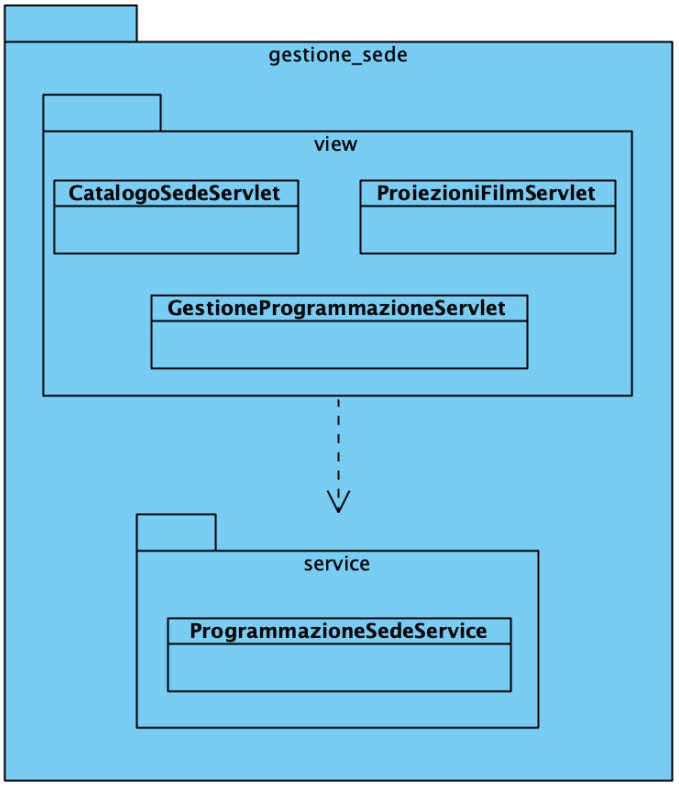
2.4.5 Package gestione_prenotazione



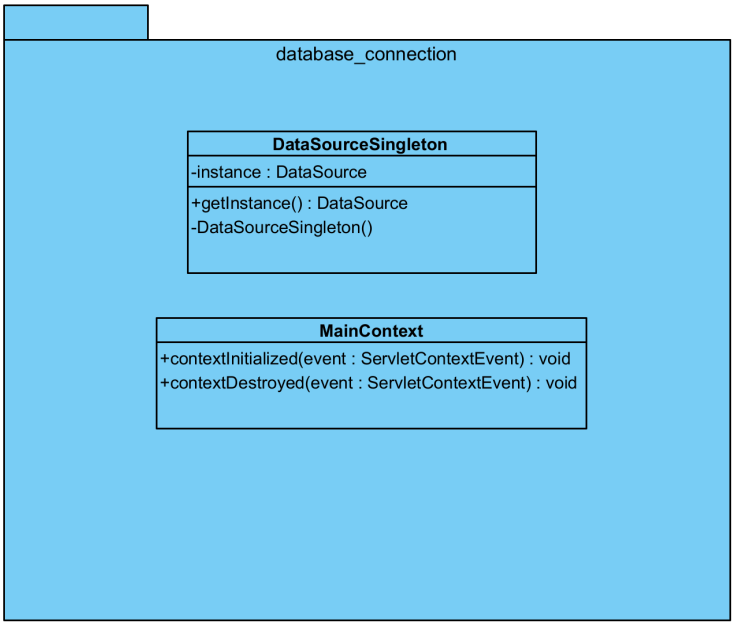
2.4.5 Package gestione_utente



2.4.6 Package gestione_sede



2.4.7 Package 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. Inoltre si specificano delle notazioni contestuali, utilizzate per la specifica delle interfacce:

- "Database": rappresenta un'astrazione del database utilizzato. Es. Database.Utente rappresenta la tabella del database che memorizza i dati persistenti dell'utente.
- dato.isValid() : specifica che il dato è conforme al formato atteso per quel dato..

