|  |
| --- |
| A logo with numbers and letters  Description automatically generated  **Università degli Studi di Salerno Corso di Ingegneria del Software** |

**Rated  
Object Design Document  
Versione 1.1**

**A black and white logo

Description automatically generated**

Data: 10/01/2025

**Coordinatore del progetto:**

|  |  |
| --- | --- |
| Nome | Matricola |
|  |  |
|  |  |

**Partecipanti:**

|  |  |
| --- | --- |
| Nome | Matricola |
| Francesco Rao | 0512116836 |
| Bruno Nesticò | 0512117268 |
|  |  |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
| **Scritto da:** | Francesco Rao, Bruno Nesticò |

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Data | Versione | Descrizione | Autore |
| 16/12/2024 | 1.0 | Prima stesura completa | Francesco Rao, Bruno Nesticò |
| 10/01/2025 | 1.1 |  | Francesco Rao, Bruno Nesticò |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Indice**

1. Introduzione 4

1.1 Scopo del Sistema 4

1.2 Obiettivi di progettazione 4

1.3 Linee guida per la documentazione dell’interfaccia 4

1.4 Ottimizzazione del modello a oggetti 4

2. Packages 5

2.1 Struttura del progetto 6

3. Interfaccia delle classi 7

4. Design Pattern 12

**1. Introduzione**

L’**Object Design Document** ha l'obiettivo di approfondire gli aspetti tecnici legati all'implementazione del sistema **Rated**, ampliando quanto definito nei documenti precedenti, focalizzati sull’architettura e sulla progettazione generale. Questo documento fornisce una descrizione dettagliata delle scelte progettuali effettuate durante le fasi di analisi e design, includendo i principali trade-off, linee guida per la documentazione delle interfacce e l’identificazione dei **Design Pattern** utilizzati.

Saranno definiti i packages, le interfacce delle classi e i relativi diagrammi, descrivendo nel dettaglio operazioni, parametri e firme, in coerenza con i sottosistemi individuati nel **System Design Document** e con i requisiti funzionali e non funzionali indicati nel **Requirements Analysis Document**.

Il documento si concentra su tutti i requisiti funzionali indicati nel **RAD**, che saranno implementati nella prima versione del sistema garantendo il rispetto delle scadenze.

* 1. **Object design trade-offs**

**Sicurezza vs. Tempi di Sviluppo:** Tenendo conto delle scadenze e dei tempi ristretti di sviluppo, saranno implementati sistemi di sicurezza essenziali per garantire un livello adeguato di protezione. Questi includono l'autenticazione tramite email e password crittografata, la sanificazione dei campi di input dei form e una gestione degli accessi alle pagine basata sui ruoli definiti nel RAD (Guest, Recensore, Gestore del Catalogo, Moderatore). Le pagine di errore, tuttavia, saranno progettate in modo minimale e orientate esclusivamente alla funzionalità essenziale. Nella prima versione del sistema, i controlli sui form saranno limitati a verifiche di base per tutti gli utenti, inclusi quelli compilati dai Gestori del Catalogo, che si presuppone, in qualità di operatori della piattaforma, abbiano familiarità con le modalità di compilazione relative alle loro funzioni. Tuttavia, per i form critici, come quelli di LogIn e Register, saranno implementati controlli più rigorosi per garantire un adeguato livello di sicurezza. Questo approccio rappresenta un compromesso mirato a proteggere i dati sensibili e assicurare il corretto funzionamento della piattaforma, mantenendo al contempo la rapidità e la semplicità di implementazione necessarie per rispettare le scadenze.

