

2021/2022

# ODD

## Object Design Document

UNICINEMA

GERARDO LEONE, ANTONIO SANTOSUOSSO, MARIO LEZZI



## Revision History

Data	Versione	Descrizione	Autori
20/12/2021	0.1	Prima Stesura	LM
28/12/2021	0.2	Creazione dei Packages	LM, LG, SA
30/12/2021	0.3	Modifiche ai Packages e aggiunta delle class Interfaces	LG, SA
20/01/2021	0.4	Modifica dei Class Interface	LG, SA
14/02/2022	1.0	Revisione finale	SA,LM

## Team Members

Nome	Ruolo del progetto	Acronimo	Informazioni di contatto
Lezzi Mario	Team Member	LM	m.lezzi@studenti.unisa.it
Leone Gerardo	Team Member	LG	g.leone35@studenti.unisa.it
Santosuosso Antonio	Team Member	SA	a.santosuosso3@studenti.unisa.it



## 1. Introduzione

Dopo la realizzazione dei documenti RAD e SDD abbiamo descritto, in linea di massima, quello che sarà il nostro sistema e quindi i nostri obiettivi, tralasciando gli aspetti dell'implementazione. Il seguente documento ha lo scopo di produrre un modello capace di integrare in modo coerente e preciso tutte le diverse funzionalità individuate nelle fasi precedenti. In particolare, questo documento si vanno a descrivere i trade-offs generali realizzati dagli sviluppatori, le linee guida sulla documentazione delle interfacce e le convenzioni di codifica, le Interfacce delle classi, le operazioni, i tipi, gli argomenti e la signature dei sottosistemi definiti nel System Design.

### 1.1 Object design goals

**Riusabilità:** Il sistema deve basarsi sulla riusabilità, attraverso l'utilizzo di ereditarietà e design patterns.

**Robustezza:** Il sistema deve risultare robusto, reagendo correttamente a situazioni impreviste attraverso il controllo degli errori e la gestione delle eccezioni.

**Incapsulamento:** Il sistema garantisce la segretezza sui dettagli implementativi delle classi grazie all'utilizzo delle interfacce, rendendo possibile l'utilizzo di funzionalità offerte da diversi componenti.

### 1.2 Object Design Trade-off

<b>Prestazioni vs costi</b>	Il team cercherà di ottenere le migliori prestazioni utilizzando il budget a disposizione.
<b>Tempi di risposta vs sicurezza</b>	Aumentare la sicurezza attraverso sistemi, comporta un aumento dei tempi di risposta delle operazioni.

### 1.2 Definizione, acronimi e abbreviazioni

- **RAD:** Requirements Analysis Document))
- **SDD:** System Design Document



- **ODD:** Object Design Document
- **Package:** raggruppamento di classi, interfacce o file correlati;
- **Design pattern:** template di soluzioni a problemi ricorrenti impiegati per ottenere riuso e flessibilità;
- **Interfaccia:** insieme di signature delle operazioni offerte dalla classe;
- **View:** nel pattern MVC rappresenta ciò che viene visualizzato a schermo da un utente e che gli permette di interagire con le funzionalità offerte dalla piattaforma;

### 1.3 Linee guida per la documentazione

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

#### **Link a documentazione ufficiale sulle convenzioni**

- Lo standard nella definizione delle classi e delle interfacce java è quello definito da Google(<https://google.github.io/styleguide/javaguide.html>).
- HTML: [HTML Style Guide and Coding Conventions \(w3schools.com\)](https://www.w3schools.com/html/html5_style_guide.asp)

### 1.4 Riferimenti

- Slides del corso;
- B.Bruegge, A. H. Dutoit, Object Oriented Software Engineering - Using UML, Pattern and Java, Prentice Hall, 3rd edition, 2009;
- RAD del progetto UniCinema;

## **Javadoc di UniCinema**

Di seguito il link al sito contenente il javadoc di UniCinema:

<https://bloodmask.github.io/UniCinema.github.io/site/JavaDoc/index.html>

## 2. Packages

In questa sezione viene mostrata la suddivisione del sistema in package, in base a quanto definito nel documento di System Design. Tale suddivisione è



motivata dalle scelte architetturelle prese e ricalca la struttura di directory standard definita da Maven.

- **.idea**
- **src** contiene tutti i file sorgente
  - **main**
    - **Java**, contiene le classi Java relative alle componenti Control e Model
    - **webapp** contiene i file relativi alle componenti View
      - ❖ **static**, contiene i fogli di stile CSS e gli script JS
      - ❖ **views**, contiene i file JSP
      - ❖ **WEB-INF** contiene librerie
  - **test**, contiene tutto il necessario per il testing
    - **java**, contiene le classi Java per l'implementazione del testing.
- **target** contiene tutti i file prodotti dal sistema di build di Maven.

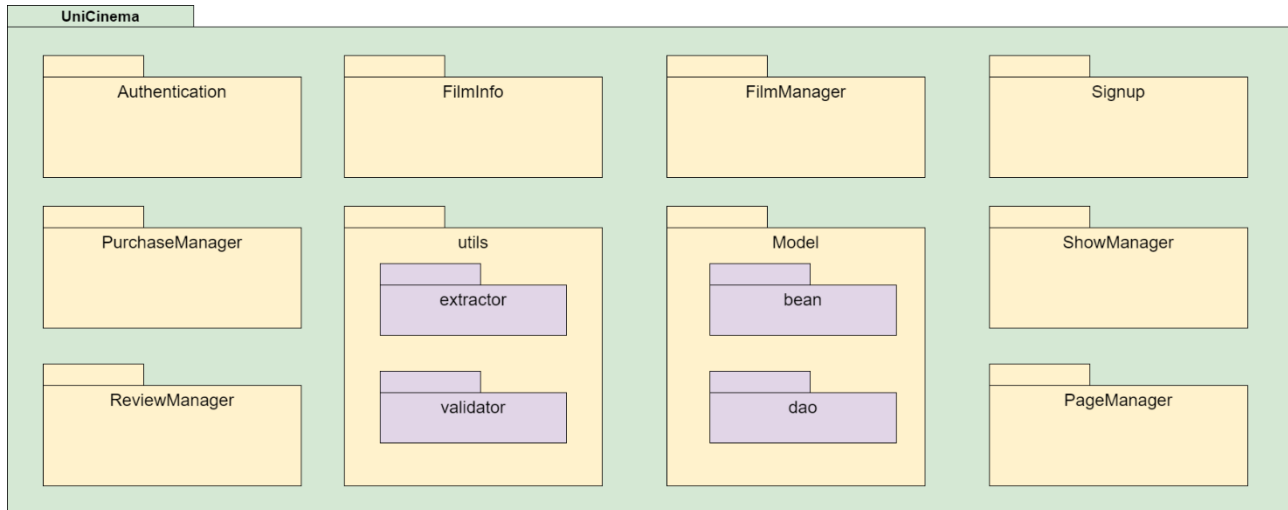
## Package UniCinema

Nella presente sezione si mostra la struttura del package principale di UniCinema. La struttura generale è stata ottenuta da quattro scelte principali, cioè:

