EvoCinema

“ Sistema per la gestione di un cinema “

Versione 1.1

Object Design Document

Coordinatori del progetto

Prof. Andrea De Lucia - Top Manager
Francesco Vicidomini - Project Manager
Ferdinando D'Avino - Project Manager

Partecipanti

Luca Strefezza	0512102474	strluca94@gmail.com
Angelo Stefano D'Auria	0512102630	angelodauria91@gmail.com
Gianluca Villani	0512102990	lucassalerno1995@gmail.com
Giuseppe D'Ambrosio	0512103472	giuseppe.dambrosio14@gmail.com
Giuseppe Apuzzo	0512103920	g.apuzzo94@gmail.com
Sara De Filippo	0512103430	s.defilippo93@gmail.com
Antonio Giulio	0512103098	antonio.giulio96@gmail.com
Michele Delli Paoli	0512103820	mikeledellipaoli@gmail.com
Giuseppe Del Gaudio	0512103690	ciaogiuseppe96@gmail.com
Pietro Dell'Isola	0512103866	dellisola.pietro@gmail.com
Emanuele Buono	0512102370	squareman93@gmail.com
Francesco De Feo	0512103274	francescodefeo94@gmail.com

Revision history

Data	***Versione***	***Descrizione***	***Autori***
21/12/2017	1.0	Stesura ODD	Team Members
02/02/2017	1.1	Revisione ODD	Giuseppe D'Ambrosio

INDICE

1. Introduzione **5**
1.1. Object design trade-offs 5
1.2. Linee guida per la Documentazione delle Interfacce 7
1.3. Definizioni acronimi e abbreviazioni 7
1.4. Riferimenti

2. Packages **7**
2.1. EvoCinema 8
2.1.1. Model 8
2.1.2. Control 9
2.1.2.1. AccountCNT 10
2.1.2.2. SalaCNT 11
2.1.2.3. LibreriaCNT 12
2.1.2.4. ProgrammazioneCNT 13
2.1.2.5. ScontiCNT 14
2.1.2.6. PrenotazioniCNT 14
2.1.2.7. AcquistiCNT 15
2.1.2.8. AnalyticsCNT 16
2.1.3. View 17
2.1.3.1. Account 18
2.1.3.2. Sala 19
2.1.3.3. Libreria 20
2.1.3.4. Programmazione 21
2.1.3.5. Sconti 22
2.1.3.6. Prenotazioni 23
2.1.3.7. Acquisti 24
2.1.3.8. Analytics 25
2.1.4. Database 26
2.1.5. Utilities 27
2.1.6. Exception

3. Interfaccia delle classi **28**
3.1. Gestione Account 28
3.2. Gestione Sala 29
3.3. Gestione Libreria 30
3.4. Gestione Programmazione 30
3.5. Gestione Sconti 31
3.6. Gestione Prenotazioni 32
3.7. Gestione Acquisti 32
3.8. Gestione Analytics

4. Design Patterns **33**
4.1. Model-View-Controller 33
4.2. Singleton 33
4.3. Strategy 34
4.4. Data-Access-Object 36

1. Introduzione

1.1. Object design trade-offs

Dopo aver stilato il documento di Requirements Analysis e il documento di System Design in cui vi è una descrizione sommaria di ciò che sarà il nostro sistema, definendo i nostri obbiettivi ma tralasciando gli aspetti implementativi, andiamo ora a stilare il documento di Object Design che ha come obiettivo quello di produrre un modello che sia in grado di integrare in modo coerente e preciso tutte le funzionalità individuate nelle fasi precedenti.
In particolar modo, in tale documento si definiscono le interfacce delle classi, le operazioni, i tipi, gli argomenti e la signature dei sottosistemi definiti nel System Design. Inoltre sono specificati i trade-off e le linee guida.

Comprensibilità vs Tempo:
Il codice del sistema deve essere comprensibile il più possibile, in modo da facilitare la fase di testing ed eventuali future modifiche da apportare. Per rispettare queste linee guida il codice sarà accompagnato da commenti volti a semplificarne la comprensione. Ovviamente questo comporterà un aumento del tempo di sviluppo del nostro progetto.

Prestazioni vs Costi:
Dato che il nostro progetto è sprovvisto di budget, per poter mantenere prestazioni elevate, in determinate funzionalità verranno utilizzati dei template open source esterni, in particolare Bootstrap.

Interfaccia vs Usabilità:
L'interfaccia grafica è stata realizzata in maniera molto semplice, chiara e concisa, vengono utilizzati i form e pulsanti con lo scopo di rendere semplice l'utilizzo del sistema da parte dell'utente finale.

Sicurezza vs Efficienza:
La sicurezza, come descritto nei requisiti non funzionali del Requirements Analysis, rappresenta uno degli aspetti importanti del sistema. Tuttavia, dati i tempi di sviluppo molto limitati, ci limiteremo ad implementare sistemi di sicurezza basati su username e password degli utenti.

1.2. Linee guida per la Documentazione delle Interfacce

Gli sviluppatori dovranno seguire determinate linee guida per la stesura del codice:

Naming Convention:
È buona norma utilizzare nomi:

- Descrittivi
- Pronunciabili
- Di uso comune

- Di lunghezza medio-corta
- Non abbreviati
- Evitando la notazione ungherese
- Utilizzando solo caratteri consentiti (a-z, A-Z, 0-9)

Variabili:

- I nomi delle variabili devono iniziare con la lettera minuscola, e le parole successive con la lettera maiuscola. La dichiarazione delle variabili deve essere effettuata ad inizio blocco; in ogni riga vi deve essere una sola dichiarazione di variabile e va effettuato l'allineamento per migliorare la leggibilità.
- In determinati casi, è possibile utilizzare il carattere underscore "_", ad esempio quando si fa uso di variabili costanti oppure quando si fa uso di proprietà statiche.

Metodi:

- I nomi dei metodi devono iniziare con la lettera minuscola, e le parole successive con la lettera maiuscola. Di solito il nome del metodo è costituito da un verbo che identifica un'azione, seguito dal nome di un oggetto. I nomi dei metodi per l'accesso e la modifica delle variabili dovranno essere del tipo getNomeVariabile() e setNomeVariabile(). Se viene dichiarata una variabile all'interno di un metodo quest'ultima deve essere dichiarata appena prima del suo utilizzo e deve servire per un solo scopo, per facilitare la leggibilità. Esempio: getId(), setId()
- Ai metodi va aggiunta una descrizione, la quale deve essere posizionata prima della dichiarazione del metodo, e deve descriverne lo scopo. La descrizione del metodo deve includere anche informazioni riguardanti gli argomenti, il valore di ritorno, le eccezioni. I metodi devono essere raggruppati in base alla loro funzionalità.

Classi e pagine:

- I nomi delle classi e delle pagine devono iniziare con la lettera maiuscola, e anche le parole successive all'interno del nome devono iniziare con la lettera maiuscola. I nomi delle classi e delle pagine devono essere evocativi, in modo da fornire informazioni sullo scopo di quest'ultime.
  Ogni file sorgente .php contiene una singola classe e dev'essere strutturato in un determinato modo:
- Una breve introduzione alla classe. L'introduzione indica: l'autore, la versione e la data.

```
/**
* sommario dello scopo della classe.
*
* @author [nome dell'autore]
* @version [numero di versione della classe]
* @since [versione di partenza]
*/
```

- L'istruzione include che permette di importare all'interno della classe gli altri oggetti che la classe utilizza.

- La dichiarazione di una classe è caratterizzata da:
    1. Dichiarazione della classe pubblica
    2. Dichiarazioni di costanti
    3. Dichiarazioni di variabili di classe
    4. Dichiarazioni di variabili d'istanza
    5. Costruttore
    6. Commento e dichiarazione metodi e variabili

1.3. Definizioni acronimi e abbreviazioni

Acronimi:

- RAD: Requirements Analysis Document
- SDD: System Design Document
- ODD: Object Design Document

Abbreviazioni:

- DB: DataBase

1.4. Riferimenti

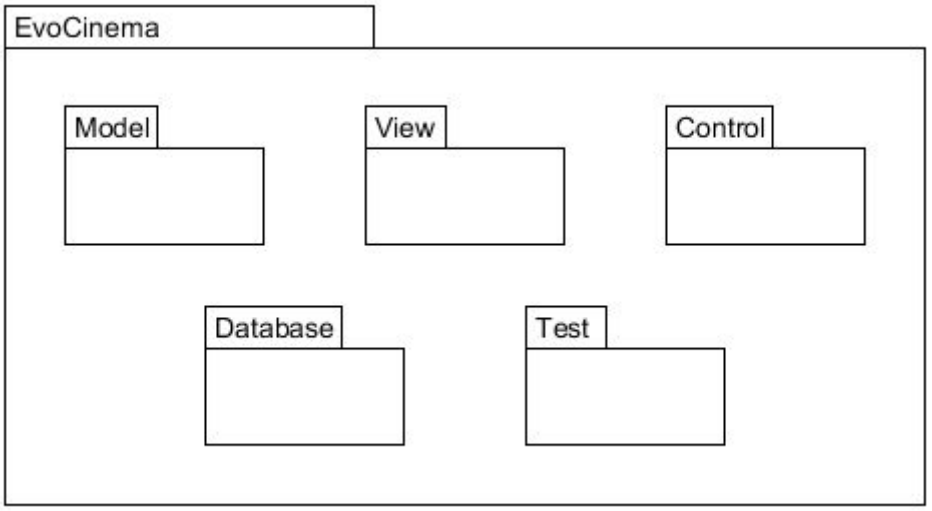
- B. Bruegge, A. H. Dutoit, Object Oriented Software Engineering - Using UML, Pattern and Java, Prentice Hall, 3rd edition, 2009
- Documento SDD del progetto Evocinema
- Documento RAD del progetto Evocinema
- Documento JavaDoc del progetto EvoCinema

2. Packages

La struttura del sistema EvoCinema è strutturata secondo una divisione in package e sottopackage che raggruppano le classi che hanno il compito di gestirne la logica in base alle richieste dell'utente che ne fa uso.