3.2 gestione_utente

NOME CLASSE	**AutenticazioneService**
METODI	+login(email : String, password : String) : Utente +logout() : void
INVARIANTI	//

NOME METODO	+login(email : String, password : String) : Utente
DESCRIZIONE	Questo metodo permette di loggarsi.
PRE-CONDIZIONE	**context:** AutenticazioneService::login(email:String, password:String) **pre:** email <> null AND email.isValid() AND email<>" " AND password <> null AND password <>" " AND password.isValid() AND password<>null
POST-CONDIZIONE	**context:** AutenticazioneService::login(email:String, password:String) **post:** result <> null AND result.email = email AND result.password = password

NOME METODO	+logout() : void
DESCRIZIONE	Questo metodo permette di effettuare il logout
PRE-CONDIZIONE	**context:** AutenticazioneService::logout() **pre:** session.user <> null
POST-CONDIZIONE	**context:** AutenticazioneServicel::logout() **post:** session.user = null

NOME CLASSE	RegistrazioneService
METODI	+registrazione(email: String, password: String, nome: String, cognome: String) : void
INVARIANTI	//

NOME METODO	+registrazione(email: String, password: String, nome: String, cognome: String) : cliente:Cliente
DESCRIZIONE	Questo metodo permette la registrazione di un nuovo Cliente.
PRE-CONDIZIONE	**context:** RegistrazioneService::registrazione(email: String, password: String, nome: String, cognome: String) **pre:** not(email = null OR email = “ ” OR not(email.isValid()) OR password = null OR password = “ ” OR not(password.isValid()) OR nome = null OR nome = “ ” OR not(nome.isValid()) OR cognome = null OR cognome = “ ” OR not(cognome.isValid()))
POST-CONDIZIONE	**context:** RegistrazioneService::registrazione(email: String, password: String, nome: String, cognome: String) **post:** Database.Cliente->include(cliente| cliente.email=email AND cliente.password=password AND cliente.nome = nome AND cliente.cognome=cognome)

3.3 gestione_catena

NOME CLASSE	CatalogoService
METODI	+addFilmCatalogo(titolo : String, durata : int, descrizione : String, locandina : byte[], genere : String, classificazione : String) : void +getCatalogo() : List<Film>
INVARIANTI	//

NOME METODO	+addFilmCatalogo(titolo : String, durata : int, descrizione : String, locandina : byte[], genere : String, classificazione : String) : void
DESCRIZIONE	Questo metodo permette di aggiungere un film al catalogo
PRE-CONDIZIONE	**context:** CatalogoService::addFilmCatalogo(titolo : String, durata : int, descrizione : String, locandina : byte[], genere : String, classificazione : String) **pre:** titolo <> null AND titolo <> “” AND durata > 0 AND descrizione <> null AND descrizione<> “”" and locandina <> null and locandina <> null AND genere <> null AND genere <> “” AND classificazione <> null AND classificazione <> “” AND self.getCatalogo()->select(f | f.titolo = titolo)->isEmpty()
POST-CONDIZIONE	**context:** CatalogoService::addFilmCatalogo(titolo : String, durata : int, descrizione : String, locandina : byte[], genere : String, classificazione : String) **post:** self.getCatalogo()->size() = self.getCatalogo()@pre->size() + 1

NOME METODO	+getCatalogo() : List<Film>
DESCRIZIONE	Questo metodo permette di ottenere il catalogo.
PRE-CONDIZIONE	//
POST-CONDIZIONE	//

3.4 gestione_prenotazione

NOME CLASSE	**PrenotazioneService**
METODI	+aggiungiOrdine(cliente:Cliente posti:List<PostoProiezione>, proiezione:Proiezione) : Prenotazione +ottieniPostiProiezione(proiezione : Proiezione) : List<PostoProiezione>
INVARIANTI	//

NOME METODO	+aggiungiOrdine(cliente:Cliente, posti:List<PostoProiezione>, proiezione:Proiezione) : Prenotazione
DESCRIZIONE	Questo metodo permette di effettuare una prenotazione.
PRE-CONDIZIONE	**context:** PrenotazioneService::aggiungiOrdine(cliente: Cliente, posti:List<PostoProiezione>, proiezione:Proiezione) **pre:** cliente <> null AND cliente.email <> null AND cliente.email<> “ “ AND posti <> null AND posti.size() > 0 AND proiezione <> null

POST-CONDIZIONE	**context:** PrenotazioneService::aggiungiOrdine(cliente: Cliente, posti: List<PostoProiezione>, proiezione:Proiezione) **post:** StoricoOrdiniService.storicoOrdini(cliente) -> include(result)

NOME METODO	+ottieniPostiProiezione(proiezione : Proiezione) : List<PostoProiezione>
DESCRIZIONE	Questo metodo permette di effettuare una prenotazione.
PRE-CONDIZIONE	**context:** PrenotazioneService::ottieniPostiProiezione(proiezione : Proiezione) **pre:** proiezione <> null
POST-CONDIZIONE	**context:** PrenotazioneService::ottieniPostiProiezione(proiezione : Proiezione) **post:** result <> null