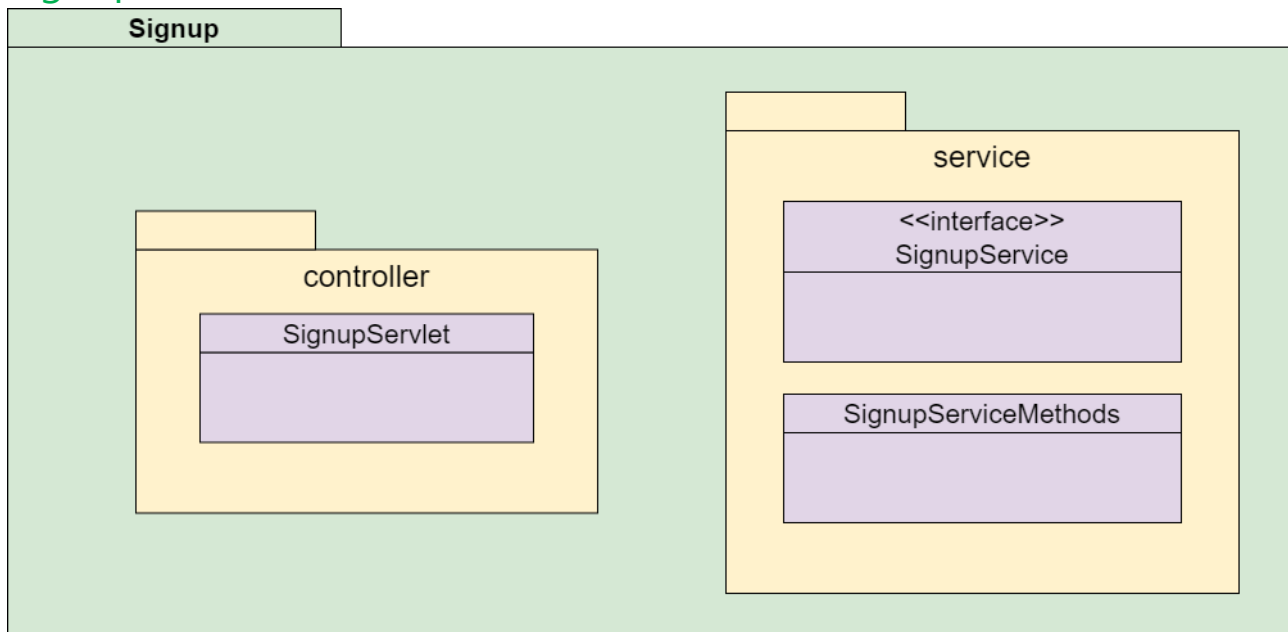
1. Creare un package per ogni sottosistema, ognuno contenente le classi service e il controller del sottosistema.
2. Creare un package a parte chiamato *model*, contenente i *bean* e i *DAO* per l'accesso al Database. È stata adottata questa disposizione per migliorare la leggibilità del codice, velocizzare il processo di ricerca dei vari moduli java, dal momento che il Database risulta essere molto complesso e pieno di operazioni.
3. Creare un package chiamato *utils* all'interno del quale sono state inserite tutte le classi di utilità generiche, dunque utilizzabili da qualsiasi sottosistema.
4. Creare un package per la servlet che gestisce la pagina principale, ovvero homepage.



Laurea Magistrale in Informatica – Università di Salerno  
Corso di Ingegneria del Software - Prof. C. Gravino