**Prestazioni vs. Supportabilità:** Nel contesto del progetto, è importante considerare il possibile trade-off tra prestazioni e supportabilità. Sebbene il requisito di prestazioni, definito nel RAD, richieda tempi di caricamento inferiori a 2 secondi per ogni pagina o funzione principale, si ritiene prioritario privilegiare la supportabilità del codice. Questa scelta si fonda sull'ipotesi che l'applicativo web non presenti una complessità elevata. Di conseguenza, è ragionevole supporre che una struttura del codice orientata alla manutenibilità e agli aggiornamenti futuri non comprometta in modo significativo le prestazioni del sistema. In altre parole, adottare pratiche di sviluppo che favoriscano la supportabilità, come una progettazione modulare, codice leggibile e testabile, dovrebbe consentire di mantenere un tempo di risposta inferiore ai 2 secondi, soddisfacendo così entrambi i requisiti. Tale approccio garantirebbe un equilibrio tra efficienza immediata e sostenibilità a lungo termine del sistema, riducendo il rischio di complicazioni durante l'evoluzione dell'applicativo.

* 1. **Interface documentation guidelines**

**Nomi dei Package**

* I nomi dei package devono essere scritti in minuscolo, senza spazi o caratteri speciali.
* Per nomi composti da più parole, è necessario utilizzare il formato snake\_case.

**Nomi delle Classi**

* Le classi devono seguire il formato PascalCase, iniziando con una lettera maiuscola.
* I nomi devono essere descrittivi, rappresentando chiaramente l’entità o la funzionalità implementata.

**Classi DAO**

* Le classi DAO devono seguire il formato PascalCase e terminare con il suffisso DAO per indicarne il ruolo di accesso ai dati.

**Classi che Forniscono Servizi**

* Queste classi devono rispettare il formato PascalCase.

**Nomi delle Servlet**

* Le Servlet devono seguire il formato PascalCase e terminare con il suffisso Servlet.

**Nomi dei Metodi**

* I metodi devono avere nomi descrittivi, che riflettano chiaramente l'operazione eseguita.
* Devono seguire il formato camelCase.

**Nomi delle Variabili**

* I nomi delle variabili devono essere descrittivi.
* È possibile utilizzare sia il formato camelCase che il formato snake\_case, in base al contesto.

**Nomi dei File JSP**

* I file JSP devono seguire il formato camelCase, riflettendo chiaramente il contenuto della pagina.

**Nomi delle Classi che Implementano il Pattern Strategy**

* Queste classi devono seguire il formato PascalCase e terminare con la parola Validator, per identificare chiaramente il loro ruolo.

**Organizzazione delle Risorse Statiche**

* Fogli di stile, script e immagini devono essere organizzati nella directory webapp/static, suddivisa in sottocartelle per ogni tipo di file.
  1. **Design Pattern**

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

**Connection Pool Management Pattern**

L'applicazione **Rated** richiede un accesso ottimizzato e centralizzato alle connessioni al database, data la natura concorrente delle operazioni effettuate dagli utenti. L’implementazione di un **Connection Pool Management** si rivela essenziale per migliorare l’efficienza nella gestione delle connessioni al database MySQL, garantendo una condivisione efficace delle risorse.

L’utilizzo di questa soluzione consente di mantenere un pool di connessioni già aperte e riutilizzabili, evitando l’overhead causato dalla creazione e dalla chiusura continua di nuove connessioni. Ciò garantisce:

* La gestione di un numero limitato di connessioni attive, prevenendo così un utilizzo inefficiente delle risorse del database.
* Una maggiore scalabilità del sistema, grazie alla possibilità di servire più richieste concorrenti.
* Un ciclo di vita chiaro e centralizzato per tutte le connessioni.

Nel progetto **Rated**, il **DriverConnectionPool** è responsabile della gestione di queste connessioni. Attraverso un’allocazione intelligente e il rilascio delle connessioni utilizzate, il pool di connessioni garantisce la continuità operativa dell'applicazione senza sovraccaricare il DBMS.

L’accesso alle connessioni nel pool avviene tramite metodi che permettono di acquisire una connessione disponibile e di restituirla una volta terminato l’utilizzo. In questo modo si ottimizza l’utilizzo delle risorse e si previene il verificarsi di problemi di saturazione delle connessioni.

