FLAT

ODD Object Design Document

Data	Versione	Cambiamenti	Autori
18/dicembre/17	0.1 Draft	Strutturazione documento e scelta dei design trade-offs	Tutti
02/febbraio/18	1.0	Revisione finale e piccole modifiche	Tutti

Sommario

Sommario

1. Introduzione

1.1 Object Design Trade-offs

Ci apprestiamo a definire gli aspetti implementativi del nostro sistema, aspetti che nei documenti precedenti (e.g. "RAD", "SSD" etc.) sono stati tralasciati per definire in modo chiaro gli obiettivi del sistema. Con il seguente documento puntiamo a creare un modello capace di descrivere in modo preciso tutte le funzionalità individuate nei modelli precedenti.

Nel particolare, definiremo le interfacce delle classi, le operazioni, i tipi e gli argomenti e le "signature" dei sottosistemi definiti nel System Design.

Sono necessari però definire una serie di trade-off e di linee guida per poter lavorare in modo ordinato alla "codebase".

Comprensibilità vs Tempo:

Un codice comprensibile e commentato sarà sempre preferito rispetto ad un codice scritto rapidamente; questo perché è importante che il codice rimanga di facile fruizione per tutti i collaboratori del progetto, anche se potrà rallentarci più avanti nello sviluppo.

Prestazioni vs Costi:

Il progetto è portato avanti senza alcuna sovvenzione economica, quindi si farà affidamento a tecnologie "Open Source" e a strumenti free online. Per il database di film utilizzeremo la versione gratuita dell'API di The Internet Movie Database che, sebbene limitante per il numero di richieste fattibili (circa 4 al secondo, troppo poche per gestire un sito professionale) e velocità, ci offre tutte le funzioni di cui abbiamo bisogno per creare il sistema in questione.

Interfaccia vs Usabilità:

L'interfaccia fa dell'usabilità il suo punto di forza. Fin dal primo momento la comodità di utilizzo è stata messa in primo piano durante il design, scelta che si rispecchia fin dai primi mock-up.

Sicurezza vs Efficienza:

La sicurezza è la principale preoccupazione di ogni piattaforma online che si rispetti, ma il poco tempo a disposizione non ci permette di rendere "bullet proof" il sistema. Metteremo comunque il minimo di sicurezza possibile nel progetto, assicurandoci che il login/registrazione e tutte le altre interazioni con il database siano sicure e quanto più efficienti possibili.

1.2 Linee guida per la Documentazione delle Interfacce

Per creare un progetto chiaro, pulito e univocamente compreso, gli sviluppatori dovranno sottostare ad alcune linee guida per la stesura del codice:

Code style:

È buona norma rispettare gli standard di Java circa la formattazione del codice, nello specifico:

- Tutto il codice sorgente deve essere indentato con tabs
- Il codice deve essere formattato secondo lo stile vigente dell'IDE utilizzato (si consiglia caldamente Eclipse)

Esempio:

```
for(Object x: listaElementi){
	System.out.println(x.getNome());
}
```

Naming Convention:

È buona norma utilizzare nomi che siano:

- Descrittivi
- Pronunciabili
- Di uso comune
- Non abbreviati (se non per variabili momentanee)
- Utilizzando solo i caratteri consentiti (A-Z, a-z, 0-9)

Variabili:

È buona norma rispettare gli standard di Java circa la nomenclatura delle variabili, nello specifico:

- Devono iniziare per lettera minuscola e ogni parola seguente deve iniziare con una maiuscola.
  Esempio: nomeDiVariabile
- Nel caso di variabili costanti, il nome deve essere interamente in maiuscola ed eventuali spazi devono essere sostituiti da underscore ('_').
  Esempio: ORE_IN_UN_GIORNO

Metodi:

È buona norma rispettare gli standard di Java circa la nomenclatura dei metodi, nello specifico:

- Devono iniziare per lettera minuscola e ogni parola seguente deve iniziare con una maiuscola.
  Esempio: nomeDiMetodo()
- Eventuali metodi per l'accesso e la modifica delle variabili d'istanza di una classe devono essere del tipo getNomeVariabile() e setNomeVariabile()
  Esempio: getOrario(), setOrario()
- I commenti devono essere raggruppati in base alla loro funzionalità, devono essere situati prima della dichiarazione del metodo. È obbligatorio usare il sistema "JavaDocs" per poter avere una documentazione completa ed uniforme.