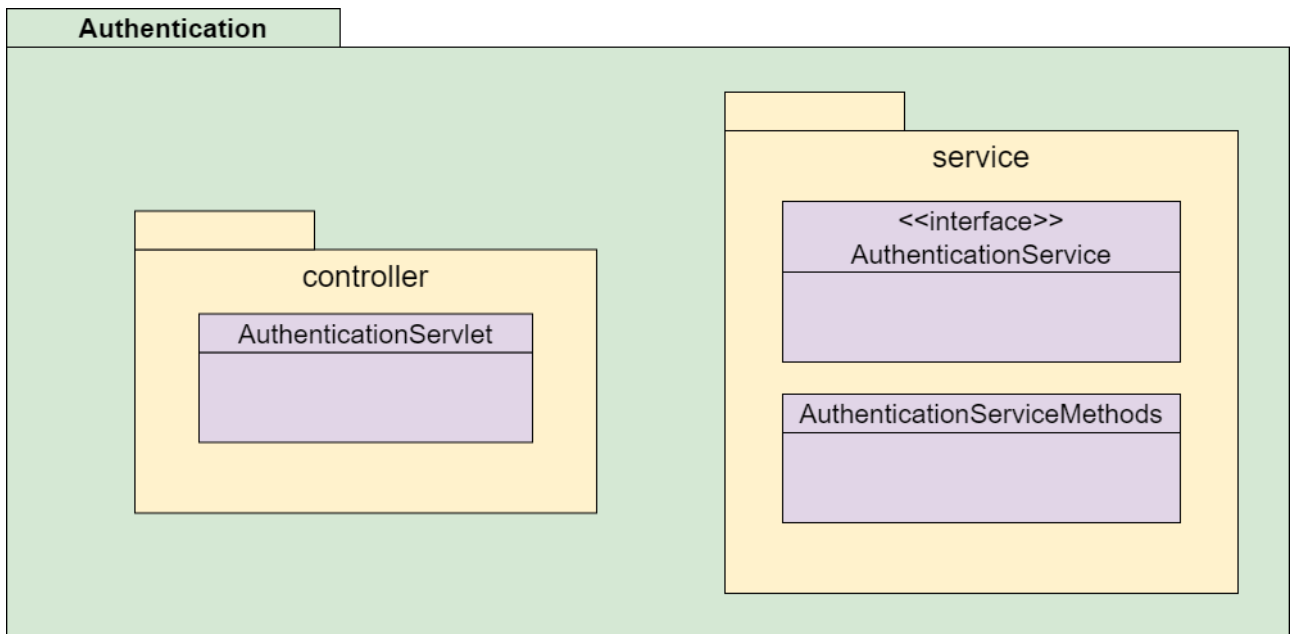


## Signup

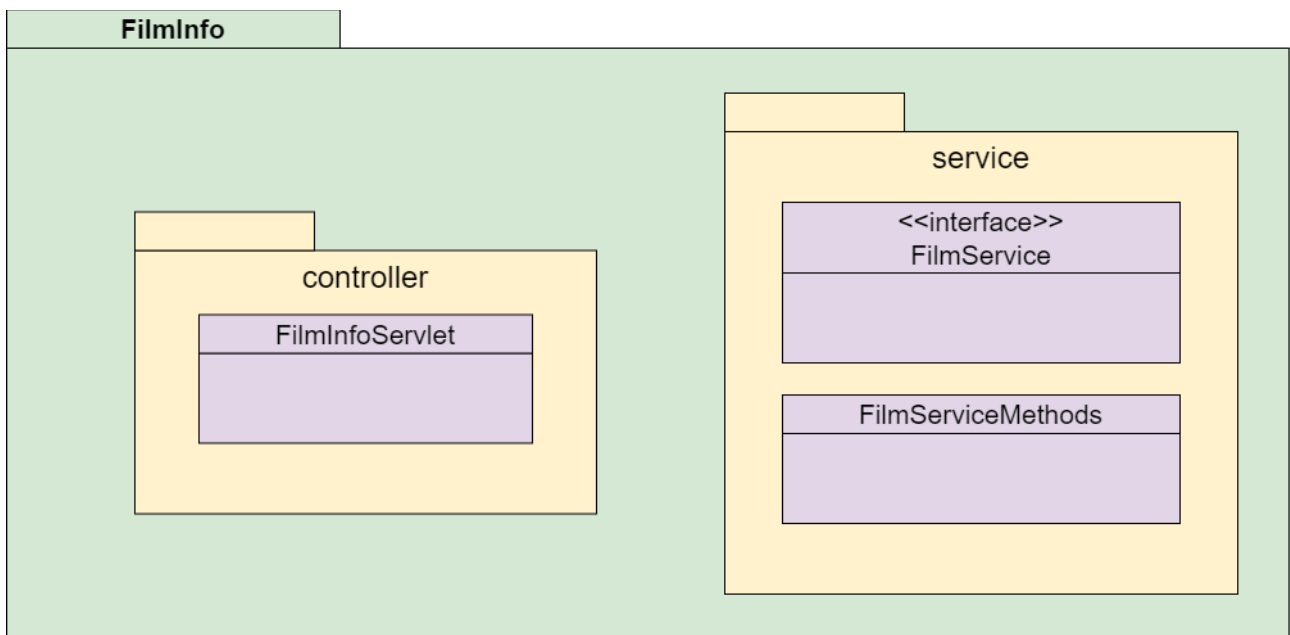




## Authentication

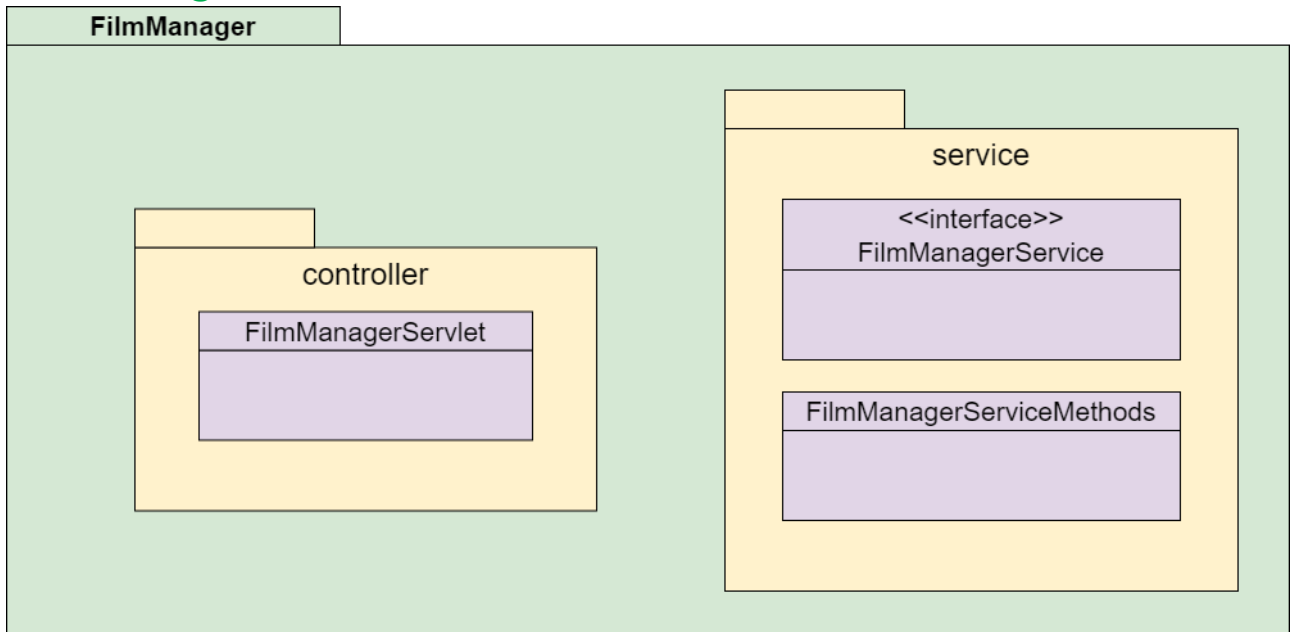


## FilmInfo

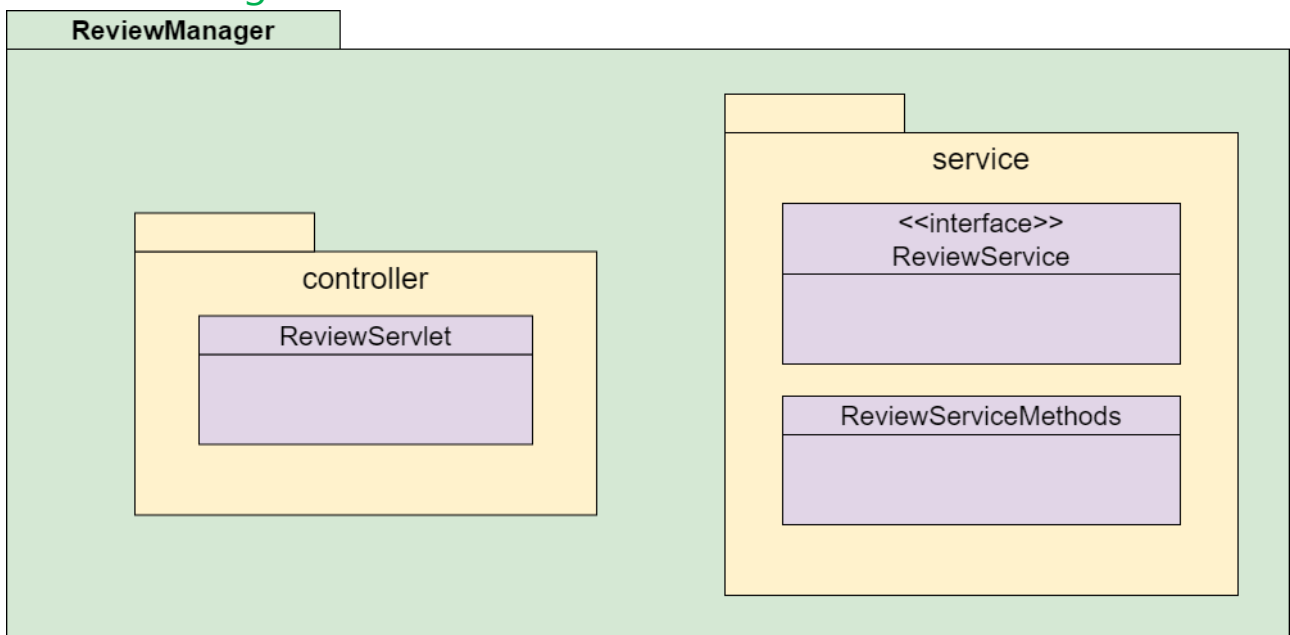