**Strategy Pattern**

Nel progetto, per la gestione della validazione dei campi di input, è stato adottato il Strategy Pattern. Questo design pattern consente di definire una famiglia di algoritmi di validazione (ad esempio, validazione di email, numeri, date, ecc.), incapsularli in metodi specifici all'interno di una classe e renderli intercambiabili. La classe di validazione creata funge da contenitore per tutte le funzioni di validazione necessarie, ognuna delle quali rappresenta una strategia separata per verificare uno specifico tipo di input. Questo approccio garantisce:

* Modularità: Ogni algoritmo di validazione è implementato come un metodo distinto, facilitando la leggibilità e la manutenzione del codice.
* Flessibilità: È possibile aggiungere facilmente nuove funzioni di validazione o aggiornare quelle esistenti senza modificare il codice già scritto.
* Riutilizzabilità: Le funzioni di validazione possono essere richiamate in modo indipendente o combinate, a seconda delle necessità del sistema. Chiarezza: Separando la logica di validazione dalla logica principale dell'applicazione, si ottiene un design più chiaro e mantenibile.

**1.4. Definizioni, acronimi e abbreviazioni**

**RAD**: Requirements Analysis Document.

**SDD**: System Design Document.

**ODD**: Object Design Document.

**OCL**: Object Constraints Language

**UG**: Utente Guest

**UR**: Utente Recensore

**GC**: Gestore del Catalogo

**MOD**: Moderatore

**2. Packages**

**2.1. Struttura del progetto**

Di seguito viene indicata la struttura organizzativa di file e cartelle che compongono la parte implementativa del sistema.

**\*main**

**❖ java:**Contiene i file sorgente Java organizzati in pacchetti.

**➢ sottosistemi:**

**●Gestione\_Catalogo**

**◆view  
 ◆service  
●Gestione\_Profilo**

**◆view  
 ◆service**

**●Gestione\_Recensioni  
 ◆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**

**● scripts**

**● css**

**\*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.

Immagine che contiene diagramma, Rettangolo, testo, Piano

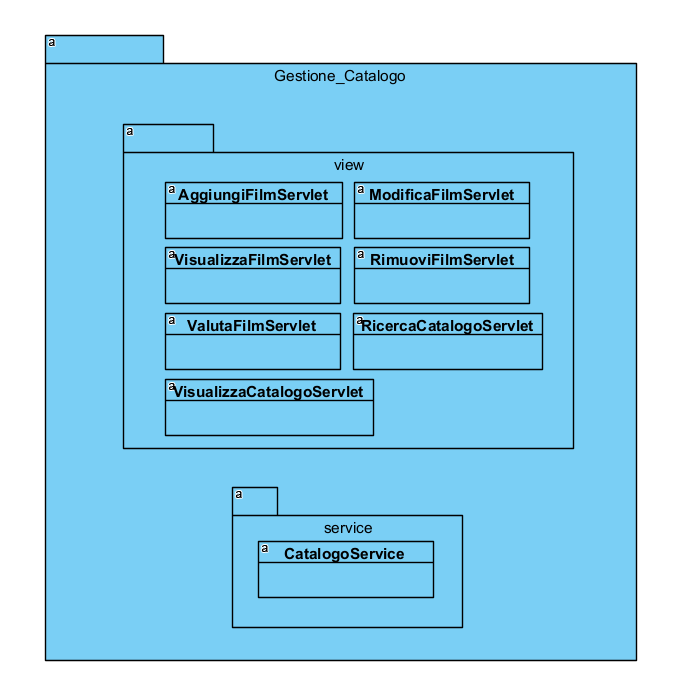
Descrizione generata automaticamente

**Immagine che contiene testo, schermata, diagramma, Parallelo

Descrizione generata automaticamente**

**Immagine che contiene testo, schermata, Rettangolo, diagramma

Descrizione generata automaticamente**

****

**Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente**

**3. Interfaccia delle classi**

**Package Gestione Utenti**

|  |  |
| --- | --- |
| **Interfaccia** | **AutenticationService** |
| Descrizione | AutenticationService fornisce il servizio relativo all’autenticazione e all’aggiornamento degli account |
| Metodi | +login(email: String, password: String) : UtenteBean  +logout(session: HttpSession) : void  +register(username: String, email: String, password: String, biografia: String, icon: byte[]) : UtenteBean |
| Invariante classe | // |