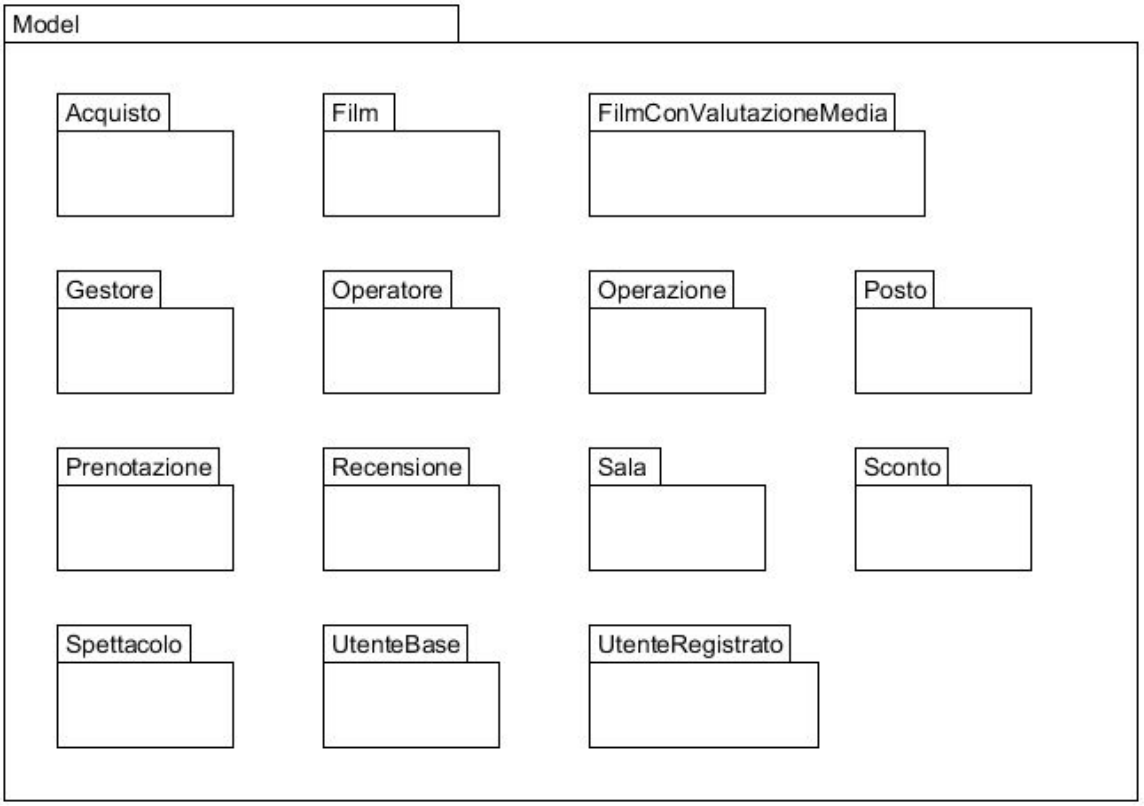
2.1. EvoCinema

Il package principale "EvoCinema" è presentato nel seguente schema.



2.1.1. Model

Il sottopackage “Model” è presentato nel seguente schema e contiene le classi Java rappresentanti le entità presenti all’interno del sistema.

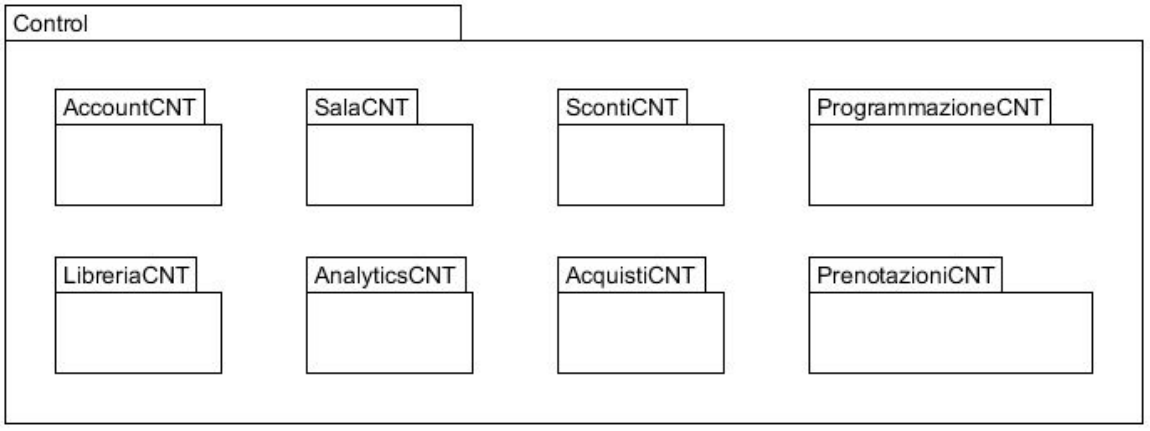


Classe:	**Descrizione:**
Acquisto	Descrive un acquisto effettuato.

Film	Descrive un film della libreria.
FilmConValutazioneMedia	Descrive un film della libreria che abbia una valutazione.
Gestore	Descrive un gestore del sistema.
Operatore	Descrive un operatore del sistema.
Operazione	Descrive una operazione generica effettuata.
Posto	Descrive un posto di una sala.
Prenotazione	Descrive una prenotazione per uno spettacolo.
Recensione	Descrive una recensione di un film.
Sala	Descrive una sala del cinema.
Sconto	Descrive uno sconto applicabile.
Spettacolo	Descrive uno spettacolo in programmazione.
UtenteBase	Descrive un utente che può effettuare acquisti.
UtenteRegistrato	Descrive un utente generico che raggruppa chi è registrato al sistema.

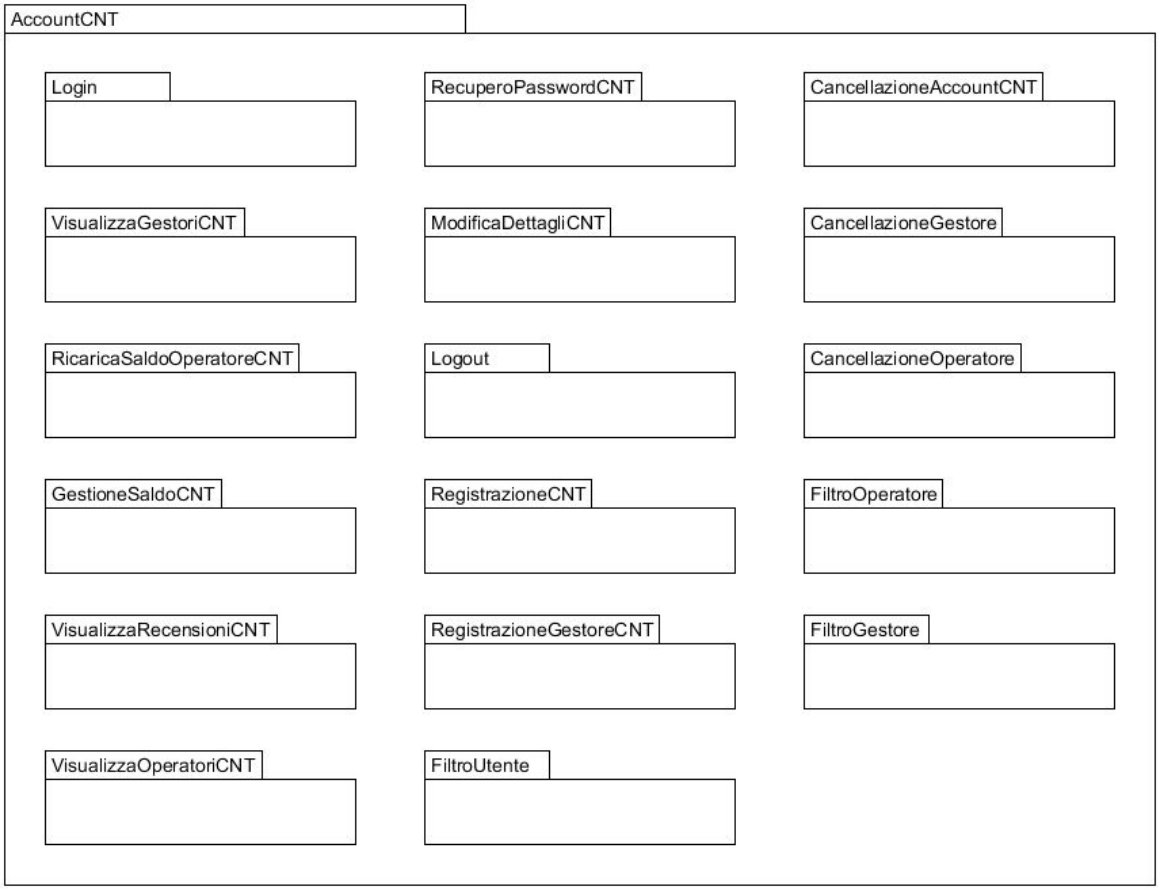
2.1.2. Control

Il sottopackage “Control” è presentato nel seguente schema e contiene le classi Java che si occupano della logica di business del sistema.



All'interno del sottopackage troviamo una ulteriore suddivisione in base alle diverse gestioni.

