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

**Rated  
Object Design Document  
Versione 1.0**

**A black and white logo

Description automatically generated**

Data: 24/11/2024

**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 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**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**

**1.1. Scopo del sistema**

Lo scopo del sistema “Rated” è offrire una piattaforma web dedicata agli appassionati di cinema, offrendo loro un ambiente per condividere recensioni, valutare i contributi della community e interagire con altri utenti. Le operazioni principali saranno: Pubblicare recensioni sui film, valutare i contenuti pubblicati da altri membri e godere di un sistema reputazionale che premia i recensori più attivi e apprezzati, favorendo la visibilità dei contenuti di qualità.

La gestione della piattaforma sarà affidata a figure dedicate, come i Gestori del catalogo, responsabili dell’aggiornamento continuo dell’offerta di film, e i Moderatori, incaricati di garantire un ambiente rispettoso e contenuti conformi alle linee guida.

**1.2. Obiettivi di progettazione**

**Usabilità**

*Validazione degli input*: saranno implementati meccanismi per prevenire errori durante l’inserimento di dati. Messaggi di errore chiari guideranno l’utente nella correzione dei valori errati.

*Design responsive*: l’interfaccia sarà ottimizzata per adattarsi a diversi dispositivi (PC, tablet, smartphone), rendendo l’esperienza uniforme e accessibile.

*Navigazione intuitiva*: ogni pagina presenterà una barra di navigazione per facilitare l’accesso rapido alle diverse sezioni.

**Riusabilità**

Il sistema garantirà la riusabilità, tramite l’ereditarietà fornita dal paradigma di programmazine object oriented e tramite l’utilizzo di diversi design pattern.

**Affidabilità**

*Controllo avanzato degli input****:*** oltre alla validazione primaria, saranno effettuati ulteriori controlli per gestire scenari non previsti e prevenire errori critici.

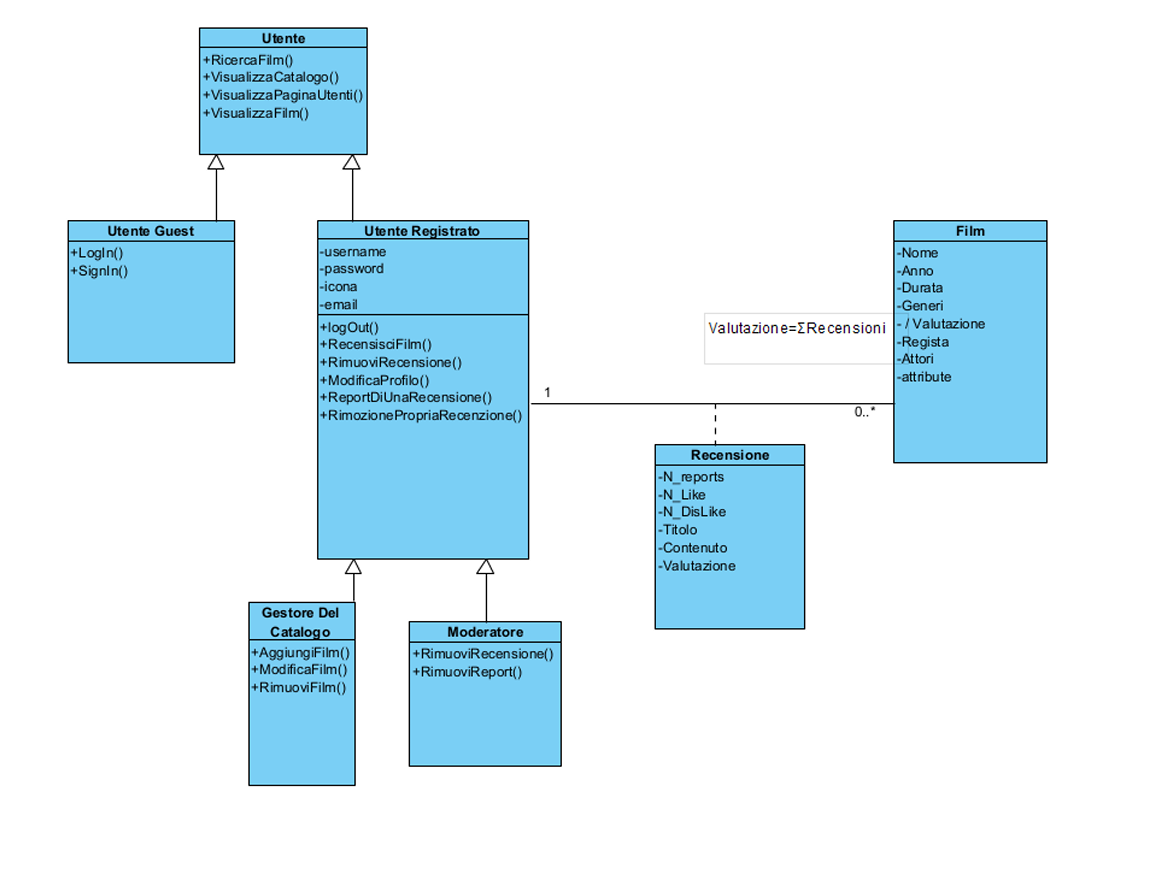
*Sicurezza dei dati****:*** saranno adottati protocolli di crittografia per proteggere le informazioni sensibili, come le credenziali degli utenti. Le password dovranno rispettare requisiti di complessità (es. lunghezza minima e inclusione di caratteri speciali).