|  |  |
| --- | --- |
| Nome Metodo | +login(email: String, password: String) : UtenteBean |
| Descrizione | Questo metodo permette a un utente di autenticarsi. |
| Pre-condizione | context:  AutenticationService::login(email: String, password: String)  pre:  email <> null AND email.trim() <> "" AND  password <> null AND password.trim() <> "" |
| Post-condizione | context:  AutenticationService::login(email: String, password: String)  post:  (result = null AND UtenteDAO.findByEmail(email) = null) OR  (result <> null AND PasswordUtility.hashPassword(password).equals(result.getPassword())) |

|  |  |
| --- | --- |
| Nome Metodo | +logout(session: HttpSession) : void |
| Descrizione | Questo metodo permette di effettuare il logout invalidando la sessione dell'utente. |
| Pre-condizione | context:  AutenticationService::logout(session: HttpSession)  pre:  session <> null |
| Post-condizione | context:  AutenticationService::logout(session: HttpSession)  post:  session.isValid() = false |

|  |  |
| --- | --- |
| Nome Metodo | +register(username: String, email: String, password: String, biografia: String, icon: byte[]) : UtenteBean |
| Descrizione | Questo metodo permette di registrare un nuovo utente. |
| Pre-condizione | context:  AutenticationService::register(username: String, email: String, password: String, biografia: String, icon: byte[])  pre:  username <> null AND username.trim() <> "" AND  email <> null AND email.trim() <> "" AND  password <> null AND password.trim() <> "" AND  UtenteDAO.findByEmail(email) = null AND  UtenteDAO.findByUsername(username) = null |
| Post-condizione | context:  AutenticationService::register(username: String, email: String, password: String, biografia: String, icon: byte[])  post:  (result <> null AND UtenteDAO.contains(result)) OR (result = null AND (UtenteDAO.findByEmail(email) <> null OR UtenteDAO.findByUsername(username) <> null)) |

|  |  |
| --- | --- |
| **Interfaccia** | **ModerationService** |
| Descrizione | ModerationService fornisce il servizio relativo moderazione tramite warn degli account |
| Metodi | +warn(email: String) : void |
| Invariante classe | // |

|  |  |
| --- | --- |
| Nome Metodo | +warn(email: String) : void |
| Descrizione | Questo metodo permette di avvisare un utente incrementando il numero di avvertimenti associati al suo account. |
| Pre-condizione | context:  ModerationService::warn(email: String)  pre:  email <> null AND email.trim() <> "" AND UtenteDAO.findByEmail(email) <> null |
| Post-condizione | context:  ModerationService::warn(email: String)  post:  UtenteDAO.findByEmail(email).getNWarning() = old(UtenteDAO.findByEmail(email).getNWarning()) + 1 |

|  |  |
| --- | --- |
| **Interfaccia** | **ProfileService** |
| Descrizione | ProfileServicefornisce il servizio relativo alla modifica e alla ricerca degli account. |
| Metodi | +ProfileUpdate(username: String, email: String, password: String, biografia: String, icon: byte[]) : UtenteBean  +PasswordUpdate(email: String, password: String) : UtenteBean  +findByUsername(username: String) : UtenteBean  +getUsers(recensioni: List<RecensioneBean>) : HashMap<String, String> |
| Invariante classe | // |

|  |  |
| --- | --- |
| Nome Metodo | +ProfileUpdate(username: String, email: String, password: String, biografia: String, icon: byte[]) : UtenteBean |
| Descrizione | Questo metodo permette di aggiornare il profilo di un utente. |
| Pre-condizione | context:  ProfileService::ProfileUpdate(username: String, email: String, password: String, biografia: String, icon: byte[])  pre:  username <> null AND username.trim() <> "" AND  email <> null AND email.trim() <> "" AND  password <> null AND password.trim() <> "" AND  UtenteDAO.findByEmail(email) <> null AND  UtenteDAO.findByUsername(username) = null |
| Post-condizione | context:  ProfileService::ProfileUpdate(username: String, email: String, password: String, biografia: String, icon: byte[])  post:  (result <> null AND result.getUsername() = username AND result.getPassword() = password AND result.getBiografia() = biografia AND result.getIcona() = icon) |