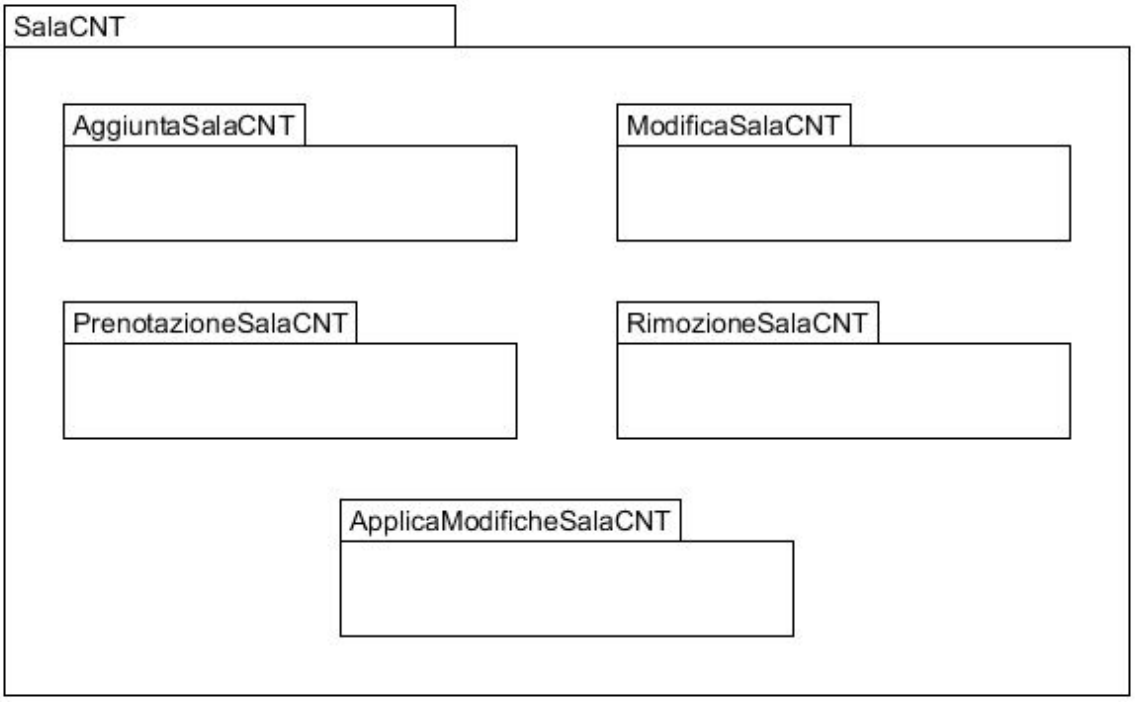
2.1.2.1. AccountCNT



Classe:	**Descrizione:**
CancellazioneAccountCNT	Permette di cancellare l'account di un UtenteBase.
CancellazioneGestore	Permette di cancellare l'account di un Gestore.
CancellazioneOperatore	Permette di cancellare l'account di un Operatore.
FiltroGestore	Permette la visualizzazione del sito nel ruolo di Gestore.
FiltroOperatore	Permette la visualizzazione del sito nel ruolo di Operatore.
FiltroUtente	Permette la visualizzazione del sito nel ruolo di UtenteBase.

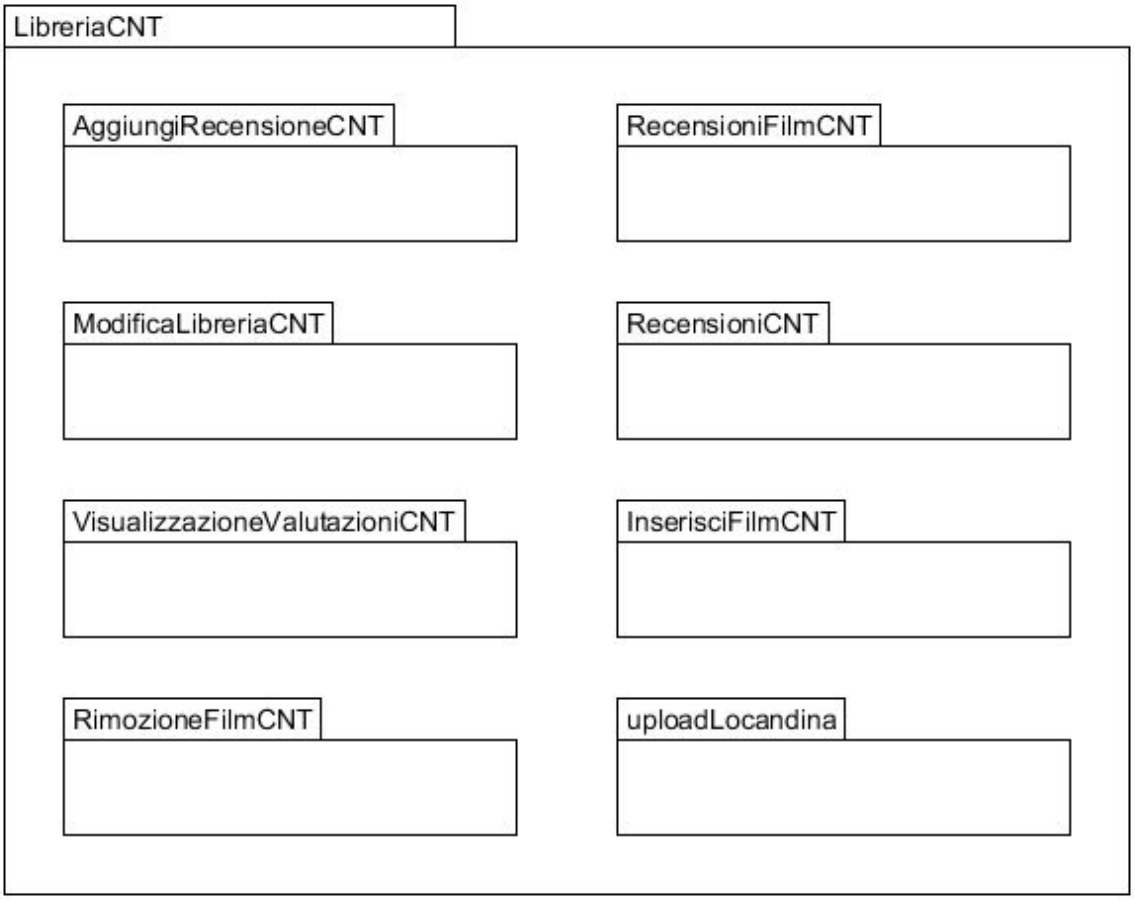
GestioneSaldoCNT	Permette la visione e la gestione del saldo.
Login	Permette l'accesso al sito.
Logout	Permette la disconnessione di un utente dal sito.
ModificaDettagliCNT	Permette la modifica del profilo.
RecuperoPasswordCNT	Permette di recuperare la password.
RegistrazioneCNT	Permette la registrazione di un nuovo utente.
RegistrazioneGestoreCNT	Permette la registrazione di un nuovo gestore.
RicaricaSaldoOperatoreCNT	Permette la ricarica del saldo da parte di un operatore.
VisualizzaGestoriCNT	Permette la visualizzazione di tutti i gestori.
VisualizzaOperatoriCNT	Permette la visualizzazione di tutti gli operatori.
VisualizzaRecensioniCNT	Permette la visualizzazione delle proprie recensioni.

2.1.2.2. SalaCNT



Classe:	**Descrizione:**
AggiuntaSalaCNT	Permette l'aggiunta di una sala.
ModificaSalaCNT	Permette la modifica di una sala.
PrenotazioneSalaCNT	Permette la prenotazione di una sala.
RimozioneSalaCNT	Permette la rimozione di una sala.
ApplicaModicheSalaCNT	Permette di applicare le modifiche effettuate ad una sala.

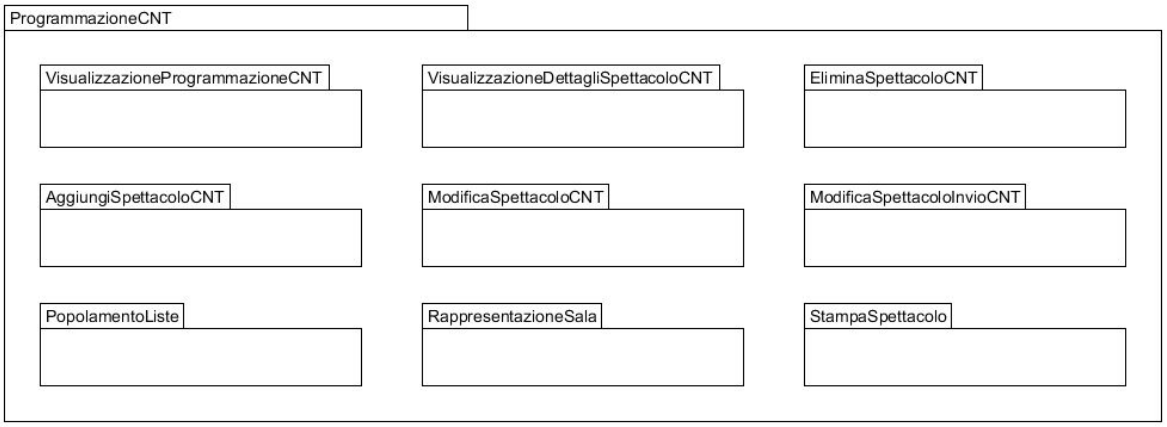
2.1.2.3. LibreriaCNT



Classe:	**Descrizione:**
AggiungiRecensioneCNT	Permette di gestire l'aggiunta di una recensione.
ModificaLibreriaCNT	Permette di gestire la modifica di un film.
RecensioniFilmCNT	Permette di gestire il caricamento di una serie di recensioni di un film.
RecensioniCNT	Permette di gestire l'aggiunta di una recensione di un film.
VisualizzaValutazioniCNT	Permette di gestire il caricamento duna serie di valutazoni di un film.
InserisciFilmCNT	Permette di inserire un film.
RimozioneFilmCNT	Permette di gestire la rimozione di un film.
uploadLocandina	Permette di gestire l'aggiunta di una nuova

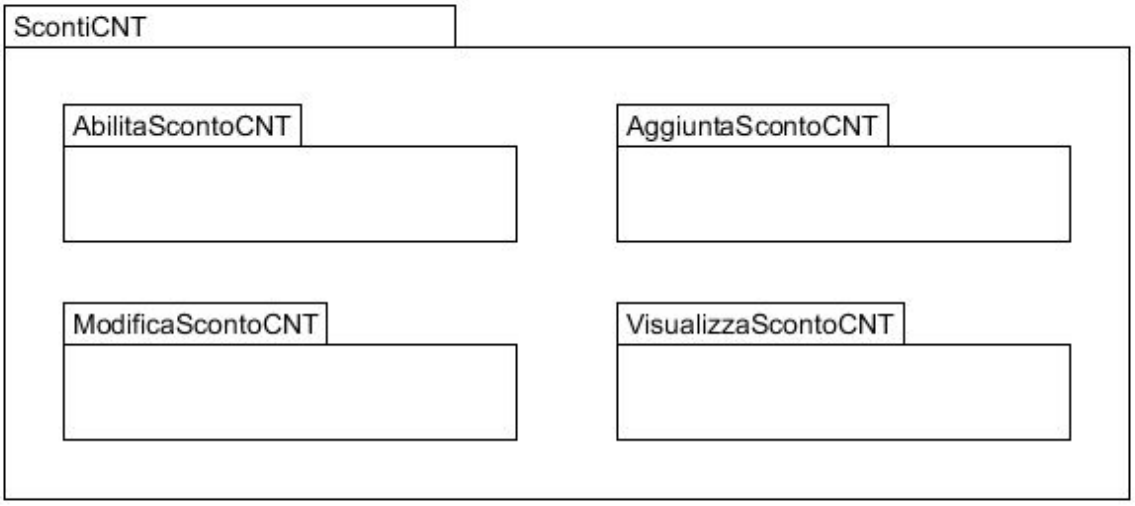
	locandina.