Classi:

È buona norma rispettare gli standard di Java circa la nomenclatura delle classi, nello specifico:

- Devono iniziare per lettera maiuscola e ogni parola seguente deve iniziare con una maiuscola.
  Esempio: NomeDellaClasse
- Devono avere un nome autodescrittivo.

1.3 Definizioni, Acronimi, Abbreviazioni e Riferimenti

Acronimi:

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

Abbreviazioni:

- DB: Database
- API: Application programming interface

Riferimenti:

- Libro di testo: Bruegge, A.H. Dutoit, Object Oriented Software Engineering.
- Slide fornite dal professore, reperibili su e-learning.
- The Movie Database (TMDB) per l'*api* offerta.

2. Packages

Il nostro sistema è suddiviso in tre livelli:

Interface layer:

Rappresenta il layer che l'utente visualizza e quello con il quale interagisce. Raccoglie i dati in input e li inoltra all'Application Logic Layer, in seguito mostra in output il risultato dell'azione svolta.

Application Logic layer:

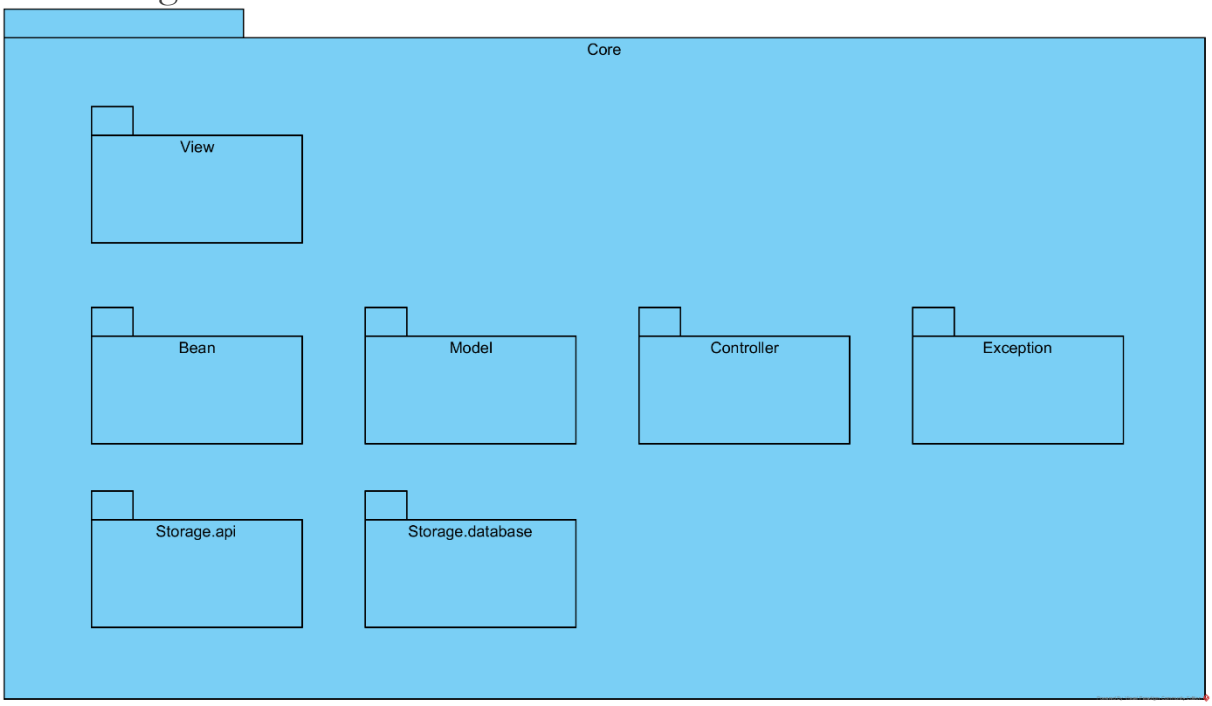
Elabora i dati che arrivano dal client e li restituisce all'Interface Layer, spesso utilizzando i dati persistenti gestiti dallo Storage layer. Svolge varie funzioni raggruppate nelle seguenti gestioni:

1. Gestione Autenticazione
2. Gestione Registrazione
3. Gestione Account
4. Gestione Recensioni
5. Gestione Watchlist
6. Gestione Amministrazione
7. Gestione Ricerca

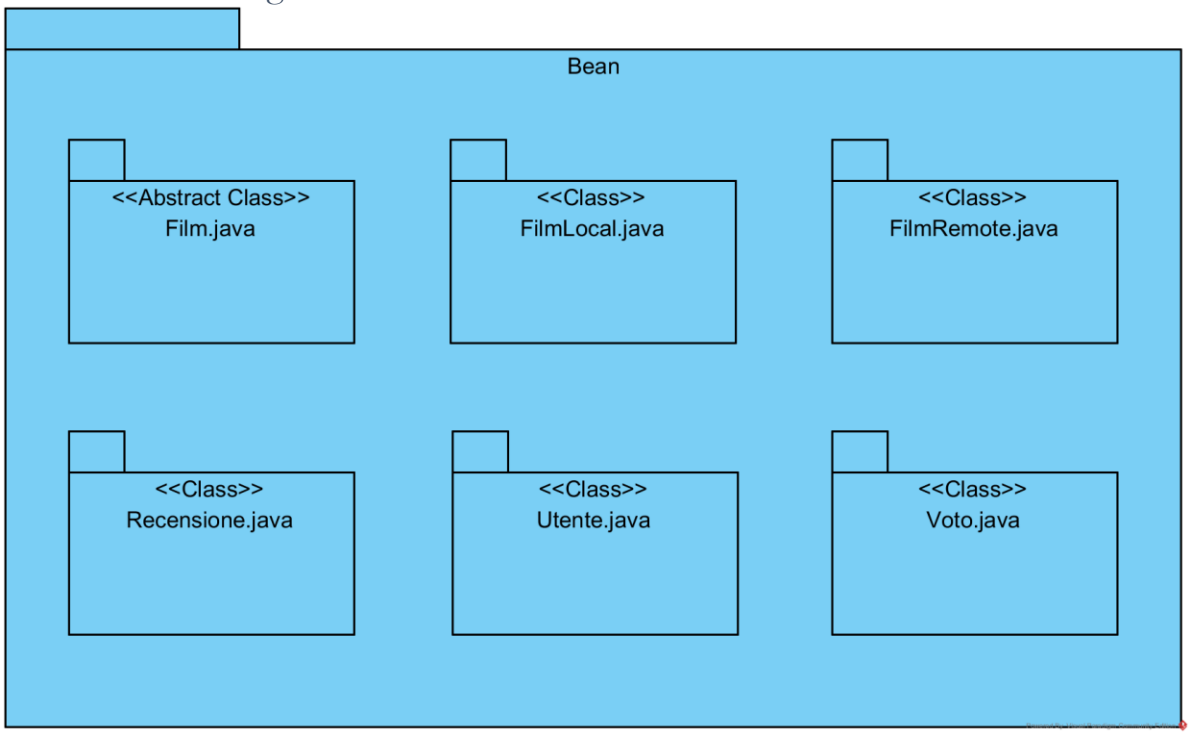
Storage layer:

Ha il compito di memorizzare i dati in maniera persistente, utilizzando un DBMS, riceve richieste dall'Application Logic layer e restituisci i dati del DBMS richiesti.