|  |  |
| --- | --- |
| Nome Metodo | +PasswordUpdate(email: String, password: String) : UtenteBean |
| Descrizione | Questo metodo permette di aggiornare la password di un utente. |
| Pre-condizione | context:  ProfileService::PasswordUpdate(email: String, password: String)  pre:  email <> null AND email.trim() <> "" AND  password <> null AND password.trim() <> "" AND  UtenteDAO.findByEmail(email) <> null |
| Post-condizione | context:  ProfileService::PasswordUpdate(email: String, password: String)  post:  (result <> null AND result.getPassword() = password) |

|  |  |
| --- | --- |
| Nome Metodo | +findByUsername(username: String) : UtenteBean |
| Descrizione | Questo metodo permette di trovare un utente tramite il suo username. |
| Pre-condizione | context:  ProfileService::findByUsername(username: String)  pre:  username <> null AND username.trim() <> "" |
| Post-condizione | context:  ProfileService::findByUsername(username: String)  post:  (result <> null AND result.getUsername() = username) OR (result = null AND UtenteDAO.findByUsername(username) = null) |

|  |  |
| --- | --- |
| Nome Metodo | +getUsers(recensioni: List<RecensioneBean>) : HashMap<String, String> |
| Descrizione | Questo metodo restituisce una mappa di utenti con la loro email e username associati alle recensioni. |
| Pre-condizione | context:  ProfileService::getUsers(recensioni: List<RecensioneBean>)  pre:  recensioni <> null |
| Post-condizione | context:  ProfileService::getUsers(recensioni: List<RecensioneBean>)  post:  result <> null AND result.size() = recensioni.size() |

**Package Gestione Catalogo**

|  |  |
| --- | --- |
| **Interfaccia** | **GestioneCatalogoService** |
| Descrizione | Gestione Catalogo fornisce il servizio relativo alla aggiunta, modifica e rimozione dei film |
| Metodi | +ProfileUpdate(username: String, email: String, password: String, biografia: String, icon: byte[]) : UtenteBean  +PasswordUpdate(email: String, password: String) : UtenteBean  +findByUsername(username: String) : UtenteBean  +getUsers(recensioni: List<RecensioneBean>) : HashMap<String, String> |
| Invariante classe | // |

|  |  |
| --- | --- |
| Nome Metodo | +getFilms() : List<FilmBean> |
| Descrizione | Questo metodo restituisce la lista di tutti i film presenti nel catalogo. |
| Pre-condizione | // |
| Post-condizione | // |

|  |  |
| --- | --- |
| Nome Metodo | +aggiungiFilm(nome: String, anno: int, durata: int, generi: String, regista: String, attori: String, locandina: byte[], trama: String) : void |
| Descrizione | Questo metodo permette di aggiungere un nuovo film al catalogo. |
| Pre-condizione | context:  CatalogoService::aggiungiFilm(nome: String, anno: int, durata: int, generi: String, regista: String, attori: String, locandina: byte[], trama: String)  pre:  nome <> null AND nome.trim() <> "" AND  anno > 0 AND  durata > 0 AND  generi <> null AND generi.trim() <> "" AND  regista <> null AND regista.trim() <> "" AND  attori <> null AND attori.trim() <> "" AND  locandina <> null AND locandina.length > 0 AND  trama <> null AND trama.trim() <> "" |
| Post-condizione | context:  CatalogoService::aggiungiFilm(nome: String, anno: int, durata: int, generi: String, regista: String, attori: String, locandina: byte[], trama: String)  post:  FilmDAO.contains(film) |

|  |  |
| --- | --- |
| Nome Metodo | +rimuoviFilm(film: FilmBean) : void |
| Descrizione | Questo metodo permette di rimuovere un film dal catalogo. |
| Pre-condizione | context:  CatalogoService::rimuoviFilm(film: FilmBean)  pre:  film <> null AND FilmDAO.contains(film) |
| Post-condizione | context:  CatalogoService::rimuoviFilm(film: FilmBean)  post:  !FilmDAO.contains(film) |