## FilmManager

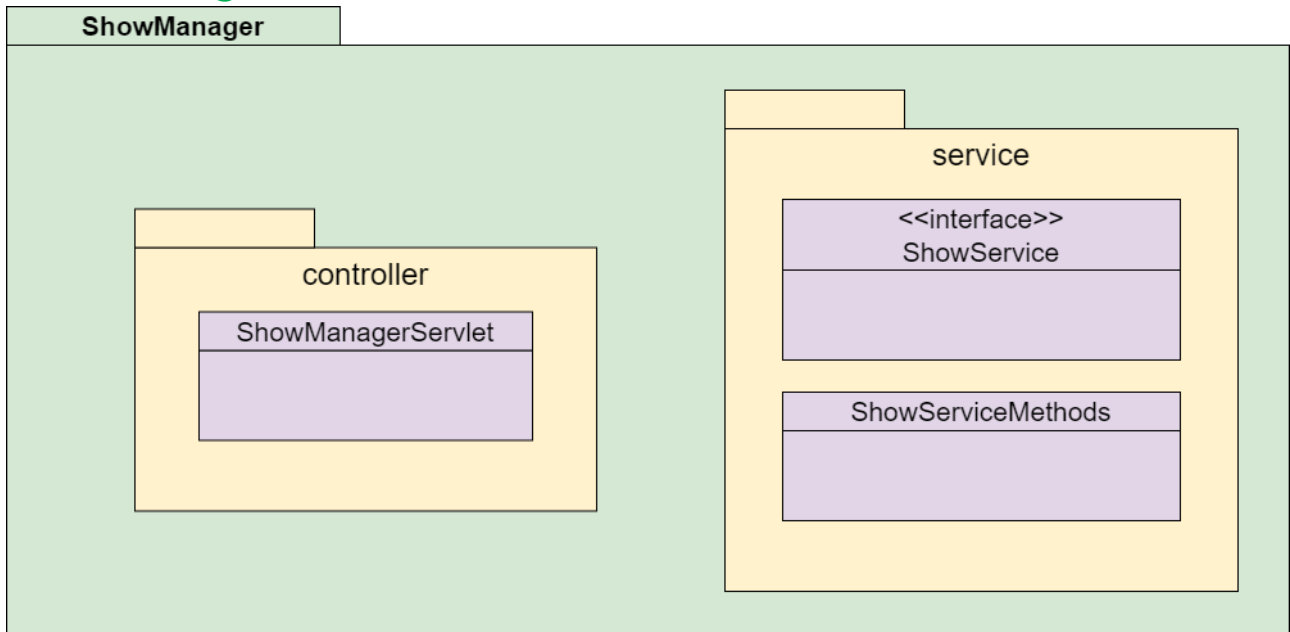


## ReviewManager

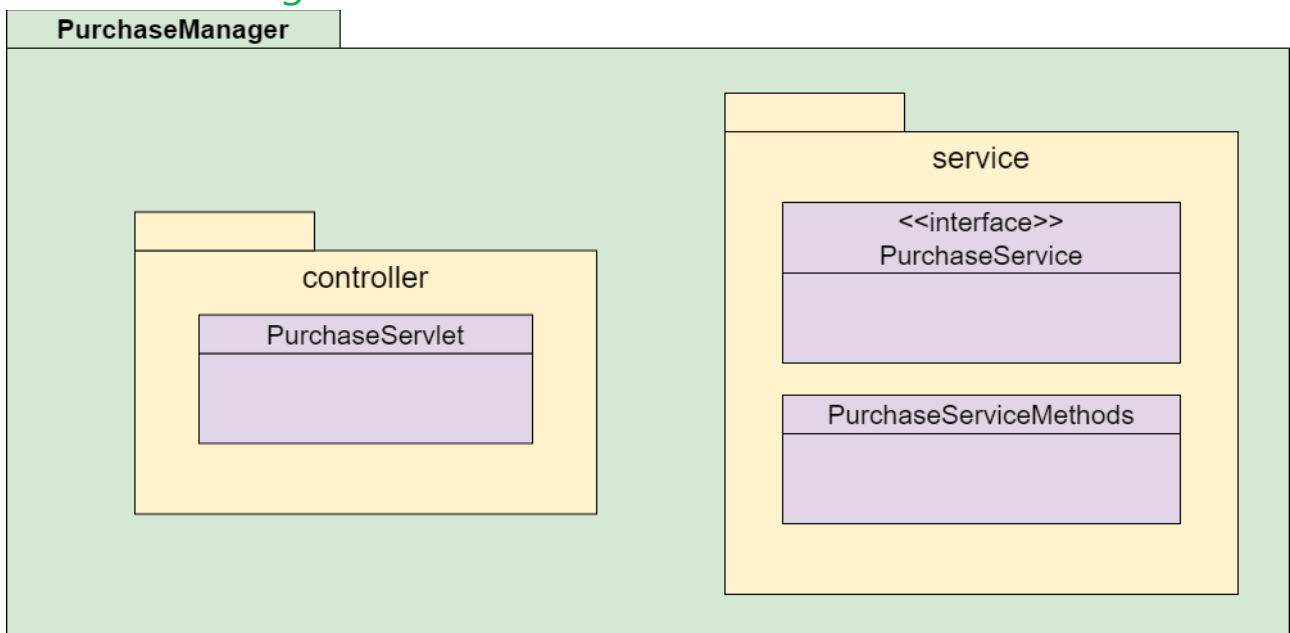




## ShowManager



## PurchaseManager



## 3. Class Interface

In questa sezione verranno presentate le interfacce di ogni package.