2.1 Package core

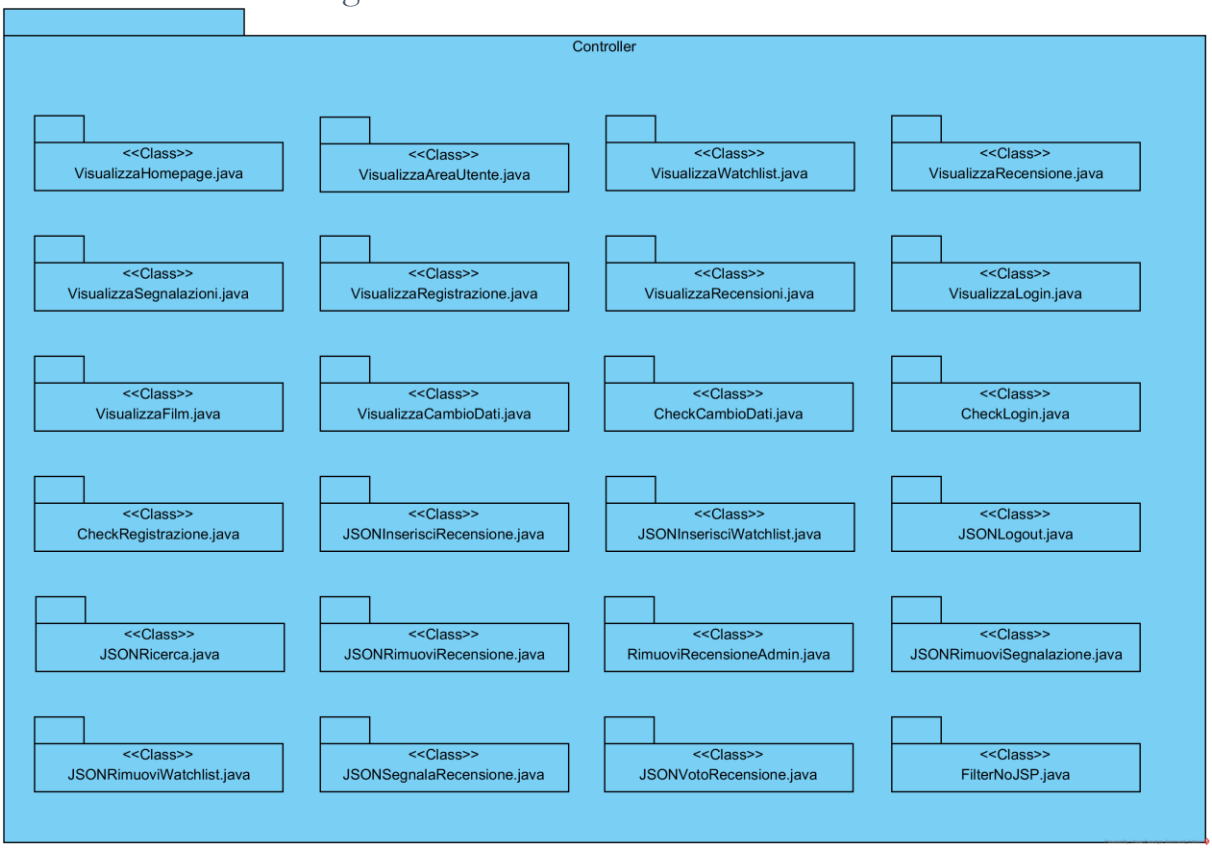


2.1.1 Bean Package



Classe:	Descrizione:
Film.java	Classe astratta che definisce la struttura minima dei film sul sito
FilmLocal.java	Classe che estende la classe astratta “Film.java”, definisce i film salvati nel database locale.
FilmRemote.java	Classe che estende la classe astratta “Film.java”, definisce i film salvati nel database remoto (API).
Recensione.java	Classe che definisce una recensione nel sistema
Utente.java	Classe che definisce un utente nel sistema
Voto.java	Classe che definisce un voto di una recensione nel sistema