|  |  |
| --- | --- |
| Nome Metodo | +ricercaFilm(name: String) : List<FilmBean> |
| Descrizione | Questo metodo permette di cercare film per nome nel catalogo. |
| Pre-condizione | context:  CatalogoService::ricercaFilm(name: String)  pre:  name <> null AND name.trim() <> "" |
| Post-condizione | context:  CatalogoService::ricercaFilm(name: String)  post:  result <> null |

|  |  |
| --- | --- |
| Nome Metodo | +getFilm(idFilm: int) : FilmBean |
| Descrizione | Questo metodo restituisce un film dato il suo ID. |
| Pre-condizione | context:  CatalogoService::getFilm(idFilm: int)  pre:  idFilm > 0 |
| Post-condizione | context:  CatalogoService::getFilm(idFilm: int)  post:  result <> null |

|  |  |
| --- | --- |
| Nome Metodo | +getFilms(recensioni: List<RecensioneBean>) : HashMap<Integer, FilmBean> |
| Descrizione | Questo metodo restituisce una mappa di film associati alle recensioni. |
| Pre-condizione | context:  CatalogoService::getFilms(recensioni: List<RecensioneBean>)  pre:  recensioni <> null |
| Post-condizione | context:  CatalogoService::getFilms(recensioni: List<RecensioneBean>)  post:  result <> null |

|  |  |
| --- | --- |
| Nome Metodo | +addFilm(anno: int, attori: String, durata: int, generi: String, locandina: byte[], nome: String, regista: String, trama: String) : void |
| Descrizione | Questo metodo aggiunge un nuovo film al catalogo. |
| Pre-condizione | context:  CatalogoService::addFilm(anno: int, attori: String, durata: int, generi: String, locandina: byte[], nome: String, regista: String, trama: String)  pre:  anno > 0 AND  durata > 0 AND  attori <> null AND attori.trim() <> "" AND  generi <> null AND generi.trim() <> "" AND  locandina <> null AND locandina.length > 0 AND  nome <> null AND nome.trim() <> "" AND  regista <> null AND regista.trim() <> "" AND  trama <> null AND trama.trim() <> "" |
| Post-condizione | context:  CatalogoService::addFilm(anno: int, attori: String, durata: int, generi: String, locandina: byte[], nome: String, regista: String, trama: String)  post:  FilmDAO.contains(film) |

|  |  |
| --- | --- |
| Nome Metodo | +modifyFilm(idFilm: int, anno: int, attori: String, durata: int, generi: String, locandina: byte[], nome: String, regista: String, trama: String) : void |
| Descrizione | Questo metodo modifica i dettagli di un film esistente. |
| Pre-condizione | context:  CatalogoService::modifyFilm(idFilm: int, anno: int, attori: String, durata: int, generi: String, locandina: byte[], nome: String, regista: String, trama: String)  pre:  idFilm > 0 AND  anno > 0 AND  durata > 0 AND  attori <> null AND attori.trim() <> "" AND  generi <> null AND generi.trim() <> "" AND  locandina <> null AND locandina.length > 0 AND  nome <> null AND nome.trim() <> "" AND  regista <> null AND regista.trim() <> "" AND  trama <> null AND trama.trim() <> "" |
| Post-condizione | context:  CatalogoService::modifyFilm(idFilm: int, anno: int, attori: String, durata: int, generi: String, locandina: byte[], nome: String, regista: String, trama: String)  post:  FilmDAO.contains(film) |

|  |  |
| --- | --- |
| Nome Metodo | +removeFilm(idFilm: int) : void |
| Descrizione | Questo metodo rimuove un film dal catalogo dato il suo ID. |
| Pre-condizione | context:  CatalogoService::removeFilm(idFilm: int)  pre:  idFilm > 0 AND FilmDAO.contains(idFilm) |
| Post-condizione | context:  CatalogoService::removeFilm(idFilm: int)  post:  !FilmDAO.contains(idFilm) |