*Connessioni protette****:*** l’intero sistema utilizzerà protocolli HTTPS per garantire comunicazioni sicure.

**1.3 Linee guida per la documentazione dell’interfaccia**

Le convenzioni utilizzate nell’implementazione delle funzionalità del sistema faranno riferimento alle specifiche di Java Sun.

**1.4 Ottimizzazione del modello a oggetti**



Il modello ad oggetti è rimasto invariato rispetto alla versione proposta nel RAD in quanto non sono stati trovati errori o imprecisioni.

**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.

**.mvn**, contenente i file di configurazione di Maven

**src**, contente i file sorgente del progetto

**main**

**java,** contentente I package files di Java

**resources,** contenente le risorse relative all’interfaccia utente

**static,** contenente le risorse statiche come CSS e JS files

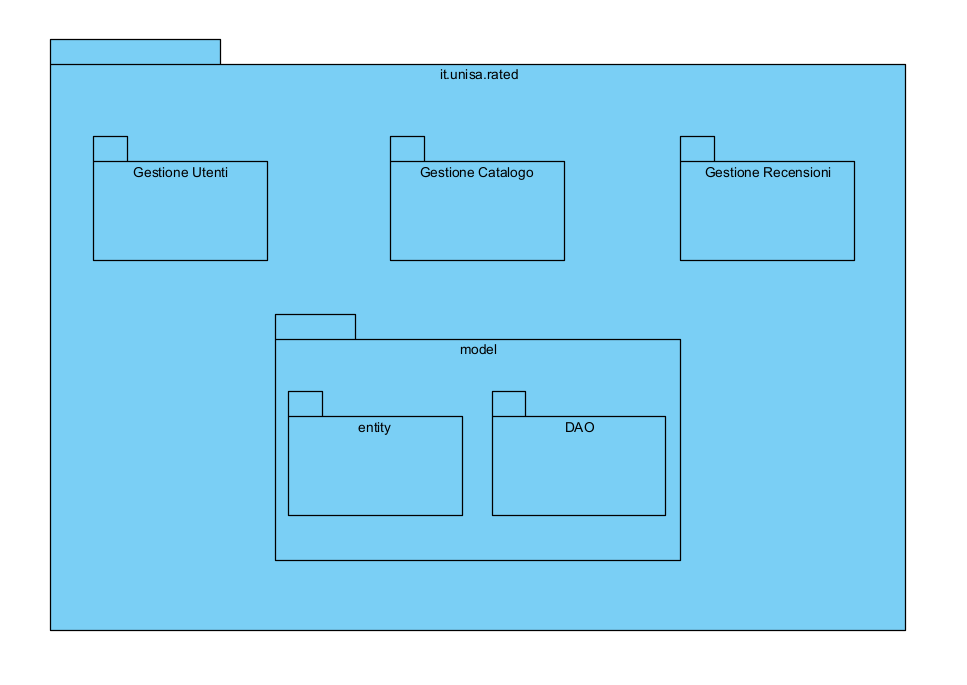
**templates,** contenente file HTML dinamici

**test**

**java,** contenente le classi di testing

**target,** contenente i file build i Maven

Il sistema è formato da un package generale chiamato *it.unisa.rated* all’interno del quale sarà presente un package per ogni sottosistema individuato. Oltre a questi sarà presente anche un package model , contenente le class entity e i DAO.