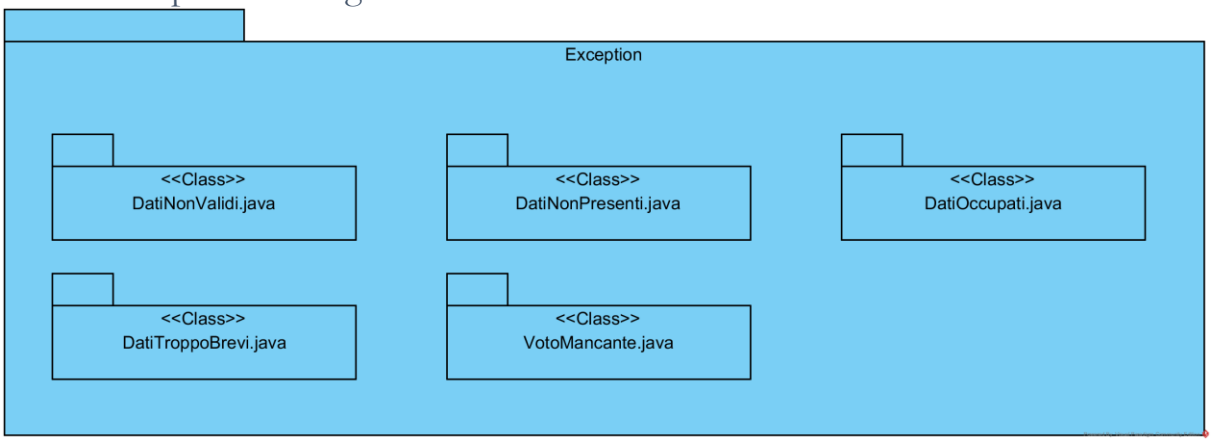
2.1.2 Controller Package



Classe:	Descrizione:
VisualizzaHomepage.java	Servlet che permette di visualizzare e mappare la homepage
VisualizzaAreaUtente.java	Servlet che permette di visualizzare e mappare l'area utente
VisualizzaWatchlist.java	Servlet che permette di visualizzare e mappare la pagina che mostra la propria watchlist
VisualizzaSegnalazioni.java	Servlet che permette di visualizzare e mappare l'area di controllo segnalazioni
VisualizzaRegistrazione.java	Servlet che permette di visualizzare e mappare la pagina di registrazione
VisualizzaRecensioni.java	Servlet che permette di visualizzare e mappare la pagina dedicata alla visualizzazione la lista di tutte le recensioni inserite sulla piattaforma
VisualizzaRecensione.java	Servlet che permette di visualizzare e mappare la pagina dedicata alla visualizzazione di una singola recensione
VisualizzaLogin.java	Servlet che permette di visualizzare e mappare la pagina di login
VisualizzaFilm.java	Servlet che permette di visualizzare e mappare la pagina dedicata alla visualizzazione di un singolo film
VisualizzaCambioDati.java	Servlet che permette di visualizzare e mappare l'area di cambio dati di utente
CheckCambioDati.java	Servlet che esegue il controllo dei dati immessi nella pagina di cambio dati

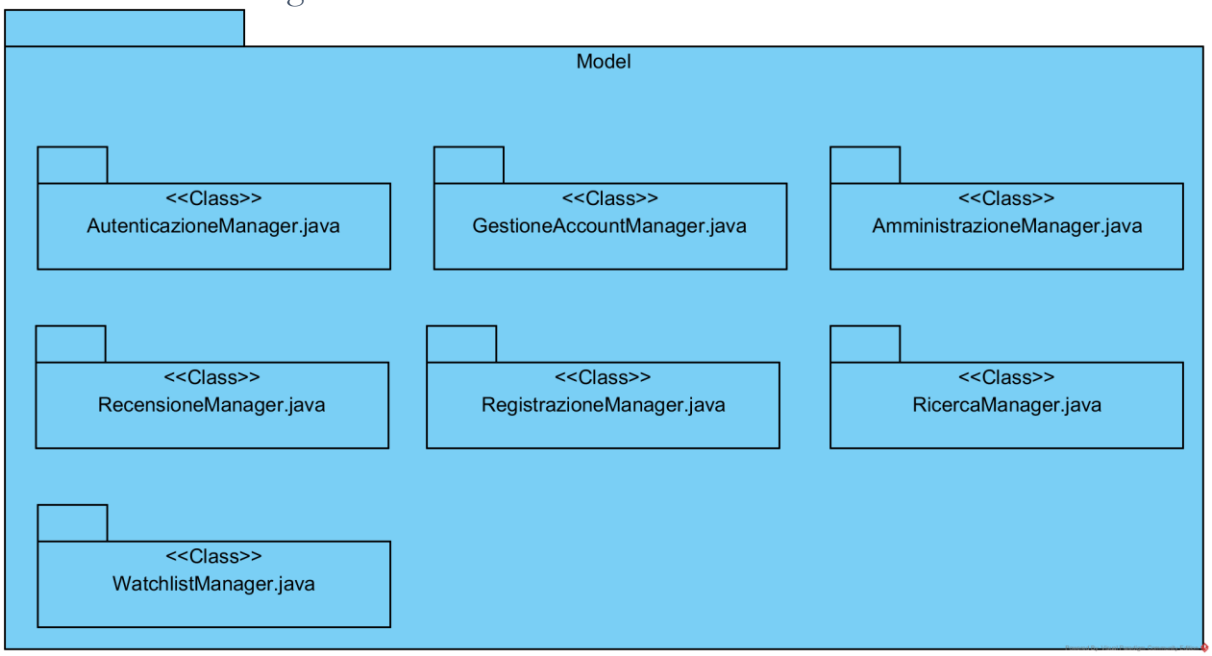
CheckRegistrazione.java	Servlet che esegue il controllo dei dati immessi nella pagina di registrazione
CheckLogin.java	Servlet che esegue il controllo dei dati immessi nella pagina di login
JSONInserisciRecensione.java	Servlet che permette di inserire in maniera asincrona una recensione nella piattaforma
JSONInserisiWatchlist.java	Servlet che permette di inserire in maniera asincrona un film nella watchlist personale
JSONLogout.java	Servlet che permette di effettuare in maniera asincrona il logout dalla piattaforma
JSONRicerca.java	Servlet che permette di ricercare un film nel database in maniera asincrona
JSONRimuoviRecensione.java	Servlet che permette ad un utente di rimuovere una recensione dal database in maniera asincrona
JSONRimuoviRecensioneAdmin.java	Servlet che permette ad un admin di rimuovere una recensione dal database in maniera asincrona
JSONRimuoviSegnalazione.java	Servlet che permette ad un admin di rimuovere una segnalazione dalla piattaforma in maniera asincrona
JSONRimuoviWatchlist.java	Servlet che permette ad un utente di rimuovere un film dalla propria watchlist in maniera asincrona
JSONSegnalaRecensione.java	Servlet che permette ad un utente di segnalare una recensione in maniera asincrona
JSONVotoRecensione.java	Servlet che permette ad un utente di votare una recensione in maniera asincrona
FilterNoJSP.java	Filtro che blocca l'accesso a pagine "*.jsp" (che sono esclusivamente di view)