2.1.2.4. ProgrammazioneCNT



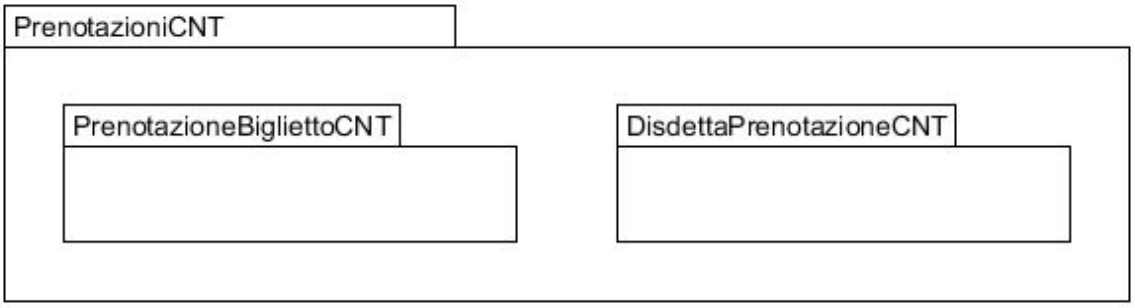
Classe:	**Descrizione:**
VisualizzazioneProgrammazioneCNT	Permette la visualizzazione della programmazione.
VisualizzazioneDettagliSpettacoloCNT	Permette la visualizzazione dei dettagli di uno spettacolo.
AggiungiSpettacoloCNT	Permette l'aggiunta di uno spettacolo.
ModificaSpettacoloCNT	Permette la modifica di uno spettacolo.
EliminaSpettacoloCNT	Permette l'eliminazione di uno spettacolo.
ModificaSpettacoloInvioCNT	Permette di rendere effettive le modifiche effettuate.
PopolamentoListe	Permette di gestire il corretto riempimento delle liste di oggetti.
RappresentazioneSala	Permette di gestire la visualizzazione della sala di uno spettacolo.
StampaSpettacolo	Permette di gestire la visualizzazione di uno spettacolo.

2.1.2.5. ScontiCNT



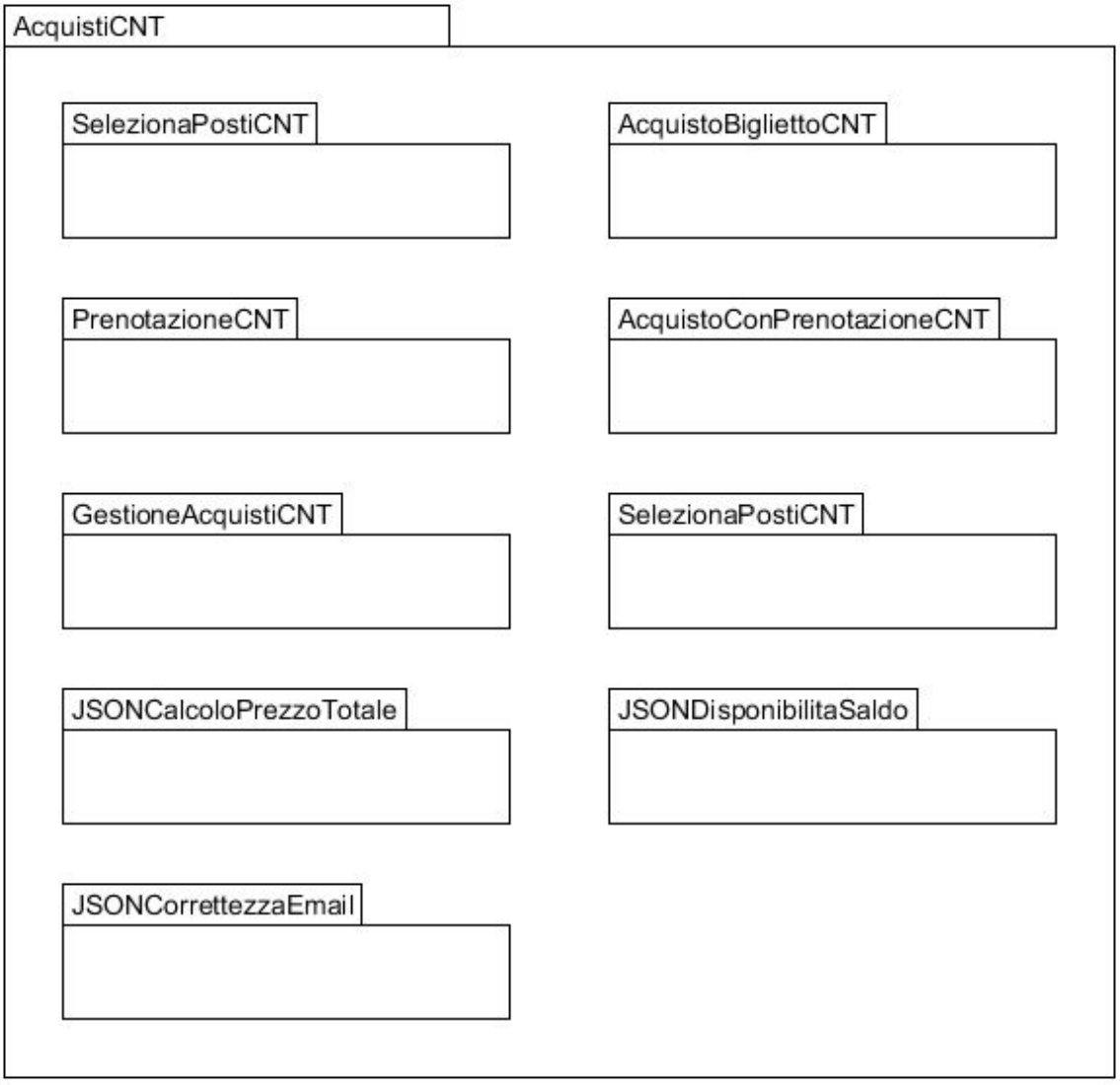
Classe:	**Descrizione:**
AbilitaScontoCNT	Permette di rendere uno sconto applicabile.
AggiuntaScontoCNT	Permette di aggiungere uno sconto.
ModificaScontoCNT	Permette di modificare uno sconto.
VisualizzaScontoCNT	Permette la visualizzazione di uno sconto.

2.1.2.6. PrenotazioniCNT



Classe:	**Descrizione:**
PrenotazioneBiglietto CNT	Permette la prenotazione di uno o più posti per uno spettacolo.
DisdettaPrenotazioneCNT	Permette di disdire una prenotazione.

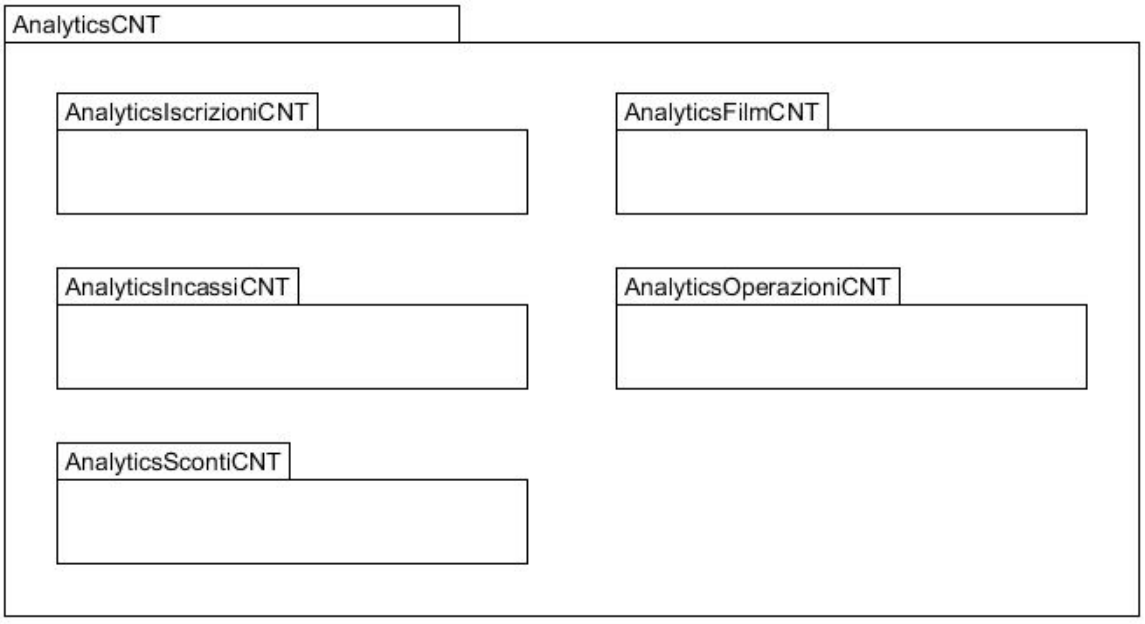
2.1.2.7. AcquistiCNT



Classe:	**Descrizione:**
SelezionaPostiCNT	Permette di visualizzare la griglia dei posti e selezionare quelli desiderati.
AcquistoBigliettoCNT	Permette l'acquisto di un biglietto.
PrenotazioneCNT	Permette di acquistare un biglietto in seguito ad una prenotazione.
AcquistoConPrenotazioneCNT	Permette l'acquisto di un biglietto in seguito ad una prenotazione.
GestioneAcquistiCNT	Permette di gestire gli acquisti da parte di un operatore.

PrenotazioneCNT	Permette di gestire la prenotazone di un biglietto.
SelezionePostiCNT	Permette la selezione dei posti di un biglietto.
JSONCalcoloPrezzoTotale	Permette l'aggiornamento istantaneo del prezzo totale.
JSONDisponibilitaSaldo	Permette il controllo istantaneo della disponibilità del saldo.
JSONCorrettezzaEmail	Permette il controllo istantaneo della correttezza di una mail.