### 3.1 Package Signup

<b>Nome classe</b>	<b>SignupService</b>
<b>Descrizione</b>	Questa classe permette di gestire le operazioni relative alla registrazione di un utente.
<b>Metodi</b>	+signup(Account account): Boolean
<b>Invariante di classe</b>	

<b>Nome Metodo</b>	Signup(Account account)
<b>Descrizione</b>	Questo metodo permette di registrare un account.
<b>Pre-condizione</b>	/
<b>Post-condizione</b>	<b>context:</b> SignupService(Account account) <b>post:</b> accountDAO.fetch(account.email).equals(account)

### 3.2 Package Authentication

<b>Nome classe</b>	<b>AuthenticationService</b>
<b>Descrizione</b>	Questa classe permette di gestire le operazioni relative all'autenticazione, modifica e visualizzazione di un Account.
<b>Metodi</b>	+signin(Account account): Account +fetch(String email): Account +edit(Account account): boolean
<b>Invariante di classe</b>	/

<b>Nome Metodo</b>	+signin(Account account)
<b>Descrizione</b>	Questo metodo consente di autenticare un nuovo account.
<b>Pre-condizione</b>	/
<b>Post-condizione</b>	<b>Context:</b> AccountService:: signin(Account account) <b>Post:</b> accountDAO.find(account.email, account.pswrd, false) != null
<b>Nome metodo</b>	+fetch(String email)
<b>Descrizione</b>	Questo metodo consente di restituire un account.



<b>Pre-condizione</b>	/
<b>Post-condizione</b>	/
<b>Nome metodo</b>	+edit(Account account)
<b>Descrizione</b>	Questo metodo consente di modificare un account
<b>Pre-condizione</b>	<b>context:</b> AccountService:: edit(Account account) <b>pre:</b> accountDAO.fetch(account.email).equals(account)
<b>Post-condizione</b>	<b>context:</b> AccountService:: edit(Account account) <b>post:</b> accountDAO.fetch(account.email).equals(@pre.accountDAO.fetch(account.email)) not

### 3.3 Package FilmInfo

<b>Nome classe</b>	<b>FilmService</b>
<b>Descrizione</b>	Questa classe permette la visualizzazione e la ricerca dei film.
<b>Metodi</b>	+fetch(int filmId): Film +search(String title): ArrayList<Film> +fetchLastReleases(int total): ArrayList<Film> +fetchComingSoon(int total): ArrayList<Film>
<b>Invariante di classe</b>	/

<b>Nome Metodo</b>	+fetch(int filmId)
<b>Descrizione</b>	Questo metodo restituisce un film a partire dal suo id.
<b>Pre-condizione</b>	/
<b>Post-condizione</b>	/
<b>Nome metodo</b>	+search(String title)
<b>Descrizione</b>	Questo metodo restituisce la lista dei film i cui titoli contengono in parte la stringa passata.
<b>Pre-condizione</b>	/
<b>Post-condizione</b>	/
<b>Nome metodo</b>	+fetchLastReleases(int total)
<b>Descrizione</b>	Questo metodo restituisce la lista degli ultimi film usciti. Il numero di questi è stabilito dal parametro <i>total</i> .
<b>Pre-condizione</b>	/
<b>Post-condizione</b>	/



<b>Nome metodo</b>	+fetchComingSoon(int total)
<b>Descrizione</b>	Questo metodo restituisce la lista dei film in uscita. Il numero di questi è stabilito dal parametro <i>total</i> .
<b>Pre-condizione</b>	/
<b>Post-condizione</b>	/

### 3.4 Package FilmManager

<b>Nome classe</b>	<b>FilmManagerService</b>
<b>Descrizione</b>	Questa classe consente la gestione dei Film.
<b>Metodi</b>	+removeFilm(int filmId): boolean +insert(Film film): boolean +update(Film film): boolean
<b>Invariante di classe</b>	/

<b>Nome Metodo</b>	+removeFilm(int filmId)
<b>Descrizione</b>	Questo metodo rimuove un film a partire dal suo id.
<b>Pre-condizione</b>	<b>context:</b> FilmManagerService:: removeFilm(int id) <b>pre:</b> filmDAO.fetch(film.id) != null
<b>Post-condizione</b>	<b>context:</b> FilmManagerService:: removeFilm(int id) <b>post:</b> filmDAO.fetch(filmId) == null
<b>Nome metodo</b>	+insert(Film film)
<b>Descrizione</b>	Questo metodo aggiunge un film.
<b>Pre-condizione</b>	<b>context:</b> FilmManagerService:: insert(Film film) <b>pre:</b> filmDAO.fetch(film.id) == null
<b>Post-condizione</b>	<b>context:</b> FilmManagerService:: insert (Film film) <b>post:</b> filmDAO.fetch(film.id).equals(film)
<b>Nome metodo</b>	+update(Film film)
<b>Descrizione</b>	Questo metodo aggiorna le informazioni di un film.
<b>Pre-condizione</b>	<b>context:</b> FilmManagerService:: update(Film film) <b>pre:</b> filmDAO.fetch(film.id).equals(film)
<b>Post-condizione</b>	<b>context:</b> FilmManagerService:: update(Film film) <b>post</b> not filmDAO.fetch(film.id).equals(@pre.filmDAO.fetch(film.id))

### 3.5 Package ReviewManager

<b>Nome classe</b>	<b>ReviewService</b>
--------------------	----------------------



<b>Descrizione</b>	Questa classe permette di gestire le operazioni relative alle recensioni.
<b>Metodi</b>	+countAll (Film film): int +fetchAll (Film film, Paginator paginator): ArrayList<Review> +fetchAll (int filmId): ArrayList<Review> +averageStars (ArrayList<Review> reviewList, int stars): int +percentageStars (ArrayList<Review> reviewList, int stars): double +insert (Review review): boolean +fetch (int accountId, int filmId): <Review> +delete(int accountId): boolean +countByAccountId(int id): int
<b>Invariante di classe</b>	