2.1.3 Exception Package



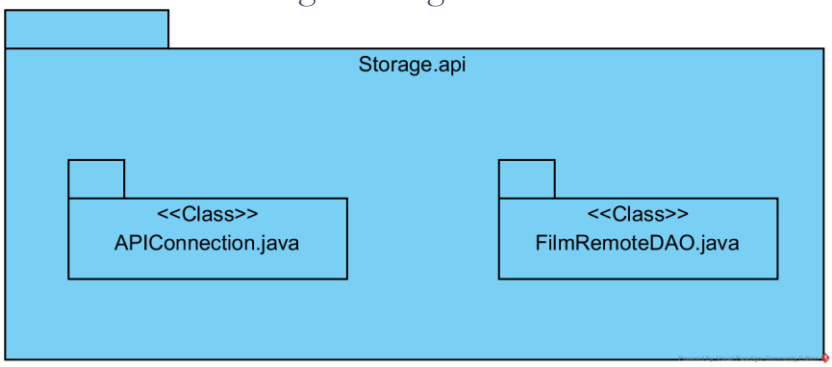
Classe:	Descrizione:
DatiNonValidi.java	Eccezione da lanciare quando i dati inseriti non sono validi
DatiNonPresenti.java	Eccezione da lanciare quando i dati inseriti non sono presenti nel database locale
DatiOccupati.java	Eccezione da lanciare quando i dati inseriti sono già utilizzati da un altro utente nella piattaforma
DatiTroppoBrevi.java	Eccezione da lanciare quando i dati inseriti sono troppo brevi
VotoMancante.java	Eccezione da lanciare quando non viene specificato un voto durante l'inserimento della recensione

