CineHub

Test Plan Document

# Introduzione

Questo documento contiene le specifiche atte a pianificare le fasi di testing di CineHub, verificando che vi sia assenza di incongruenze tra il comportamento effettivo e quello specificato. Lo scopo del Testing è quello di individuare quante più fault allo scopo di correggerli, permettendoci così di aumentare la fiducia che abbiamo nel sistema.

Le funzionalità da testare saranno estratte dai documenti precedentemente realizzati.

## Documenti Correlati

Dal RAD vengono estratti i requisiti funzionali che saranno a testati. Mentre dal SDD riprenderemo la divisione in tre layer, cioè: presentation, business logic e data layer.

# System Overview

CineHub è stato realizzato attraverso l’uso di alcuni componenti *off-the-shelf* open source che hanno facilitato la fase di implementazione, permettendoci di concentrare gli sforzi unicamente sulla logica di business.  
L’accesso hai dati è delegato alle JpaRepository che si pongono come layer tra il sistema e il DBMS permettendoci di evitare la scrittura di classi DAO.

Il sistema è suddiviso in tre layer principali: presentation, business logic e data layer. Il business logic layer è composto dai seguenti sottosistemi:

* Gestione Catalogo
* Catalogo
* Gestione Utenti
* Moderazione

A livello implementativo questi sottosistemi sono rappresentati dagli omonimi control che racchiudono le funzioni relative ad ognuno di essi. Ogni classe control utilizza delle classi service che contengono la vera e propria logica di business; le classi service utilizzano le JpaRepository per accedere ai dati.  
Abbiamo quindi una **architettura chiusa** dove ogni sottosistema può interagire unicamente con quello sottostante permettendoci una maggiore manutenibilità.

# Feature da testare

I requisiti funzionali che andremo a testare sono:

|  |  |
| --- | --- |
| RF3 | Il sistema deve permettere di cercare un film per nome e/o filtrare per genere. |
| RF4 | Il sistema deve fornire la possibilità ad un utente di registrarsi. |
| RF5 | Il sistema deve fornire la possibilità di eseguire il login |
| RF6 | Il sistema deve fornire la possibilità di visualizzare la propria pagina del profilo. |
| RF7 | Il sistema deve fornire la possibilità di segnalare la recensione di un altro utente. |
| RF8 | Il sistema deve fornire la possibilità al recensore di valutare un Film o una puntata di una Serie TV con un punteggio e un breve testo. |
| RF9 | Il sistema deve fornire la possibilità al recensore di mettere mi piace / non mi piace alla recensione di un altro recensore. |
| RF10 | Il sistema deve fornire la possibilità al recensore di rispondere alla recensione di un altro recensore. |
| RF18 | Il sistema deve permette di aggiungere un film, una puntata di una serie tv o una serie tv al catalogo. |
| RF19 | Il sistema deve permette di rimuovere un film, una puntata di una serie tv o una serie tv al catalogo |
| RF20 | Il sistema deve permette di modificare le informazioni un film, di una serie tv o di una puntata di una serie tv in catalogo |
| RF21 | Il sistema deve permettere di aggiungere persone del mondo dello spettacolo in modo da poterle associare a film o serie tv durante il loro inserimento. |

I requisiti funzionali: RF1, RF2, RF12, RF15 non verranno testati perché riguardano la semplice visualizzazione di informazioni.

I requisiti funzionali: RF13, RF15, RF17 non verranno testati perché il loro funzionamento è garantito dal componente *off-the-shelf* “Grid” del framework front-end Vaadin.

## Pass and Fail cirteria

Gli input vengono suddivisi in classi di equivalenza. Per il Test di Sistema gli input vengono individuati a partire dai casi d’uso, per il Test di Integrazione gli input vengono individuati a partire dalle specifiche presenti nel Object Design Document e per il Test di Unità abbiamo scelto un approccio black-box individuando gli input anche in questo caso a partire dalle specifiche presenti nel Object Design Document.

Se l’elaborazione dell’input genererà il risultato atteso allora la classe di equivalenza presa in considerazione avrà superato il test.

# Approach

La fase di testing partirà dal test delle singole unità, in modo da verificarne il funzionamento prima di integrarle nel sistema. Dopo il test di unità si procede con il test di integrazione, in cui la singola componente testata in isolamento viene testata con le altre. Dopo che tutte le componenti avranno superato il test di integrazione sarà effettuato il test di sistema nel quale si verificherà il funzionamento corretto.

## Test di unità

Nel test di unità considereremo le singole componenti in modo isolato, sfruttando un test driver e stub ove fosse necessario. Il test di unità è effettuato utilizzando tecniche black-box. Le JpaRepository non saranno testate in quanto componenti *off-the-shelf*.

## Test di integrazione

Una volta che una componente ha superato il test di unità viene integrata con le altre già testate, la strategia di integrazione è bottom-up. Ovviamente il procedimento verrà iterato per tutte le funzionalità implementate.

## Test di sistema

Una volta che tutte le componenti saranno state integrate si procederà al test di sistema. Ci assicureremo quindi che il sistema risponde ai requisiti definiti nel RAD.

# Testing Materials

Le tecnologie utilizzate per il test di unità sono JUnit per definire i singoli casi di test e Mockito per la realizzazione degli stub.

La tecnologia usata per il test di sistema è TestBench, basata su Selenium; è richiesto quindi un browser con JavaScript abilitato e un web driver.

# Scheduling

Ogni sviluppatore effettuerà il test di unità delle componenti da lui sviluppate, mentre i test di integrazione e di sistema saranno effettuati dal team di testing.