<b>Nome Metodo</b>	+countAll (Film film)
<b>Descrizione</b>	Questo metodo restituisce il numero di recensioni a partire dal film a cui si riferiscono.
<b>Pre-condizione</b>	/
<b>Post-condizione</b>	/
<b>Nome metodo</b>	+fetchAll(Film film, Paginator paginator)
<b>Descrizione</b>	Questo metodo restituisce la lista delle recensioni a partire dal film a cui si riferiscono.
<b>Pre-condizione</b>	/
<b>Post-condizione</b>	/
<b>Nome Metodo</b>	+fetchAll (int filmId)
<b>Descrizione</b>	Questo metodo restituisce tutte le recensioni a partire dall'identificato del film a cui si riferiscono.
<b>Pre-condizione</b>	/
<b>Post-condizione</b>	/
<b>Nome Metodo</b>	+averageStars (ArrayList<Review> reviewList, int stars)
<b>Descrizione</b>	Questo metodo restituisce la media delle valutazioni in base al numero di stelle passato.
<b>Pre-condizione</b>	/
<b>Post-condizione</b>	/
<b>Nome metodo</b>	+percentageStars (ArrayList<Review> reviewList, int stars)
<b>Descrizione</b>	Questo metodo restituisce la percentuale delle valutazioni in base al numero di stelle passato.
<b>Pre-condizione</b>	/
<b>Post-condizione</b>	/
<b>Nome metodo</b>	+insert (Review review): boolean
<b>Descrizione</b>	Questo metodo registra una recensione.
<b>Pre-condizione</b>	<b>context:</b> ReviewService:: insert(Review review) <b>pre:</b> not



	reviewDAO.fetchAll(filmId).includes(review)
<b>Post-condizione</b>	<b>context:</b> ReviewService:: insert(Review review) <b>post:</b> reviewDAO.fetchAll(filmId).includes(review) and reviewDAO.fetchAll(filmId).size() == @pre. reviewDAO.fetchAll(filmId).size()+1
<b>Nome metodo</b>	+fetch (int accountId, int filmId)
<b>Descrizione</b>	Questo metodo restituisce la recensione a partire dall'account che l'ha scritta e dal film a cui si riferisce.
<b>Pre-condizione</b>	/
<b>Post-condizione</b>	/
<b>Nome metodo</b>	+delete(int accountId)
<b>Descrizione</b>	Questo metodo permette di cancellare una recensione pubblicata da un account.
<b>Pre-condizione</b>	<b>context:</b> ReviewService:: delete(int accountId) <b>pre:</b> reviewDAO.fetchAll(filmId).includes(review)
<b>Post-condizione</b>	<b>context:</b> ReviewService:: delete(int accountId) <b>post:</b> not reviewDAO.fetchAll(filmId).includes(review) and reviewDAO.fetchAll(filmId).size() == @pre. reviewDAO.fetchAll(filmId).size()-1
<b>Nome metodo</b>	+countByAccountId(int id)
<b>Descrizione</b>	Questo metodo restituisce il numero di recensioni scritte da un account.
<b>Pre-condizione</b>	/
<b>Post-condizione</b>	/

### 3.6 Package ShowManager

<b>Nome classe</b>	<b>ShowService</b>
<b>Descrizione</b>	Questa classe permette di gestire le operazioni relative agli spettacoli e alle sale.
<b>Metodi</b>	+fetchAll (Film film): ArrayList<Show> +fetchAll(): ArrayList<Show> +fetchDaily (int roomId, LocalDate date): ArrayList<Show> +fetchDaily (int roomdId, LocalDate date, Show show): ArrayList<Show> +fetch (int id): Show +fetchRoom (int showId): Room +remove (int showId): boolean +insert (Show show): boolean +update (Show show): boolean



<b>Invariante di classe</b>	
-----------------------------	--

<b>Nome Metodo</b>	+fetchAll (Film film)
<b>Descrizione</b>	Questo metodo restituisce tutti gli spettacoli che proiettano un dato film.
<b>Pre-condizione</b>	/
<b>Post-condizione</b>	/
<b>Nome metodo</b>	+fetchAll()
<b>Descrizione</b>	Questo metodo restituisce tutti gli spettacoli registrati nel database.
<b>Pre-condizione</b>	/
<b>Post-condizione</b>	/
<b>Nome Metodo</b>	+fetchDaily (int roomId, LocalDate date)
<b>Descrizione</b>	Questo metodo restituisce tutti gli spettacoli in una data specifica.
<b>Pre-condizione</b>	/
<b>Post-condizione</b>	/
<b>Nome Metodo</b>	+fetchDaily (int roomdId, LocalDate date, Show show)
<b>Descrizione</b>	Questo metodo restituisce tutti gli spettacoli in una data escludendo lo spettacolo specificato. Tale esclusione avviene perché quando bisogna modificare l'orario di uno spettacolo, non si deve tener conto degli orari occupati in precedenza dallo stesso.
<b>Pre-condizione</b>	/
<b>Post-condizione</b>	/
<b>Nome metodo</b>	+fetch (int id)
<b>Descrizione</b>	Questo metodo restituisce uno spettacolo a partire dal suo identificativo.
<b>Pre-condizione</b>	/
<b>Post-condizione</b>	/
<b>Nome metodo</b>	+fetchRoom (int showId)
<b>Descrizione</b>	Questo metodo restituisce una sala a partire dall'identificativo dello spettacolo.
<b>Pre-condizione</b>	/
<b>Post-condizione</b>	/
<b>Nome metodo</b>	+remove (int showId)
<b>Descrizione</b>	Questo metodo rimuove uno spettacolo a partire dal suo identificativo.
<b>Pre-condizione</b>	<b>context:</b> ShowService:: remove(int id) <b>pre:</b> showDAO.fetch(showId) != null
<b>Post-condizione</b>	<b>context:</b> ShowService:: remove(int id) <b>post:</b> showDAO.fetch(showId) == null and



	showDAO.fetchAll().size == @pre.showDAO.fetchAll().size()-1
<b>Nome metodo</b>	+insert (Show show)
<b>Descrizione</b>	Questo metodo registra uno spettacolo.
<b>Pre-condizione</b>	<b>context:</b> ShowService:: insert(Show show) <b>pre:</b> not showDAO.fetchAll().includes(show)
<b>Post-condizione</b>	<b>context:</b> ShowService:: insert(Show show) <b>post:</b> showDAO.fetchAll().includes(show) and showDAO.fetchAll().size == @pre.showDAO.fetchAll().size()+1
<b>Nome metodo</b>	+update (Show show)
<b>Descrizione</b>	Questo metodo aggiorna le informazioni dello spettacolo.
<b>Pre-condizione</b>	<b>context:</b> ShowService:: update(Show show) <b>pre:</b> showDAO.fetch(show.id).equals(account)
<b>Post-condizione</b>	<b>context:</b> ShowService:: update(Show show) <b>post:</b> <b>post:</b> not showDAO.fetch(show.id).equals(@pre.showDAO.fetch(show.id))

