

Pisa University  
  
  
TASK 3  
LARGE-SCALE AND MULTI-STRUCTURED DATABASES

**“*PisaFlix 3.0” project documentation***  
academic year 2019-2020  
  
  
  
  
stefano petrocchi, Andrea Tubak, Francesco Ronchieri, Alessandro Madonna

Summary

[Design Document 3](#_Toc32165893)

[Description 3](#_Toc32165894)

[Requirements 3](#_Toc32165895)

[Main Actors 3](#_Toc32165896)

[Functional 3](#_Toc32165897)

[Non-Functional 4](#_Toc32165898)

[Use Cases 4](#_Toc32165899)

[Data Model 4](#_Toc32165900)

[Architecture 5](#_Toc32165901)

[Interface Design Pattern 5](#_Toc32165902)

# Design Document

## Description

Have you ever found yourself in a gloomy day? Everyone is at home, no one knows what to do and time seems to slow down. That’s the perfect time for a movie!

PisaFlix is a platform in which users can find quality and updated information regarding movies. It provides a service to help you to choose what film to watch. PisaFlix has a comment section that gives at the users the possibility to create a community around their favourite movies, exchanging opinions and news regarding them. It is also possible to add films to a favourite list in order to find them quicker. The possibility to see other user’s favourites is essential to find new friends with the same cinematic tastes. PisaFlix also offers the possibility to follow a user in order to be informed about the activity of that user. Lastly it is possible to view interesting statistics on films, useful both for normal users and for other people involved in the production of films.

PisaFlix offers services that will change the way users approach the world of the movie, providing them everything they need to enjoy at best their passions.

## Requirements

### Main Actors

The application will interact only with the **users**, distinguished by their privilege level:

* **Normal User**: a normal user of the application with the possibility of basic inaction.
* **Social Moderator**: a trusted user with the possibility to moderate the comments.
* **Moderator**: a verified user with the possibility to add and modify elements in the application, like films, cinemas or projections.
* **Admin:** an administrator of the application, with possibility of a complete interaction.

### Functional

1. *Users* can **view** the list of **Movies** available on the platform.
2. *Users* can **view** the information about a specific *Movie*.
3. *Users* can **view** the ***statistics*** of a *Film* page.
4. *Users* can **view** a set of **analytics** on Movie and other user.
5. *Users* can **register** an account on the platform.
6. *Users* can **log in** as *Normal users* on the platform in order to do some specific operations:
   1. If logged a *Normal user* can **add/remove** to **favourite** a *Movie*.
   2. If logged a *Normal user* can **comment** a *Movie*.
   3. If logged a *Normal user* can **modify** his *Movie Comments*.
   4. A *Normal user* can **modify/delete** his account.
7. *Users* that can **log in** as *Social moderator* can do all operation of a *Normal user* plus:
   1. If logged as *Social moderator* can **delete** other users’ comments.
   2. If logged as *Social moderator* can **delete** other users’ posts.
   3. If logged as *Social moderator* can **recruit** others *Social moderator*s.
8. *Users* that can **log in** as M*oderator* can do all operation of a *Social moderator* plus:
   1. If logged a *Moderator* can **add/delete/modify** a *Movie*.
   2. If logged as *Moderator* can **recruit** other *Moderator*s
9. *Users* that can **log in** as*Admins* can do all operation of a M*oderator* plus:
   1. If logged an *Admin* can **delete** another user’s account.
   2. If logged as *Admin* can **recruit** other *Admin*s.

### Non-Functional

1. The application’s focus is the *quality* of the information provided to users.
2. The application needs to be **consistent**, in order to provide correct information to all the users.
3. The application needs to be **tolerant to partitions**, in order to scale the system if needed, preserving the consistency.
4. The application needs to store **replicas** of the data in case of server fault, all the replicas need to be **consistent**.
5. The transactions must be **monotonic:** every user must see the last version of the data and modifications are done in the same order that are committed.
6. The application needs to be *usable* and *enjoyable* for the user, therefore the system needs **limited response times**.
7. The *password* must be protected and stored *encrypted* for privacy issues.

## Use Cases

MODIFICARE UML USE CASES aggiungendo la possibilità per qualsiasi utente di seguire un altro utente; aggiungere la possibilità per ogni utente di seguire un film; aggiungere la possibilità di scrivere post (volendo anche taggando uno specifico film); aggiungere la possibilità di vedere la propria bacheca.

## Data Model

Abbiamo i nodi Utente, Film e Post. I collegamenti sono Utente Follows Utente, Utente Follows Film, Utente Writes Post, Post Tags Film.

## Architecture

Users can use a java application with a **GUI** to take advantage of all the functionalities of the platform.

The client Application it’s made in *Java* using **JavaFX framework** for the *front-end* and the **MongoDB driver** to manage *back-end* functionalities. **Services** and ***JavaBean* objects** compose the *middleware* infrastructure that connect *front-end* and *back-end.*

### Interface Design Pattern

The graphic user interface was build following the software design pattern of **Model-View-Controller**.

#### **Model**

**Services** module represent the *model* and is the central component of the pattern. It is the application's dynamic data structure, independent of the user interface. It directly manages logic and rules of the application receiving inputs from the controller. The model is also responsible for managing the application’s data in form of JavaBean objects, exchanging them with the controller.

#### **View**

The **fxml files** represents the *view* and are responsible fosr all the components visible in the user’s interface.

#### **Controller**

The **page controllers** are the *controller* of the application. They receive inputs from the *view* and converts them into commands for the *model* or *view* itself. Controllers can also validate inputs and data without the intervention of the *model*. Data is exchanged between *model* and *controller* using JavaBean objects.

Immagine che contiene screenshot

Descrizione generata automaticamente