**Package Gestione Recensioni**

|  |  |
| --- | --- |
| Interfaccia | RecensioniService |
| Descrizione | Gestione Recensioni fornisce il servizio relativo alla aggiunta, modifica e rimozione delle recensioni e della vautazione/report delle stesse |
| Metodi | +addValutazione(email: String, idFilm: int, email\_recensore: String, nuovaValutazione: boolean) : void  +addRecensione(email: String, idFilm: int, recensione: String, Titolo: String, valutazione: int) : void  +FindRecensioni(email: String) : List<RecensioneBean>  +deleteRecensione(email: String, ID\_Film: int) : void  +deleteReports(email: String, ID\_Film: int) : void  +GetRecensioni(ID\_film: int) : List<RecensioneBean>  +GetValutazioni(ID\_film: int, email: String) : HashMap<String, ValutazioneBean>  +GetAllRecensioniSegnalate() : List<RecensioneBean>  +report(email: String, emailRecensore: String, idFilm: int) : void |
| Invariante classe | // |

|  |  |
| --- | --- |
| Nome Metodo | +addValutazione(email: String, idFilm: int, email\_recensore: String, nuovaValutazione: boolean) : void |
| Descrizione | Questo metodo aggiunge o modifica una valutazione su una recensione di un film. |
| Pre-condizione | context:  RecensioniService::addValutazione(email: String, idFilm: int, email\_recensore: String, nuovaValutazione: boolean)  pre:  email <> null AND email.trim() <> "" AND  idFilm > 0 AND  email\_recensore <> null AND email\_recensore.trim() <> "" |
| Post-condizione | context:  RecensioniService::addValutazione(email: String, idFilm: int, email\_recensore: String, nuovaValutazione: boolean)  post:  (result = null OR RecensioneDAO.findById(email\_recensore, idFilm).getNLike() >= 0 AND RecensioneDAO.findById(email\_recensore, idFilm).getNDislike() >= 0) |

|  |  |
| --- | --- |
| Nome Metodo | +addRecensione(email: String, idFilm: int, recensione: String, Titolo: String, valutazione: int) : void |
| Descrizione | Questo metodo permette di aggiungere una nuova recensione per un film. |
| Pre-condizione | context:  RecensioniService::addRecensione(email: String, idFilm: int, recensione: String, Titolo: String, valutazione: int)  pre:  email <> null AND email.trim() <> "" AND  idFilm > 0 AND  recensione <> null AND recensione.trim() <> "" AND  Titolo <> null AND Titolo.trim() <> "" AND  valutazione >= 0 AND valutazione <= 5 |
| Post-condizione | context:  RecensioniService::addRecensione(email: String, idFilm: int, recensione: String, Titolo: String, valutazione: int)  post:  RecensioneDAO.findById(email, idFilm) <> null |

|  |  |
| --- | --- |
| Nome Metodo | +FindRecensioni(email: String) : List<RecensioneBean> |
| Descrizione | Questo metodo restituisce tutte le recensioni di un utente. |
| Pre-condizione | context:  RecensioniService::FindRecensioni(email: String)  pre:  email <> null AND email.trim() <> "" |
| Post-condizione | context:  RecensioniService::FindRecensioni(email: String)  post:  result <> null |

|  |  |
| --- | --- |
| Nome Metodo | +deleteRecensione(email: String, ID\_Film: int) : void |
| Descrizione | Questo metodo permette di eliminare una recensione di un film. |
| Pre-condizione | Context context:  RecensioniService::deleteRecensione(email: String, ID\_Film: int)  pre:  email <> null AND email.trim() <> "" AND  ID\_Film > 0 AND  RecensioneDAO.findById(email, ID\_Film) <> null |
| Post-condizione | context:  RecensioniService::deleteRecensione(email: String, ID\_Film: int)  post:  RecensioneDAO.findById(email, ID\_Film) = null |