NOME CLASSE	**StoricoOrdiniService**
METODI	+storicoOrdini(cliente : Cliente) : List<Prenotazione>
INVARIANTI	//

NOME METODO	+storicoOrdini(cliente : Cliente) : List<Prenotazione>
DESCRIZIONE	Il metodo che restituisce la lista delle prenotazioni di un cliente.
PRE-CONDIZIONE	**context:** StoricoOrdiniService::storicoOrdini(cliente: Cliente) **pre:** cliente <> null
POST-CONDIZIONE	**context:** StoricoOrdiniService::storicoOrdini(cliente: Cliente) **post**: result <> null

3.5 gestione_sala

NOME CLASSE	SlotService
METODI	+slotDisponibili(sede:Sede, sala:Sala dataInizio :LocalDate, dataFine : LocalDate) : List<Slot>
INVARIANTI	//

NOME METODO	+slotDisponibili(sede:Sede, sala:Sala dataInizio :LocalDate, dataFine : LocalDate) : List<Slot>
DESCRIZIONE	Questo metodo permette di ottenere gli slot disponibili per ogni sala di una sede.
PRE-CONDIZIONE	**context:** SlotService::slotDisponibili(sede:Sede, sala:Sala dataInizio :LocalDate, dataFine : LocalDate) : List<Slot> **pre:** sede <> null AND sala <> null AND dataInizio <= dataFine AND (dataInizio <> null AND dataFine <> null)
POST-CONDIZIONE	**context:** SlotService::slotDisponibili(sede:Sede, sala:Sala dataInizio :LocalDate, dataFine : LocalDate) : List<Slot> **post:** result-> forAll(slot | sala.slots->includes(slot) AND sede.sale->includes(sala) AND slot.data >= dataInizio AND slot.data <= dataFine)

3.6 gestione_sede

NOME CLASSE	ProgrammazioneSedeService
METODI	+getProgrammazione(sedeId : int) : List<Proiezione> +getProiezioniFilm(sedeId : int, filmId : int) : List<Proiezione> +getCatalogoSede(sede : Sede) : List<Film>
INVARIANTI	//

NOME METODO	+getProgrammazione(sedeId : int) : List<Proiezione>
DESCRIZIONE	Questo metodo permette di ottenere la programmazione per una sede.
PRE-CONDIZIONE	**context:** ProgrammazioneSedeService::getProgrammazione(sedeId : int) **pre:** sedeId > 0
POST-CONDIZIONE	**context:** ProgrammazioneSedeService::getProgrammazione(sedeId : int) **post:** result = Database.Proiezione->select(proiezione | proiezione.sala.id_sede = sedeId)

NOME METODO	+getProiezioniFilm(sedeId : int, filmId : int) : List<Proiezione>
DESCRIZIONE	Questo metodo permette di ottenere la programmazione per un film di una determinata sede.
PRE-CONDIZIONE	**context:** ProgrammazioneSedeService::getProiezioniFilm(sedeId : int, filmId : int) **pre:** filmId > 0 AND sedeId > 0

POST-CONDIZIONE	**context:** ProgrammazioneSedeService:: getProiezioniFilm(sedeId : int, filmId : int) **post:** result = self.getProgrammazione(sedeId)-> select(proiezione | proiezione.id_film = filmId)

NOME METODO	+getCatalogoSede(sede : Sede) : List<Film>
DESCRIZIONE	Questo metodo permette di ottenere il catalogo di una specifica sede.
PRE-CONDIZIONE	**context:** ProgrammazioneSedeService::getCatalogoSede (sede : Sede) **pre:** sede <> null
POST-CONDIZIONE	**context:** ProgrammazioneSedeService:: getCatalogoSede(sede : Sede) **post:** result<>isEmpty() AND result <> null

4. CLASS DIAGRAM OTTIMIZZATO

Per il class diagram delle entità, completo e ottimizzato, si rimanda a ClassDiagram_CineNow.

5. GLOSSARIO

Prenotazione	Rappresenta l'acquisto di uno o più tickets da parte di un cliente registrato del sistema
Catalogo	Elenco di film disponibili.
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
Proiezione	Specifico spettacolo di un film, a una data e orario definiti, associato a una sala.
Programmazione/Proiezioni	Insieme delle proiezioni di un cinema.
Database	Viene utilizzato nella definizione delle interfacce, come astrazione del database effettivamente utilizzato.