2.1.4 Model Package



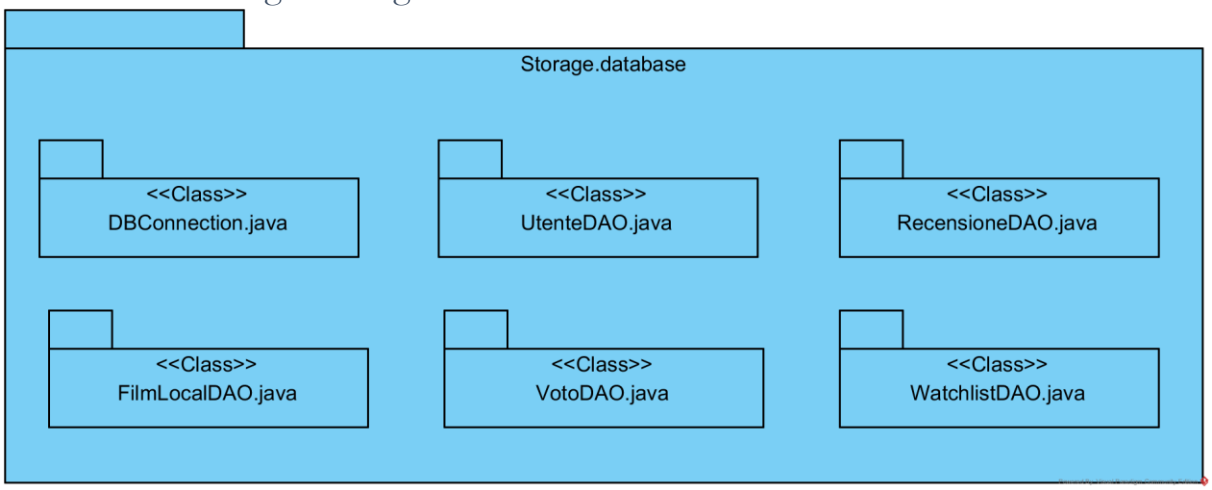
Classe:	Descrizione:
AutenticazioneManager.java	Gestisce tutte le funzionalità per l'autenticazione degli utenti
RegistrazioneManager.java	Gestisce tutte le funzionalità per la registrazione degli utenti
GestioneAccountManager.java	Gestisce tutte le funzionalità per la modifica e visualizzazione della pagina personale degli utenti
RecensioniManager.java	Gestisce tutte le funzionalità per gestire l'aggiunta, la selezione e la rimozione delle recensioni
RicercaManager.java	Gestisce la funzionalità di ricerca sull'API online
WatchlistManager.java	Gestisce la funzionalità di aggiunta e rimozione dalla watchlist
AmministrazioneManager.java	Gestisce tutte le funzionalità da amministratore

2.1.5 Remote Storage Package



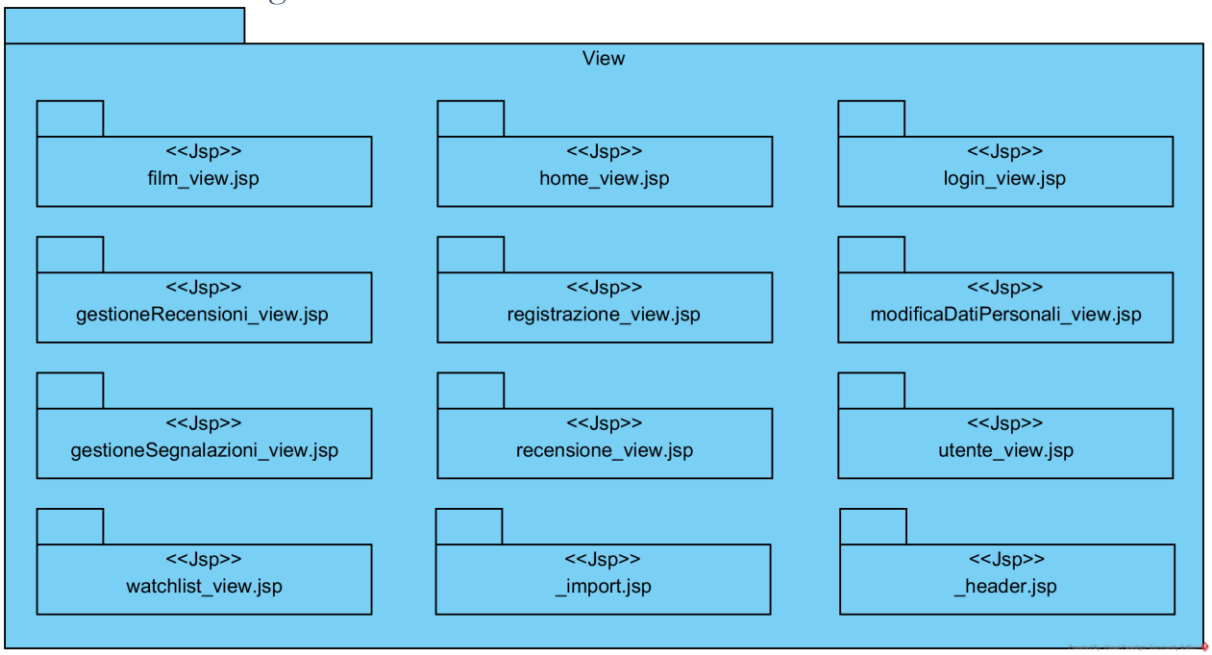
Classe:	Descrizione:
APIConnection.java	Si occupa di inviare le "Request" all'API online
FilmRemoteDAO.java	Gestisce le operazioni per ottenere un film dall'API online