2.1.2.8. AnalyticsCNT

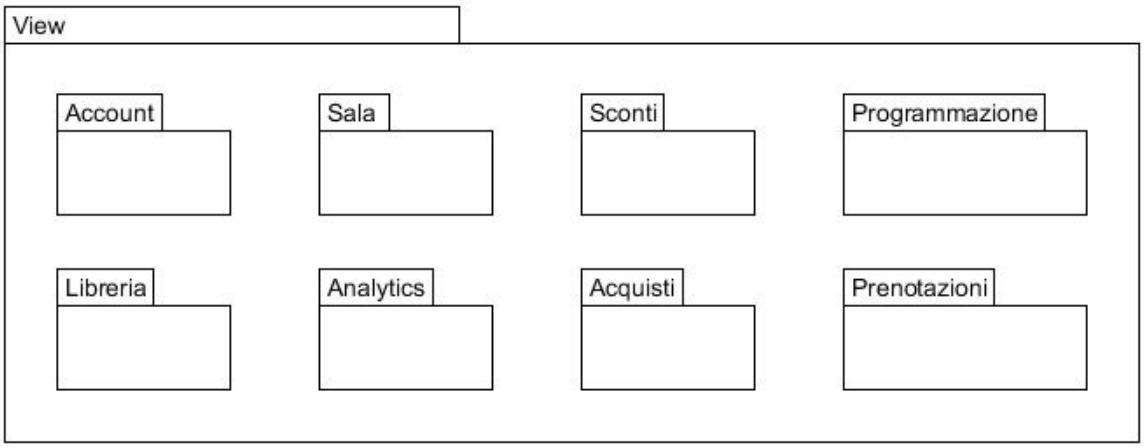


Classe:	**Descrizione:**
AnalyticsIscrizioniCNT	Permette di ottenere le analytics delle iscrizioni.
AnalyticsFilmCNT	Permette di ottenere le analytics dei film.
AnalyticsOperazioniCNT	Permette di ottenere le analytics delle operazioni.
AnalyticsScontiCNT	Permette di ottenere le analytics degli sconti.
AnalyticsIncassiCNT	Permette di ottenere le analytics degli

	incassi.

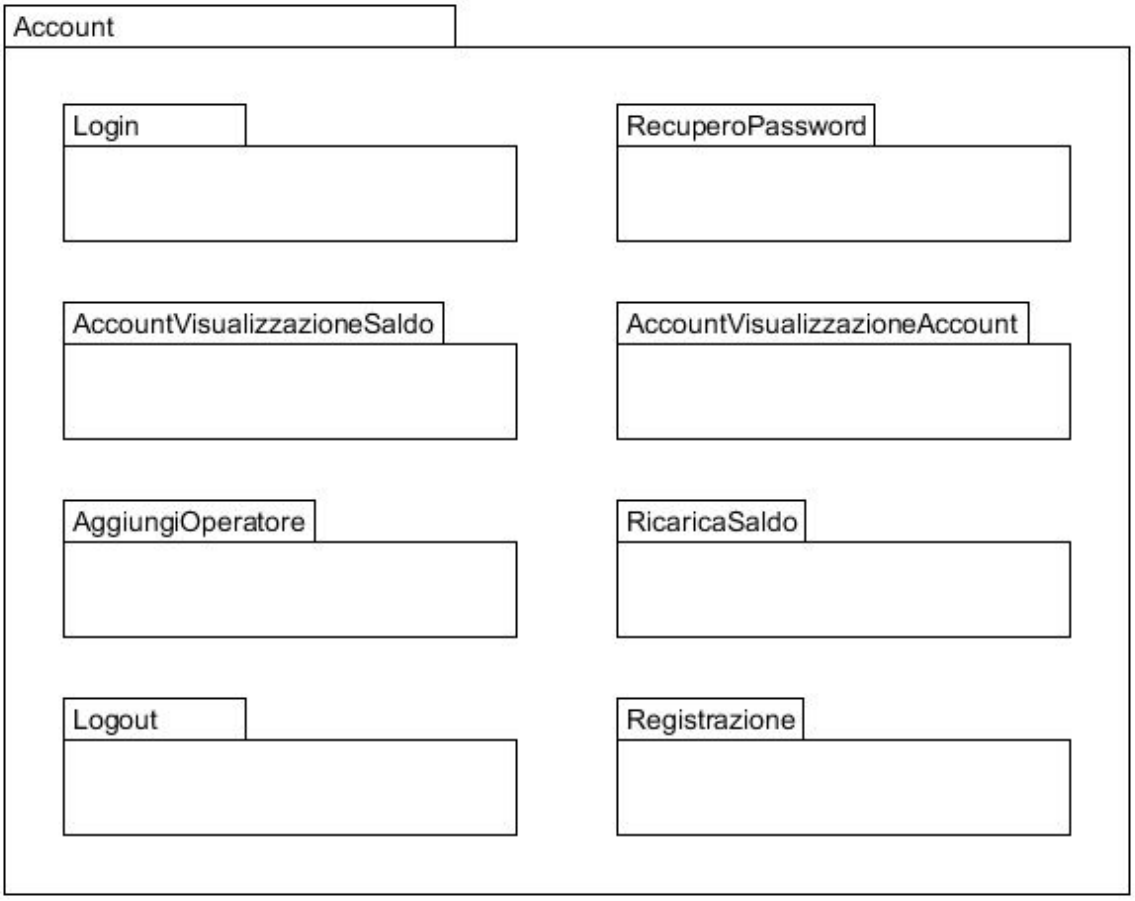
2.1.3. View

Il sottopackage “View” è presentato nel seguente schema e contiene le JSP rappresentanti le schermate del sistema che vengono visualizzate.



All’interno del sottopackage troviamo una ulteriore suddivisione in base alle diverse gestioni.

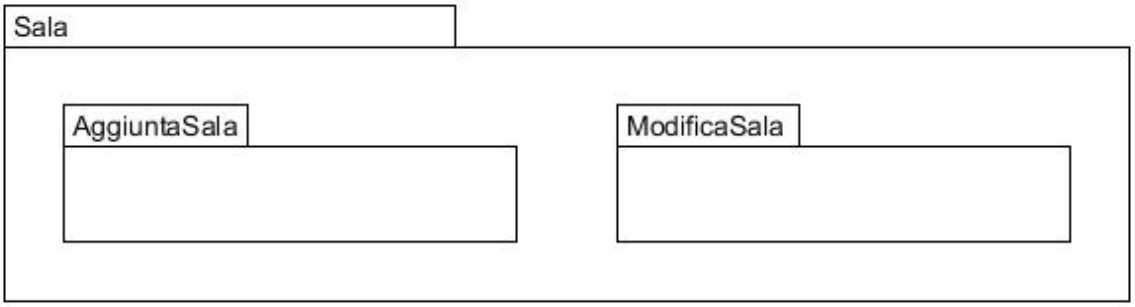
2.1.3.1. Account



Classe:	**Descrizione:**
Login	Rappresenta la schermata di accesso al sistema.
Logout	Rappresenta la schermata per l'uscita dal sistema.
AccountVisualizzazioneSaldo	Rappresenta la schermata di visualizzazione del saldo di un utente..
AccountVisualizzazioneAccount	Rappresenta la schermata per la visualizzazione dei dettagli dell'account.
RecuperoPassword	Rappresenta la schermata per il recupero della password.
AggiungiOperatore	Rappresenta la schermata per la visualizzazione dell'aggiunta di un operatore.

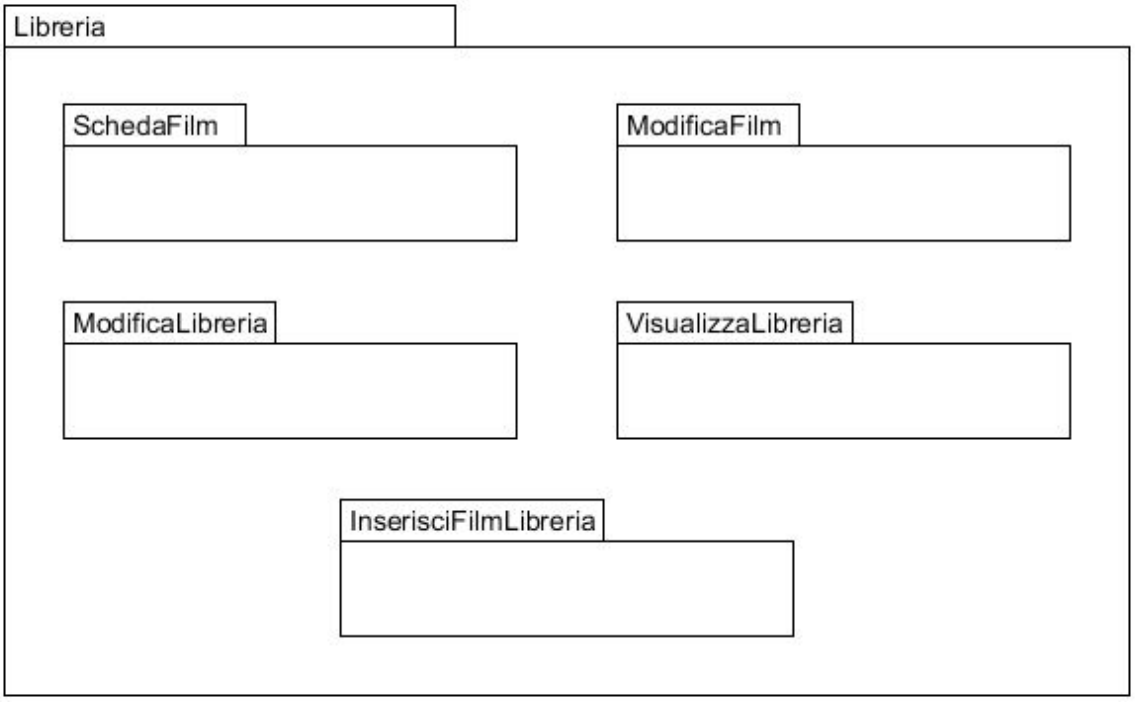
Registrazione	Rappresenta la schermata per la registrazione.
RicaricaSaldo	Rappresenta la schermata che permette la ricarica del saldo.