### 3.7 Package PurchaseManager

<b>Nome classe</b>	<b>PurchaseService</b>
<b>Descrizione</b>	Questa classe permette di gestire gli acquisti e i biglietti.
<b>Metodi</b>	+fetchTickets(int showId): ArrayList<Ticket> +findTicket(int showId, char row, int seat): Boolean +insert(ArrayList<Ticket> ticketList): boolean +insert(Purchase purchase): int +fetchAll(int accountId, Paginator paginator): ArrayList<Purchase> +countAll(int accountId): int
<b>Invariante di classe</b>	/

<b>Nome Metodo</b>	+fetchTickets(int showId)
<b>Descrizione</b>	Questo metodo restituisce la lista di tutti i biglietti acquistati per un dato spettacolo, identificato da showId.
<b>Pre-condizione</b>	/
<b>Post-condizione</b>	/
<b>Nome metodo</b>	+findTicket(int showId, char row, int seat)
<b>Descrizione</b>	Questo metodo verifica che un biglietto è stato acquistato per un dato spettacolo, in una data poltrona. Viene utilizzato per verificare che il posto selezionato non sia occupato da altre persone.
<b>Pre-condizione</b>	/
<b>Post-condizione</b>	/
<b>Nome Metodo</b>	+insert(ArrayList<Ticket> ticketList)





<b>Descrizione</b>	Questo metodo permette di registrare un insieme di biglietti acquistati.
<b>Pre-condizione</b>	<b>context:</b> PurchaseService:: insert(ArrayList<Ticket>ticketList) <b>pre:</b> ticketList.size() <= 4 and ticketList.size() >= 1
<b>Post-condizione</b>	<b>context:</b> PurchaseService:: insert(ArrayList<Ticket>ticketList) <b>post:</b> ticketDAO.fetchTickets(showId).size() == @pre.ticketDAO.fetchTickets(showId).size()+ ticketList.size()
<b>Nome Metodo</b>	+insert(Purchase purchase)
<b>Descrizione</b>	Questo metodo permette di registrare l'acquisto effettuato.
<b>Pre-condizione</b>	<b>context:</b> PurchaseService:: +insert(Purchase purchase) <b>pre:</b> not purchaseDAO.fetchAll(accountId, paginator).includes(purchase)
<b>Post-condizione</b>	<b>context:</b> PurchaseService:: +insert(Purchase purchase) <b>post:</b> purchaseDAO.fetchAll(accountId, paginator).includes(purchase) and purchaseDAO.fetchAll(accountId, paginator).size() == @pre. purchaseDAO.fetchAll(accountId, paginator).size()+ 1
<b>Nome metodo</b>	+fetchAll(int accountId, Paginator paginator)
<b>Descrizione</b>	Questo metodo restituisce una lista di acquisti effettuati da un Utente Registrato, identificato dall'accountId.
<b>Pre-condizione</b>	/
<b>Post-condizione</b>	/
<b>Nome metodo</b>	+countAll(int accountId)
<b>Descrizione</b>	Questo metodo restituisce il numero di acquisti effettuati da un utente.
<b>Pre-condizione</b>	/
<b>Post-condizione</b>	/

## 4. Class Diagram

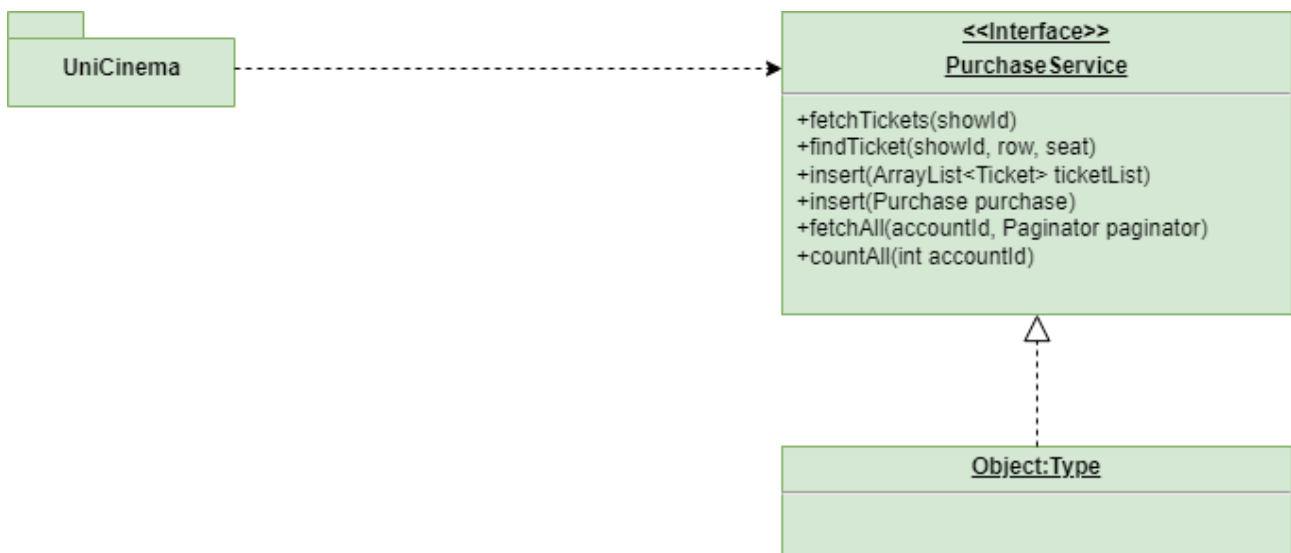
Per motivi di leggibilità, data la grandezza del class diagram, si riporta il link per visionarlo: <https://ibb.co/g99wbsY>

## 5. Design Pattern

Nella presente sezione verranno presentati i design pattern utilizzati per lo sviluppo del sistema.