**3. Interfaccia delle classi**

**Package Gestione Utenti**

|  |  |
| --- | --- |
| Interfaccia | GestioneUtentiService |
| Descrizione | Gestione Utenti fornisce il servizio relativo all’autenticazione e all’aggiornamento degli account |
| Metodi | + LogIn(String email, String password) : Utente  + SingUp(String username, Blob icon, String email, String password) : Utente  + getUtente(String email) : Utente  + getUtenti() : Collection<Utente> |
| Invariante classe | // |

|  |  |
| --- | --- |
| Nome Metodo | + LogIn(String email, String password) : Utente |
| Descrizione | Il metodo LogIn consente di verificare se un utente esiste nel sistema ed è autorizzato ad accedere. |
| Pre-condizione | context GestioneUtentiService::autentica(String email, String password): Utente  pre: email != null and password != null |
| Post-condizione | context GestioneUtentiService::autentica(String email, String password): Utente  post: result != null and result.email = email |

|  |  |
| --- | --- |
| Nome Metodo | + SingUp(String username, Blob icon, String email, String password) : Utente |
| Descrizione | Il metodo SingUp consente di reggistrare un nuovo utente nel sistema |
| Pre-condizione | context GestioneUtentiService:: SingUp(String username, Blob icon, String email, String password) : Utente  pre: username != null and nome.size() > 0  pre: email != null and email.size() > 0  pre: password != null and password.size() >= 8  pre: self.getUtente(email) = null |
| Post-condizione | context GestioneUtentiService:: SingUp(String username, Blob icon, String email, String password) : Utente  post: result != null and self.getUtente(email) = result |

|  |  |
| --- | --- |
| Nome Metodo | + getUtente(String email) : Utente |
| Descrizione | Il mettodo getUtente() permette di trovare l’utente associato ad una mail |
| Pre-condizione | Context GestioneUtentiService::getUtente(String email) : Utente  pre email != null and email.size() > 0 |
| Post-condizione | Context GestioneUtentiService::getUtente(String email) : Utente  post result != null and self.getUtenti().exist(u | u.email = email)) |

|  |  |
| --- | --- |
| Nome Metodo | + getUtenti() : Collection<Utente> |
| Descrizione | Il metodo getUtenti() restituisce la lista di tutti gli utenti del sistema |
| Pre-condizione | // |
| Post-condizione | // |

**Package Gestione Catalogo**

|  |  |
| --- | --- |
| Interfaccia | GestioneCatalogoService |
| Descrizione | Gestione Catalogo fornisce il servizio relativo alla aggiunta, modifica e rimozione dei film |
| Metodi | + getFilm(String name) : Collection<Film> +addFilm(String name, int year, int length, String Attori):Film +updateFilm(Film m): Film +deleteFilm(Film m):boolean |
| Invariante classe | // |

|  |  |
| --- | --- |
| Nome Metodo | + getFilm(String name) : Collection<Camera> |
| Descrizione | Il metodo getFim(String name) permette di cercare un film tramite il suo nome |
| Pre-condizione | // |
| Post-condizione | // |

|  |  |
| --- | --- |
| Nome Metodo | +addFilm(String name, int year, int length, String Attori):Film |
| Descrizione | Il metodo addFilm permette di aggiungere/creare nuovi film nel catalogo |
| Pre-condizione | context GestioneCatalogoService:: addFilm(String name, int year, int length, String Attori):Film  pre: name != null and nome.size() > 0  pre: year != null  pre: length >0 |
| Post-condizione | context GestioneCatalogoService:: addFilm(String name, int year, int length, String Attori):Film  post: result != null and self.getFilm(name) = result |