2.1.3.2. Sala



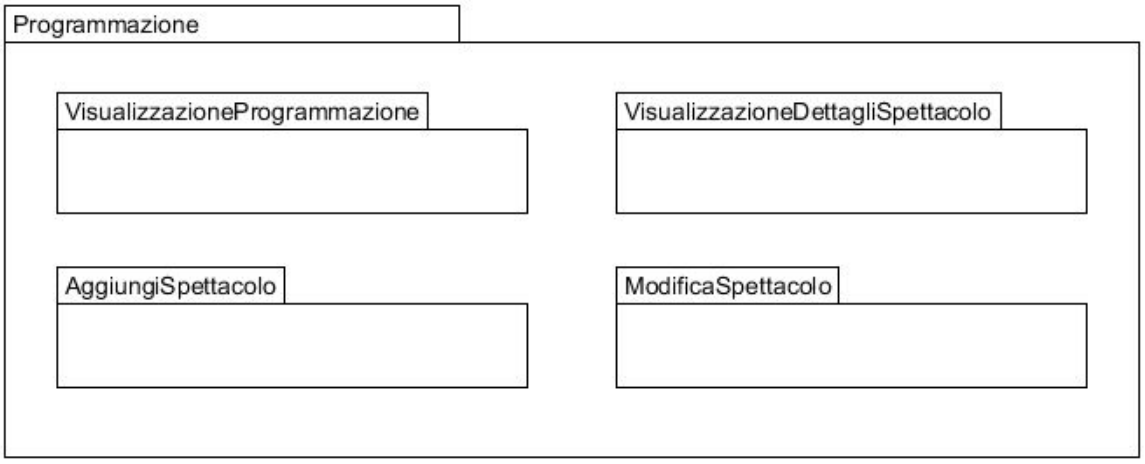
Classe:	**Descrizione:**
AggiuntaSala	Rappresenta la schermata che permette l'aggiunta di una sala.
ModificaSala	Rappresenta la schermata che permette la modifica di una sala.

2.1.3.3. Libreria



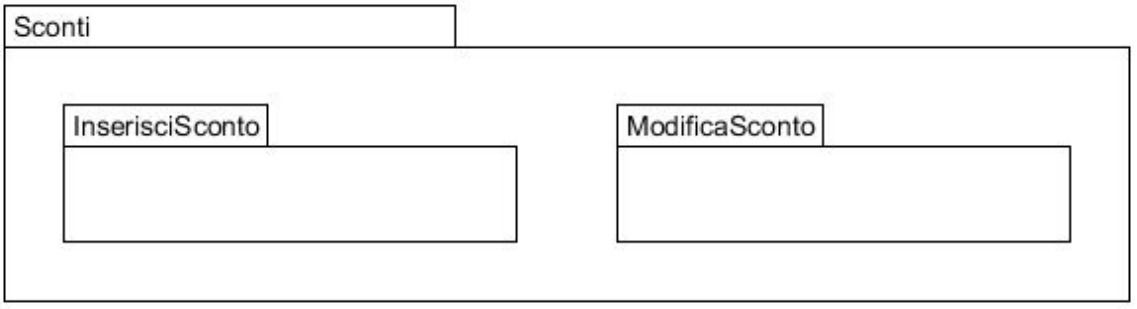
Classe:	**Descrizione:**
SchedaFilm	Rappresenta la schermata che mostra la scheda di un film.
InserisciFilmLibreria	Rappresenta la schermata che permette di inserire un film.
ModificaFilm	Rappresenta la schermata che permette di modificare un film.
ModificaLibreria	Rappresenta la schermata per la visualizzazione della modifica della libreria.
VisualizzaLibreria	Rappresenta la schermata per la visualizzazione della libreria.

2.1.3.4. Programmazione



Classe:	**Descrizione:**
VisualizzazioneProgrammazione	Rappresenta la schermata che mostra l'intera programmazione.
VisualizzazioneDettagliSpettacolo	Rappresenta la schermata che mostra i dettagli di uno spettacolo.
AggiungiSpettacolo	Rappresenta la schermata che permette l'aggiunta di uno spettacolo.
ModificaSpettacolo	Rappresenta la schermata che permette la modifica di uno spettacolo.

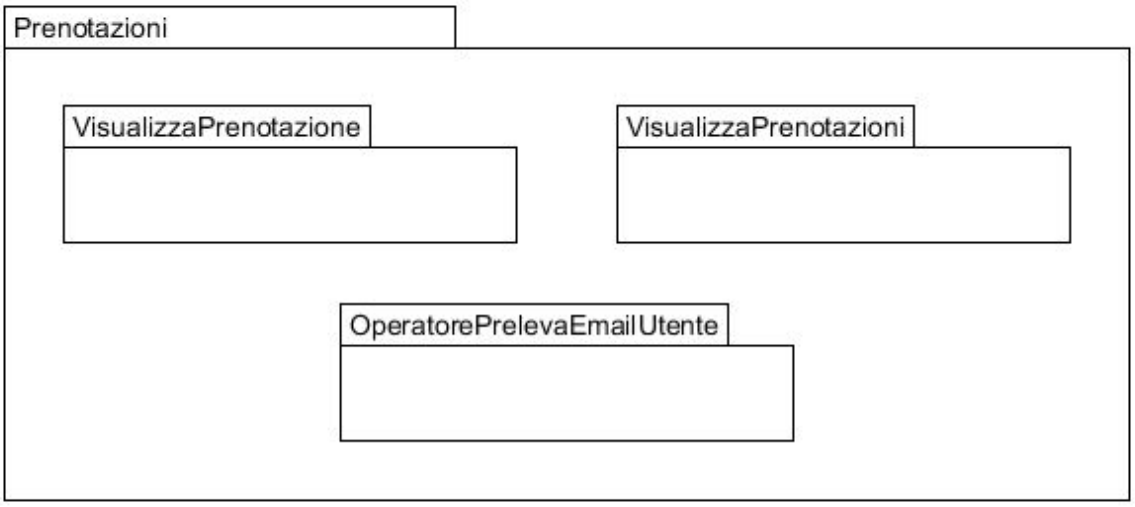
2.1.3.5. Sconti



Classe:	**Descrizione:**
InserisciSconto	Rappresenta la schermata che permette di aggiungere uno sconto.

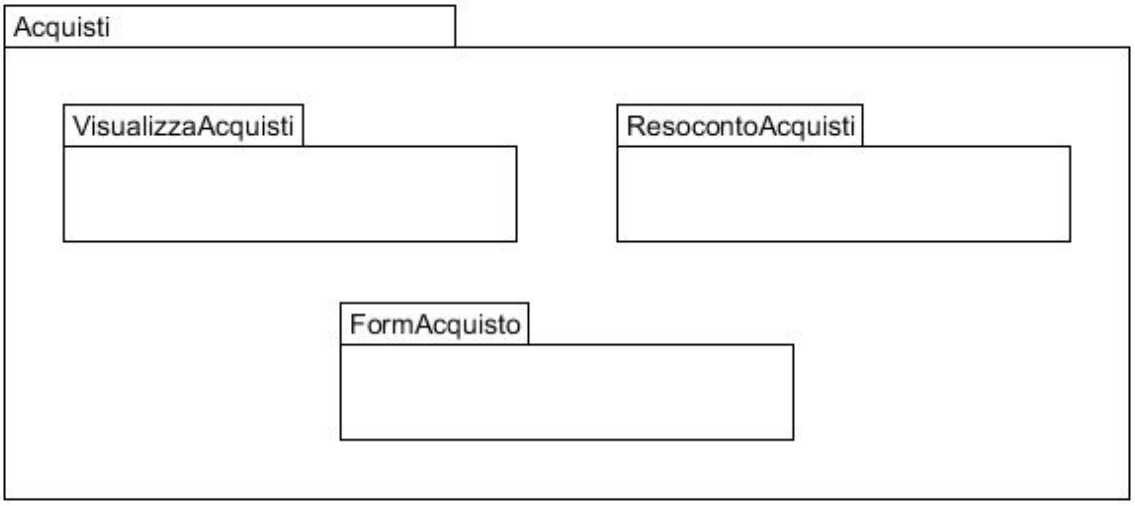
ModificaSconto	Rappresenta la schermata che permette la modifica di uno sconto.

2.1.3.6. Prenotazioni



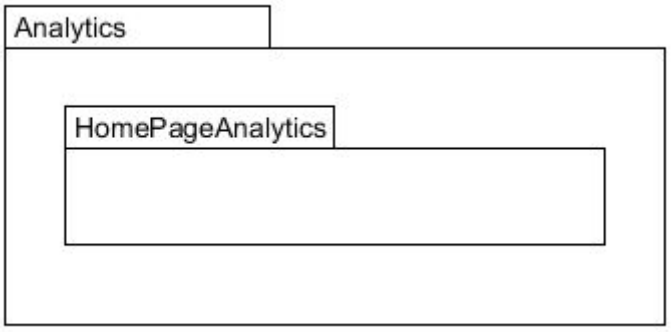
Classe:	**Descrizione:**
VisualizzaPrenotazione	Rappresenta la schermata che permette di visualizzare una prenotazione.
VisualizzaPrenotazioni	Rappresenta la schermata che di visualizzare le prenotazioni di un utente.
OperatorePrelevaEmailUtente	Rappresenta la schermata che permette di accedere alle prenotazioni di un utente tramite email.