2.1.6 Local Storage Package



Classe:	Descrizione:
DBConnection.java	Si occupa di gestire il pool delle connessioni al database
FilmLocalDAO.java	Gestisce tutte le funzionalità CRUD per i film locali
RecensioneDAO.java	Gestisce tutte le funzionalità CRUD per le recensioni
UtenteDAO.java	Gestisce tutte le funzionalità CRUD per gli utenti
VotoDAO.java	Gestisce tutte le funzionalità CRUD per i voti delle recensioni
WatchlistDAO.java	Gestisce tutte le funzionalità CRUD per la watchlist

2.1.7 View Package



Classe:	Descrizione:
home_view.jsp	Gestisce la visualizzazione della Homepage
film_view.jsp	Gestisce la visualizzazione delle pagine dei film
gestioneRecensioni_view.jsp	Gestisce la visualizzazione della pagina di moderazione “Gestione recensioni”
gestioneSegnalazioni_view.jsp	Gestisce la visualizzazione della pagina di moderazione “Gestione segnalazioni”
login_view.jsp	Gestisce la visualizzazione della pagina di login
registrazione_view.jsp	Gestisce la visualizzazione della pagina di registrazione
modificaDatiPersonali_view.jsp	Gestisce la visualizzazione della pagina per la modifica dei dati personali
recensione_view.jsp	Gestisce la visualizzazione delle pagine delle recensioni
utente_view.jsp	Gestisce la visualizzazione delle pagine degli utenti
watchlist_view.jsp	Gestisce la visualizzazione delle pagine delle watchlist personali degli utenti
_import.jsp	Serie di file da importare per ogni pagina, viene inclusa da ogni altra pagina
_header.jsp	Header delle pagine, viene inclusa da ogni altra pagina

3. Interfaccia Delle Classi

L'interfaccia delle classi, ogni metodo ed ogni classe è opportunamente descritto/a nella documentazione "javadoc" allegata.

4. Design Patterns

Per poter sviluppare un sistema coeso e facilmente modificabile è stato deciso di utilizzare diversi design patterns in diverse aree di gestione:

3.1 Data Access Object Pattern

DAO (Data Access Object) è un pattern per la gestione della persistenza: si tratta fondamentalmente di una classe con relativi metodi che rappresenta un'entità tabellare di un database, 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).
Nel nostro sistema abbiamo una serie di classi DAO che ci permettono di comunicare con il database (locale e non) senza doverci preoccupare di gestire le connessioni, le request http e i dati ricevuti da queste ultime.

3.2 Façade Pattern

Façade è un pattern per la gestione dell'accesso di sottosistemi: si tratta di una classe con i relativi metodi che possono essere usati dal livello superiore per interagire con il livello inferiore.
Nel nostro sistema abbiamo una serie di classi dette "Manager" che raccolgono metodi statici atti a comunicare con i livelli inferiori senza dover far controlli sui dati inseriti o ricevuti allo strato superiore.

3.3 Abstract Factory Pattern

Abstract Factory è uno dei principali pattern creazionali il cui scopo è quello di fornire un'interfaccia per creare famiglie di oggetti interconnessi fra loro facilitando, in questo modo, la creazione di un sistema indipendente dall'implementazione degli oggetti concreti.
Nel nostro sistema viene utilizzato una versione "ridotta" per poter gestire i due tipi di film (FilmLocal e FilmRemote) dato che ognuno di essi ha un differente tipo di utilizzo e diversi tipi di dati pur venendo gestiti in maniera simile.