### Facade

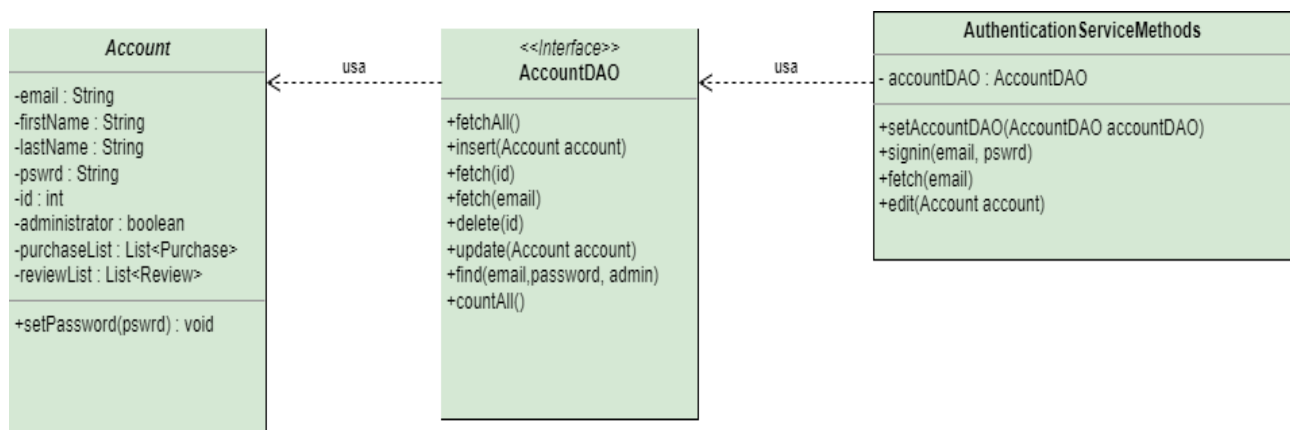
Il Facade è un design pattern che permette, implementando una interfaccia semplificata di accedere a sottosistemi più complessi. In questo modo si può nascondere al sistema la complessità delle librerie, dei framework o dei set di classi che si stanno usando. Si garantisce così un alto disaccoppiamento e si rende la piattaforma più manutenibile e più aggiornabile, poiché basterà cambiare l'implementazione dei metodi dell'interfaccia per implementare le modifiche. UniCinema, essendo un sistema molto complesso, sfrutta il design pattern Facade per implementare tutta la sua logica di business e rendere più facile l'interfacciarsi con essa. Nello specifico UniCinema utilizza il Facade per ogni suo sottosistema, implementandolo attraverso delle interfacce che sono usate per accedere ai metodi interni. Di seguito un esempio di Facade nel sistema UniCinema:



### DAO

Un DAO (Data Access Object) è un pattern che offre un'interfaccia astratta per alcuni tipi di database. Mappando le chiamate dell'applicazione allo stato persistente, il DAO fornisce alcune operazioni specifiche sui dati senza

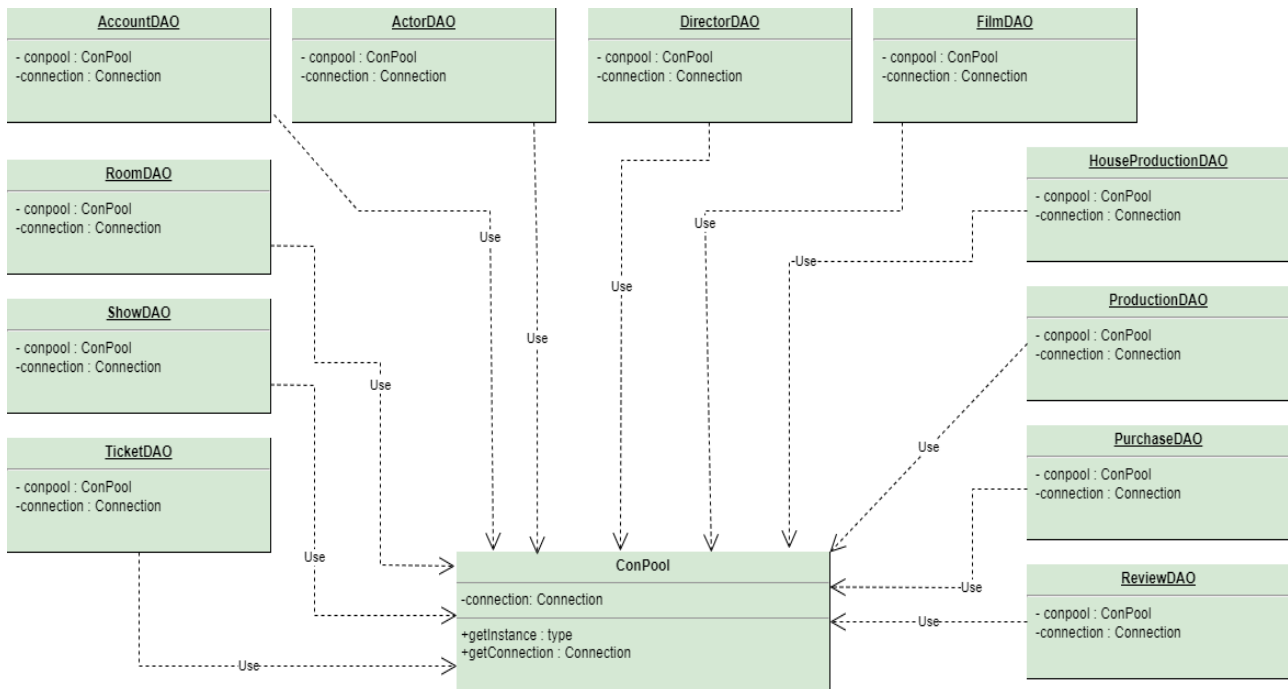
esporre i dettagli del database. I DAO sono utilizzabili nella maggior parte dei linguaggi e la maggior parte dei software con bisogni di persistenza, principalmente viene associato con applicazioni JavaEE che utilizzano database relazionali. UniCinema è una web application che presenta un database molto vasto; quindi, ha bisogno di poter interagire con il database in modo rapido e sicuro con numerose query per quella che è la moltitudine di dati da gestire. Il DAO, inoltre, ci consente di avere una separazione tra logica di business e persistenza.



## Singleton

Il Singleton è un pattern creazionale, ha lo scopo di istanziare oggetti garantendo la creazione di una sola istanza di quella classe, il Singleton pattern risolve la problematica relativa alla creazione di una sola connessione per ogni evocazione di un metodo DAO al fine di migliorar la performance.

La classe ConPool si occupa di creare e mantenere una connessione al database, è una classe singleton, così può essere acceduta in maniera atomica, senza creare molteplici connessioni.



## 6. Glossario

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



- **Singleton:** E' un design pattern creazionale che ha lo scopo di garantire che di una determinata classe venga strutturata una sola istanza e di fornire un punto di accesso globale a tale istanza.