2.1.3.7. Acquisti



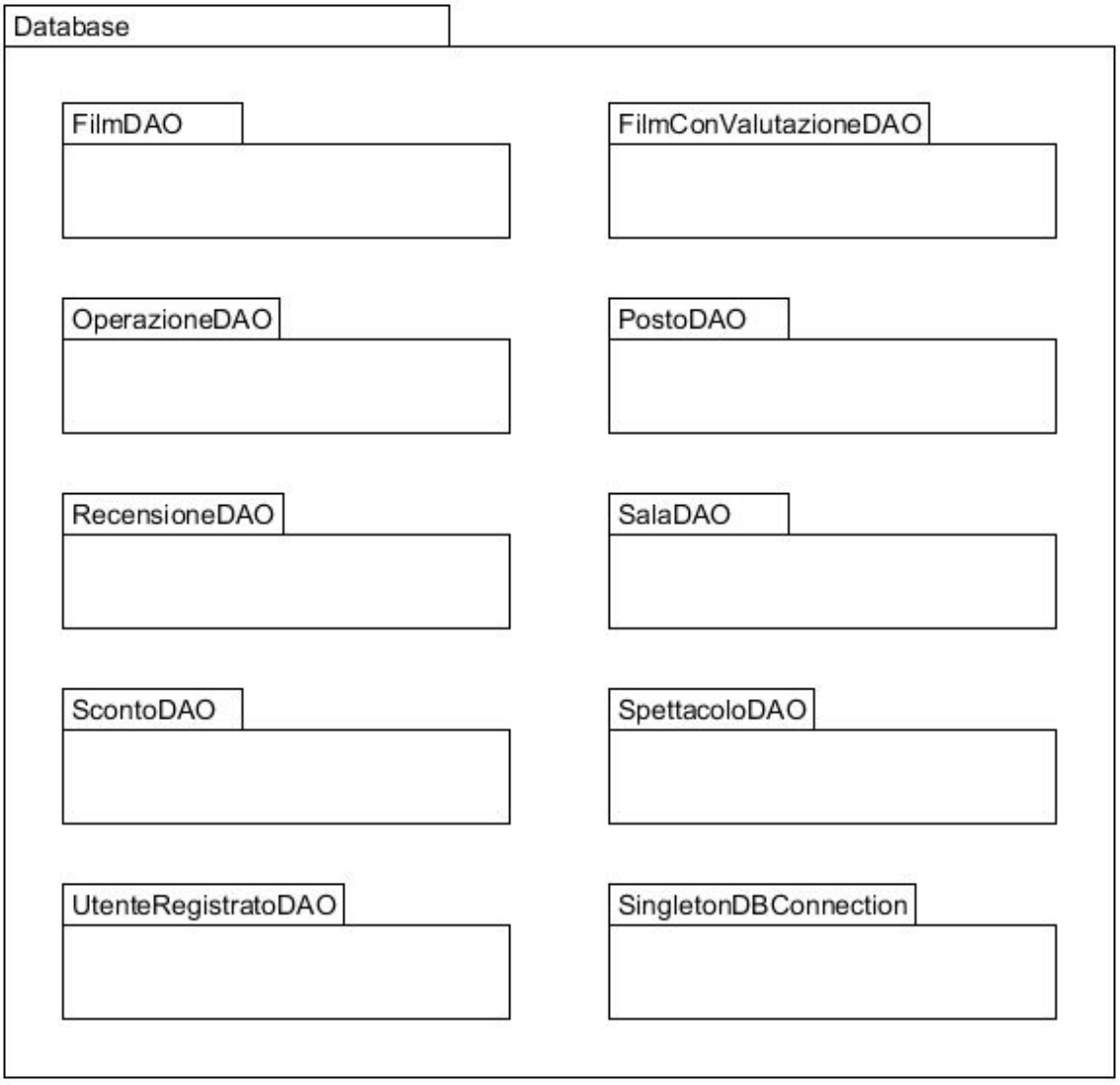
Classe:	**Descrizione:**
FormAcquisto	Rappresenta la schermata che permette di effettuare un acquisto.
ResocontoAcquisto	Rappresenta la schermata che mostra il resoconto degli acquisti effettuati.
VisualizzaAcquisti	Rappresenta la schermata che permette di visualizzare gli acquisti effettuati.

2.1.3.8. Analytics



Classe:	**Descrizione:**
HomePageAnalytics	Rappresenta la schermata principale della sezione Analytics.

2.1.4. Database



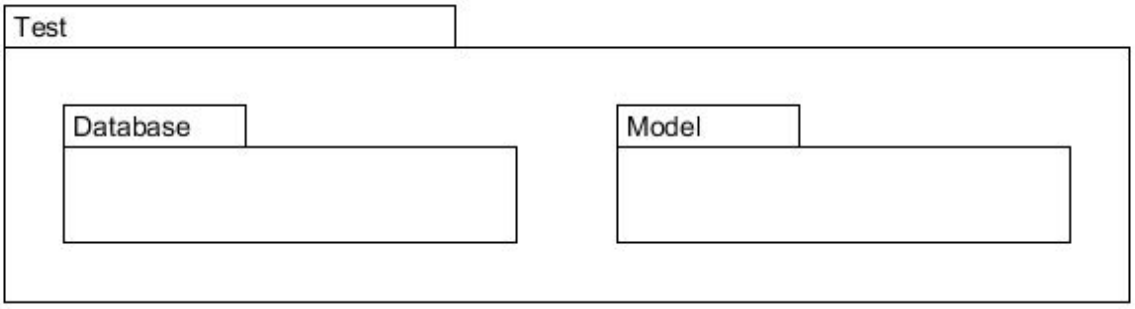
Il sottopackage “Database” è presentato nel seguente schema e contiene le classi Java che si occupano delle interazioni con il Database.

Classe:	**Descrizione:**
FilmDAO	Gestisce le operazioni che interessano l’entità Film all’interno del database.
FilmConValutazioneDAO	Gestisce le operazioni che interessano l’entità FilmConValutazioneMedia all’interno del database.
OperazioneDAO	Gestisce le operazioni che interessano l’entità Operazione all’interno del database.

PostoDAO	Gestisce le operazioni che interessano l'entità Posto all'interno del database.
RecensioneDAO	Gestisce le operazioni che interessano l'entità Recensione all'interno del database.
SalaDAO	Gestisce le operazioni che interessano l'entità Sala all'interno del database.
ScontoDAO	Gestisce le operazioni che interessano l'entità Sconto all'interno del database.
SpettacoloDAO	Gestisce le operazioni che interessano l'entità Spettacolo all'interno del database.
UtenteRegistratoDAO	Gestisce le operazioni che interessano l'entità UtenteRegistrato all'interno del database.
SingletonDBConnection	Gestisce le operazioni necessarie ad istanziare la connessione con il database.

2.1.5. Test

Il sottopackage "Test" è presentato nel seguente schema e contiene le classi Java che si occupano di testare il corretto funzionamento delle classi implementate.



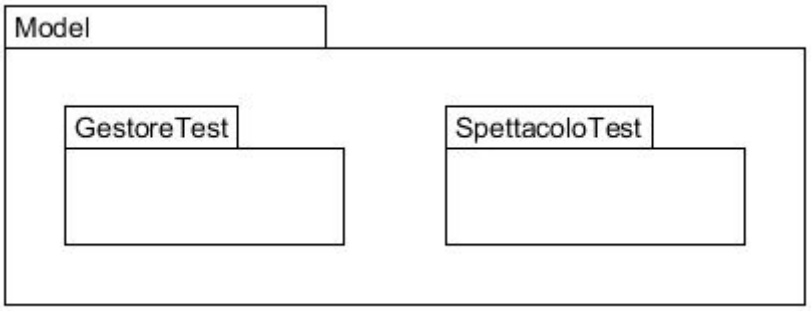
All'interno del sottopackage troviamo una ulteriore suddivisione in base alle tipologie di componenti testate.

2.1.5.1. Database

Classe:	**Descrizione:**
FilmDAOTest	Si occupa del testing dei metodi della classe FilmDAO.
FilmValutazioneDAOTest	Si occupa del testing dei metodi della classe FilmConValutazioneDAO.
OperazioneDAOTest	Si occupa del testing dei metodi della classe OperazioneDAO.
RecensioneDAOTest	Si occupa del testing dei metodi della classe RecensioneDAO.
SalaDAOTest	Si occupa del testing dei metodi della classe SalaDAO.
ScontoDAOTest	Si occupa del testing dei metodi della classe ScontoDAO.
SpettacoloDAOTest	Si occupa del testing dei metodi della classe SpettacoloDAO.

UtenteRegistratoDAOTest	Si occupa del testing dei metodi della classe UtenteRegistratoDAO.

2.1.5.2. Model



Classe:	Descrizione:
GestoreTest	Si occupa del testing dei metodi della classe Gestore.
SpettacoloTest	Si occupa del testing dei metodi della classe Spettacolo.

3. Interfaccia delle classi

3.1. Gestione Account

- public void login (String email, String password)
  Effettua il login di un utente.

- public void logout ()
  Effettua il logout di un utente.

- public void registrazioneOperatore (String nomeUtente, String password, String nome, String cognome, String email, String cellulare, String indirizzo, GregorianCalendar dataNascita, String sesso)
  Registra un nuovo account operatore.

- public void registrazioneUtente (String email, String nomeUtente, String password, String nome, String cognome, String cellulare, String indirizzo, GregorianCalendar dataNascita, String sesso)
  Registra un nuovo account utente base.

- public void cancellaAccount (String email)
  Permette di cancellare un account di un utente.

- public void recuperoPassword (String email)
  Recupera la password di un utente.

- public String visualizzaAccount (String email)
  Visualizza le informazioni dell'account di un utente.

- public void modificaAccount (String nomeUtente, String password, String nome, String cognome, String email, String cellulare, String indirizzo, GregorianCalendar dataNascita, String sesso)
  Modifica le informazioni dell'account di un utente.

- public float visualizzaSaldo (String email)
  Visualizza il saldo di un utente.

- public float ricaricaSaldo (float saldo, String email)
  Ricarica il saldo di un utente

3.2. Gestione Sala

- public void aggiuntaSala (int numeroPosti)
  Crea una nuova sala.

- public void modificaSala (int idSala, int numeroPosti)
  Modifica i dettagli di una sala.

- public void prenotazioneSala (intidSala)
  Occupa un'intera sala.

3.3. Gestione Libreria

- public ArrayList getLibreria ()
  Visualizza la libreria completa.

- public String insertRecensioneFilm (String email, String testo, float valutazione)
  Recensisce un film.

- public void visualizzaSchedaFilm (int idFilm)
  Visualizza i dettagli di un film.

- public void insertFilm (String titolo, String descrizione, String genere, GregorianCalendar annoUscita, String locandina, String regia, String cast, String distribuzione)
  Inserisce un nuovo film in libreria.

- public void modificaFilm (String titolo, String descrizione, String genere, GregorianCalendar annoUscita, String locandina, String regia, String cast, String distribuzione)
  Modifica un film in libreria.

- public void rimozioneFilm (int idFilm)
  Rimuove un film dalla libreria..

3.4. Gestione Programmazione

- public ArrayList getProgrammazione ()
  Visualizza tutti i film in programmazione.