|  |  |
| --- | --- |
| Nome Metodo | +deleteReports(email: String, ID\_Film: int) : void |
| Descrizione | Questo metodo rimuove tutti i report associati a una recensione. |
| Pre-condizione | context:  RecensioniService::deleteReports(email: String, ID\_Film: int)  pre:  email <> null AND email.trim() <> "" AND  ID\_Film > 0 |
| Post-condizione | context:  RecensioniService::deleteReports(email: String, ID\_Film: int)  post:  RecensioneDAO.findById(email, ID\_Film).getNReports() = 0 |

|  |  |
| --- | --- |
| Nome Metodo | +GetRecensioni(ID\_film: int) : List<RecensioneBean> |
| Descrizione | Questo metodo restituisce tutte le recensioni associate a un film. |
| Pre-condizione | context:  RecensioniService::GetRecensioni(ID\_film: int)  pre:  ID\_film > 0 |
| Post-condizione | context:  RecensioniService::GetRecensioni(ID\_film: int)  post:  result <> null |

|  |  |
| --- | --- |
| Nome Metodo | +GetValutazioni(ID\_film: int, email: String) : HashMap<String, ValutazioneBean> |
| Descrizione | Questo metodo restituisce tutte le valutazioni associate a un film e un utente specifico. |
| Pre-condizione | context:  RecensioniService::GetValutazioni(ID\_film: int, email: String)  pre:  ID\_film > 0 AND  email <> null AND email.trim() <> "" |
| Post-condizione | context:  RecensioniService::GetValutazioni(ID\_film: int, email: String)  post:  result <> null |

|  |  |
| --- | --- |
| Nome Metodo | +GetAllRecensioniSegnalate() : List<RecensioneBean> |
| Descrizione | Questo metodo restituisce tutte le recensioni che sono state segnalate. |
| Pre-condizione | context:  RecensioniService::GetAllRecensioniSegnalate()  pre:  true |
| Post-condizione | context:  RecensioniService::GetAllRecensioniSegnalate()  post:  result <> null AND result.size() >= 0 |

|  |  |
| --- | --- |
| Nome Metodo | +report(email: String, emailRecensore: String, idFilm: int) : void |
| Descrizione | Questo metodo permette di segnalare una recensione. |
| Pre-condizione | context:  RecensioniService::report(email: String, emailRecensore: String, idFilm: int)  pre:  email <> null AND email.trim() <> "" AND  emailRecensore <> null AND emailRecensore.trim() <> "" AND  idFilm > 0 |
| Post-condizione | context:  RecensioniService::report(email: String, emailRecensore: String, idFilm: int)  post:  ReportDAO.findById(email, emailRecensore, idFilm) <> null |

**4. Design Pattern**

**Design Pattern: Facade**

Il Facade Pattern è un design pattern strutturale utilizzato per semplificare l'interazione con sistemi complessi composti da più sottosistemi. Questo approccio fornisce un'interfaccia chiara e accessibile, nascondendo i dettagli di implementazione e offrendo agli utenti una visione unificata e di alto livello.

L'applicazione di questo pattern è particolarmente utile quando è necessario semplificare un sistema complesso, rendendolo più intuitivo e facile da utilizzare. Inoltre, il Facade favorisce un elevato livello di disaccoppiamento, migliorando la manutenibilità e l'estensibilità del sistema: eventuali modifiche possono essere apportate intervenendo esclusivamente sui metodi esposti dall’interfaccia, senza impattare direttamente i singoli sottosistemi.

In questo contesto specifico, ogni sottosistema dispone della propria interfaccia dedicata, ad esempio:

* GestioneUtentiService
* GestioneCatalogoService
* GestioneRecensioniService

**Data Access Object (DAO)**

Il Data Access Object (DAO) è un design pattern che astrae e centralizza l'accesso ai dati, fornendo un'interfaccia uniforme per l'interazione con il database. Grazie a questa astrazione, il codice applicativo può manipolare i dati senza doversi preoccupare dei dettagli specifici del database o delle query SQL.

Il DAO agisce come un ponte tra l'applicazione e il sistema di persistenza, garantendo un codice più pulito, organizzato e facilmente manutenibile. Questo pattern facilita inoltre l'integrazione di eventuali cambiamenti nella logica di accesso ai dati, riducendo al minimo l'impatto sulle altre componenti del sistema.