|  |  |
| --- | --- |
| Nome Metodo | +updateFilm(Film f): Film |
| Descrizione | Il metodo updateFilm permette di salvare le modifiche effettuate su di un film |
| Pre-condizione | context GestioneCatalogoService:: updateFilm(Film f): Film  pre: name != null and nome.size() > 0  pre: year != null  pre: length >0 |
| Post-condizione | context GestioneCatalogoService:: addFilm(String name, int year, int length, String Attori):Film  post: result != null and self.getFilm(name) = result |

|  |  |
| --- | --- |
| Nome Metodo | +deleteFilm(Film m):Film |
| Descrizione | Il metodo permette di eliminare un determinato film dal Catalogo |
| Pre-condizione | context GestioneCatalogoService:: deleteFilm(Film m):Film: boolean  pre: m != null  pre: self.getFilm (“”) 🡪 include(m) |
| Post-condizione | context GestioneServiziService:: deleteFilm(Film m):Film: boolean  post: result = true and (not self.getFilm () 🡪 include(m)) |

**Package Gestione Recensioni**

|  |  |
| --- | --- |
| Interfaccia | GestioneRecensioniService |
| Descrizione | Gestione Recensioni fornisce il servizio relativo alla aggiunta, modifica e rimozione delle recensioni e della vautazione delle stesse |
| Metodi | +getRecensioni(Film f): Collection<Recensioni> +addRecensione(Film f, Utente u, String titolo, String Contenuto, int valutazione): Recensione +addValutazione(Film f, Utente u, Vote v, Recensione r) : boolean  +removeRecensione(Recensione r):boolean  +getValutazione (Film f, Utente u):Valutazione |
| Invariante classe | // |

|  |  |
| --- | --- |
| Nome Metodo | +getRecensioni(Film f): Collection<Recensioni> |
| Descrizione | Il metodo getRecensioni permette di ottenere tutte le recensioni di un determinato film |
| Pre-condizione | // |
| Post-condizione | // |

|  |  |
| --- | --- |
| Nome Metodo | +addRecensione(Film f, Utente u, String titolo, String Contenuto, int valutazione): Recensione |
| Descrizione | Il metodo addRecensione permette di aggiungere una recensione scritta da una persona ad un film |
| Pre-condizione | context GestioneRecensioniService:: addRecensione(Film f, Utente u, String titolo, String Contenuto, int valutazione): Recensione  pre: f != null pre: u != null  pre: titolo != null and titolo.length()>0 pre: Contenuto != null and Contenuto.legth()>0 pre: valutazione>0 and valutazione<=5 |
| Post-condizione | context GestioneRecensioniService:: addRecensione(Film f, Utente u, String titolo, String Contenuto, int valutazione): Recensione   post: result != null and self.getRecensione (titolo) = result |

|  |  |
| --- | --- |
| Nome Metodo | +addValutazione(Film f, Utente u, Vote v, Recensione r) : boolean |
| Descrizione | Il metodo addValutazione permette di aggiungere la valutazione di un utente rispetto ad una recensione di un film |
| Pre-condizione | context GestioneRecensioniService:: addValutazione(Film f, Utente u) : boolen pre: f != null pre: u != null  pre: v != null pre r != null |
| Post-condizione | context GestioneRecensioniService:: addValutazione(Film f, Utente u) : Valutazione  post: result = self.getValutazione (Film f, Utente u,) !=null |

|  |  |
| --- | --- |
| Nome Metodo | +removeRecensione(Recensione r):boolean |
| Descrizione | Il metodo removeRecensione permette di rimuovere una recensione |
| Pre-condizione | Context GestioneRecensioniService::removeRecensione(Recensione r) : boolean  pre: r!=null |
| Post-condizione | post: result = !self.getRecensioni(r.getFilm()).iclude(r) |

|  |  |
| --- | --- |
| Nome Metodo | +getValutazione (Film f, Utente u):Valutazione |
| Descrizione | Il metodo getValutazione permette di ottenere la valutazione di un utente su riguardante un film specificato |
| Pre-condizione | // |
| Post-condizione | Context GestioneRecensioniService:: getValutazione (Film f, Utente u):Valutazione post: result =recensione and recensione.getFilm()=f and recensione.getUtente()=u |

**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.