- public void getDettagliSpettacolo (int idSpettacolo)
  Visualizza i dettagli di uno spettacolo in programmazione.

- public void insertSpettacolo (GregorianCalendar orario, GregorianCalendar dataInizio, GregorianCalendar dataFine, float prezzo, Sala sala)
  Inserisce un nuovo spettacolo in programmazione.

- public void modificaSpettacolo (int idSpettacolo, GregorianCalendar orario, GregorianCalendar dataInizio, GregorianCalendar dataFine, float prezzo, Sala sala)
  Modifica uno spettacolo in programmazione.

- public void eliminaSpettacolo(int idSpettacolo)
  Elimina uno spettacolo dalla programmazione

3.5. Gestione Sconti

- public void insertPoliticaSconto (String nome, int percentuale, float prezzo, String parametroTipologia)
  Inserisce una nuova politica di sconto.

- public void modificaPoliticaSconto (String idSconto, String nome, int percentuale, float prezzo, String parametroTipologia)
  Modifica una politica di sconto.

- public void abilitaPoliticaSconto (int idSconto)
  Abilita una politica di sconto.

- public void disabilitaPoliticaSconto (intidSconto)
  Disabilita una politica di sconto.

- public ArrayList getPoliticaSconto ()
  Visualizza tutte le politiche di sconto.

3.6. Gestione Prenotazioni

- public String (String idSala, String configPosti, String email, Calendar data)
  Prenota dei posti per uno spettacolo.

- public void disdettaPrenotazione (String idPrenotazione, String email)
  Disdice una prenotazione effettuata.

3.7. Gestione Acquisti

- public void acquisto (String idSpettacolo, float prezzoFinale, Calendar data, String email)
  Acquista un biglietto di uno spettacolo.

3.8. Gestione Analytics

- public float getListaIncassi ()
  Visualizza tutti gli incassi.

- public ArrayList getListaIscrizioni ()
  Visualizza tutte le informazioni degli utenti.

- public ArrayList getListaFilm ()
  Visualizza tutte le informazioni riguardanti i film.

- public ArrayList requestListaSconti ()
  Visualizza tutte le informazioni sugli sconti.

- public ArrayList requestListaOperazioni ()
  Visualizza tutte le operazioni effettuate dagli operatori.

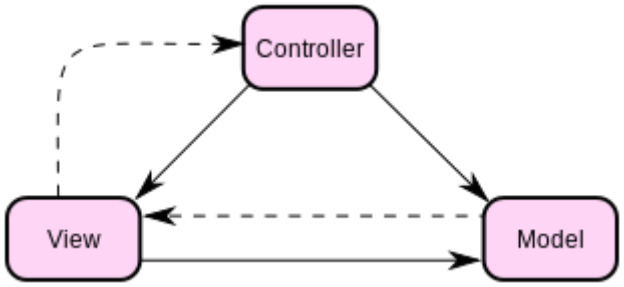
4. Design Patterns

4.1. Model-View-Controller

Il Model-View-Controller (MVC, talvolta tradotto in italiano con la dicitura Modello-Vista-Controllo),Questo pattern si posiziona nel livello di presentazione in una Architettura multi-tier.

Il pattern è basato sulla separazione dei compiti fra i componenti software che interpretano tre ruoli principali:

- il model fornisce i metodi per accedere ai dati utili all'applicazione;
- il view visualizza i dati contenuti nel model e si occupa dell'interazione con utenti e agenti;
- il controller riceve i comandi dell'utente (in genere attraverso il view) e li attua modificando lo stato degli altri due componenti.

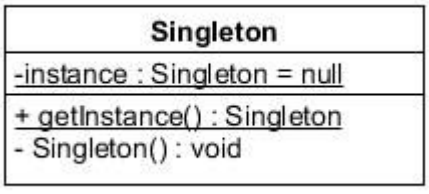


Questo schema, implica anche la tradizionale separazione fra la logica applicativa (in questo contesto spesso chiamata "logica di business"), a carico del *controller* e del *model*, e l'interfaccia utente a carico del *view*.

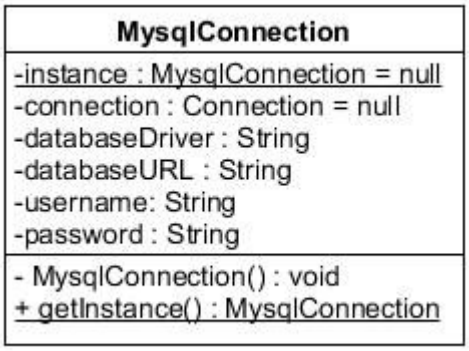
4.2. Singleton

Per aumentare l'efficienza e garantire la sicurezza del sistema è stato deciso di mantenere una sola istanza di connessione al database tramite l'utilizzo del design pattern Singleton.

Il Singleton è un pattern creazionale che ha lo scopo di garantire che di una determinata classe venga creata una e una sola istanza, e di fornire un punto di accesso globale a tale istanza.
L'implementazione di questo pattern prevede che la classe singleton abbia un unico costruttore privato, in modo da impedire l'istanziazione diretta della classe. La classe fornisce inoltre un metodo "getter" statico che restituisce l'istanza della classe (sempre la stessa), creandola preventivamente o alla prima chiamata del metodo, e memorizzandone il riferimento in un attributo privato anch'esso statico. Il secondo approccio si può classificare come basato sul principio della lazy initialization (letteralmente "inizializzazione pigra") in quanto la creazione dell'istanza della classe viene rimandata nel tempo e messa in atto solo quando ciò diventa strettamente necessario (al primo tentativo di uso).
Di seguito il diagramma UML di una classe singleton:

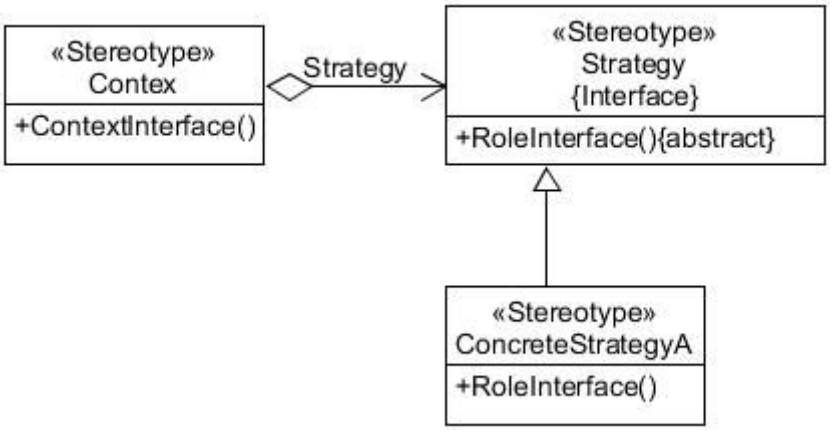


Un esempio di singleton il nostro sistema:



4.3. Strategy

Il pattern Strategy permette di definire una famiglia di algoritmi, di incapsularli e renderli intercambiabili fra loro. Questo pattern consente agli algoritmi di variare in modo indipendente rispetto al loro contesto di utilizzo, fornendo un basso accoppiamento tra le classi partecipanti del pattern e un'alta coesione funzionale delle diverse strategie di implementazione.
Questo pattern prevede che gli algoritmi siano intercambiabili tra loro, in base ad una specificata condizione, in modalità trasparente al client che ne fa uso. In altre parole, data una famiglia di algoritmi che implementa una certa funzionalità, come può essere ad esempio un algoritmo di visita oppure di ordinamento, essi dovranno esportare sempre la medesima interfaccia, così il client dell'algoritmo non dovrà fare nessuna assunzione su quale sia la strategia istanziata in un particolare istante.

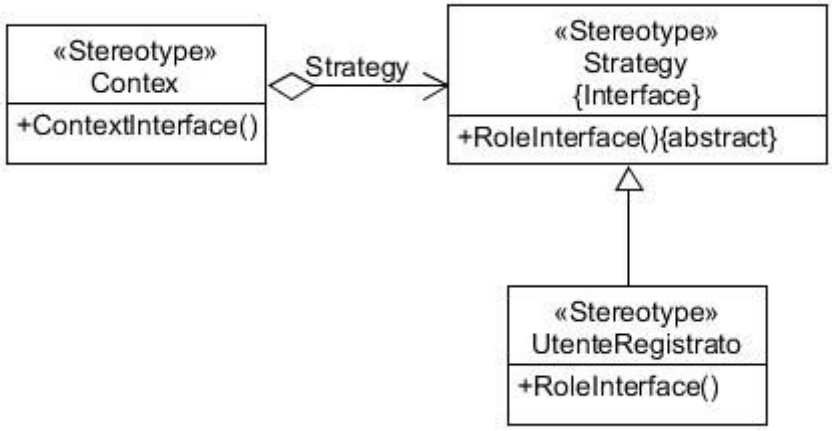


I partecipanti di questo pattern sono:

- ***Strategy***

- Dichiara l'interfaccia di riferimento per tutti gli algoritmi concreti.
- ***ConcreteStrategy***
- Implementa un particolare algoritmo utilizzando l'interfaccia definita da *Strategy*.
- ***Context***
- Carica un oggetto *ConcreteStrategy* e utilizza un riferimento a *Strategy* per eseguire l'algoritmo concreto. Definisce l'interfaccia per accedere ai membri dell'algoritmo caricato.

Nel nostro sistema questo pattern sarà utilizzato per riconoscere i vari ruoli degli utenti presenti nel sistema.



4.4. Data-Access-Object

Il *DAO* (*Data Access Object*) è un pattern architetturale per la gestione della persistenza. Si tratta di una classe usata principalmente in applicazioni web sia di tipo Java EE sia di tipo EJB, per stratificare e isolare l'accesso ad una tabella tramite query (poste all'interno dei metodi della classe) ovvero al *data layer* da parte della *business logic* creando un maggiore livello di astrazione ed una più facile manutenibilità. I metodi del DAO con le rispettive query dentro verranno così richiamati dalle classi della business logic.
Il vantaggio relativo all'uso del DAO è dunque il mantenimento di una rigida separazione tra le componenti di un'applicazione, le quali potrebbero essere il "Modello" e il "Controllo" in un'applicazione basata sul paradigma MVC.