



PISA UNIVERSITY

TASK 2
LARGE-SCALE AND MULTI-STRUCTURED DATABASES

“PISAFLIX 2.0” PROJECT DOCUMENTATION

ACADEMIC YEAR 2019-2020

STEFANO PETROCCHI, ANDREA TUBAK, FRANCESCO RONCHIERI, ALESSANDRO MADONNA



SUMMARY

Design Document..... 3

 Description 3

 Requirements 3

 Main Actors 3

 Functional..... 3

 Non-Functional 4

 Analytics 4

 Use Cases..... 5

ANALYTICS USE CASES..... 5

 View Average Rating 5

 View Ranking 6

 View Engagement Activity 7

Analysis Classes 7

Data Model..... 8

Architecture..... 8

 Interface Design Pattern 9

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 provide a service to help you to choice 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 users favourites it is essential to find new friends with the same cinematic tastes. 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 *Projections* scheduled for a *Film*.
4. *Users* can **view** the **statistics** of a *Cinema* or *Film* page.
5. *Users* can **view** a set of **analytics** on *Movie* and other user.
6. *Users* can **register** an account on the platform.
7. *Users* can **log in** as *Normal users* on the platform in order to do some specific operations:
 - a. If logged a *Normal user* can **add/remove** to **favourite** a *Movie*.
 - b. If logged a *Normal user* can **comment** a *Movie*.
 - c. If logged a *Normal user* can **modify** his *Movie Comments*.
 - d. A *Normal user* can **modify/delete** his account.
8. *Users* that can **log in** as *Social moderator* can do all operation of a *Normal user* plus:
 - a. If logged as *Social moderator* can **delete** other users' comments.
 - b. If logged as *Social moderator* can **recruit** others *Social moderators*.
9. *Users* that can **log in** as *Moderator* can do all operation of a *Social moderator* plus:
 - a. If logged a *Moderator* can **add/delete/modify** a *Movie/Projection*.

- b. If logged as *Moderator* can **recruit** other *Moderators*
- 10. Users that can **log in** as *Admins* can do all operation of a *Moderator* plus:
 - a. If logged an *Admin* can **delete** another user's account.
 - b. If logged as *Admin* can **recruit** other *Admins*.

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.

ANALYTICS

In PisaFlix there are three main types of analytics/statistics:

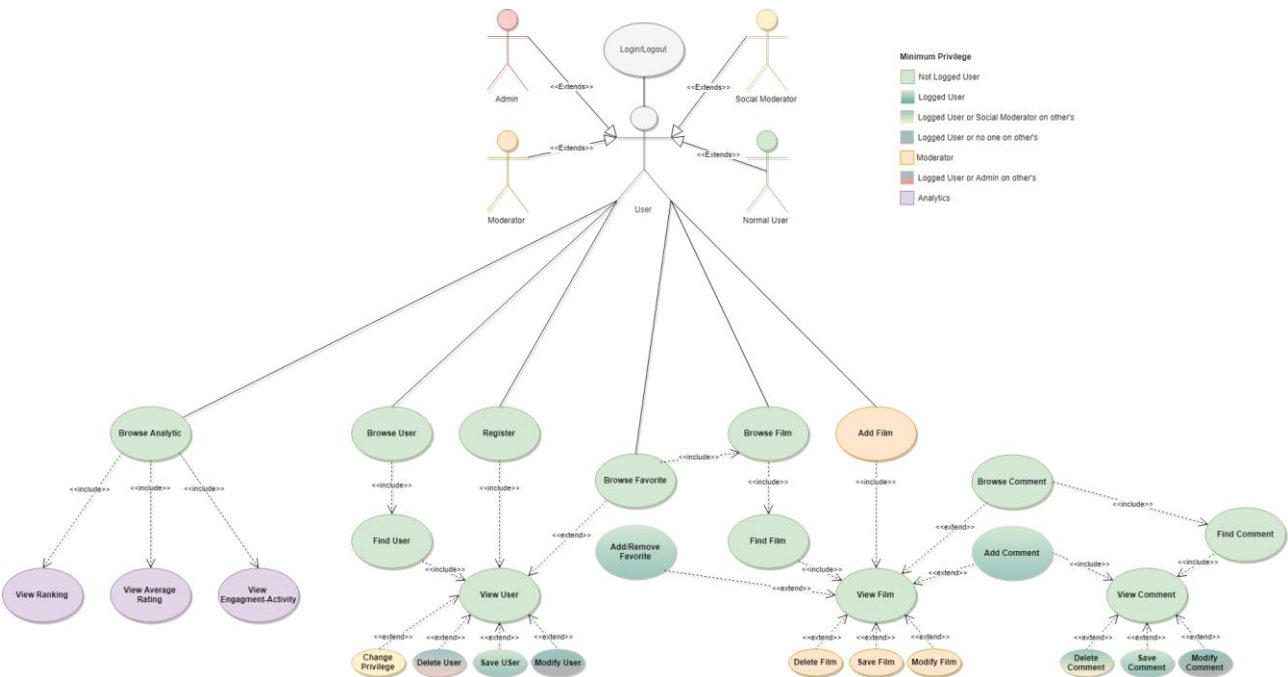
1. Chosen a temporal interval, the application shows the mean value of movie rating grouped by genre.
2. Chosen a film and two years (one for the beginning and one for the end of a period), the application shows the engagement of the chosen film by year, in the interval of time specified before. Moreover, the application will show also the composition of the value of the engagement for a chosen year.

This is the formula for the engagement $E = (3 * f) + (2 * c) + (1 * v)$. It is basically a weighted sum where E is the engagement, f is the favourite number of the film, c is the number of comments of that film, and v is the number of time that the page of the film has been visited.

3. Chosen a temporal interval, the application shows a table with the most involved user in the application.

This is done by calculating a value that is a weighted sum of what the user do on the application. More on the point, the formula is $I = (3 * c) + (2 * f) + (1 * v)$. Where c stand for the number of comments that the user does, f for the number of favorites film of the user, and v for the number of page requested by the user.

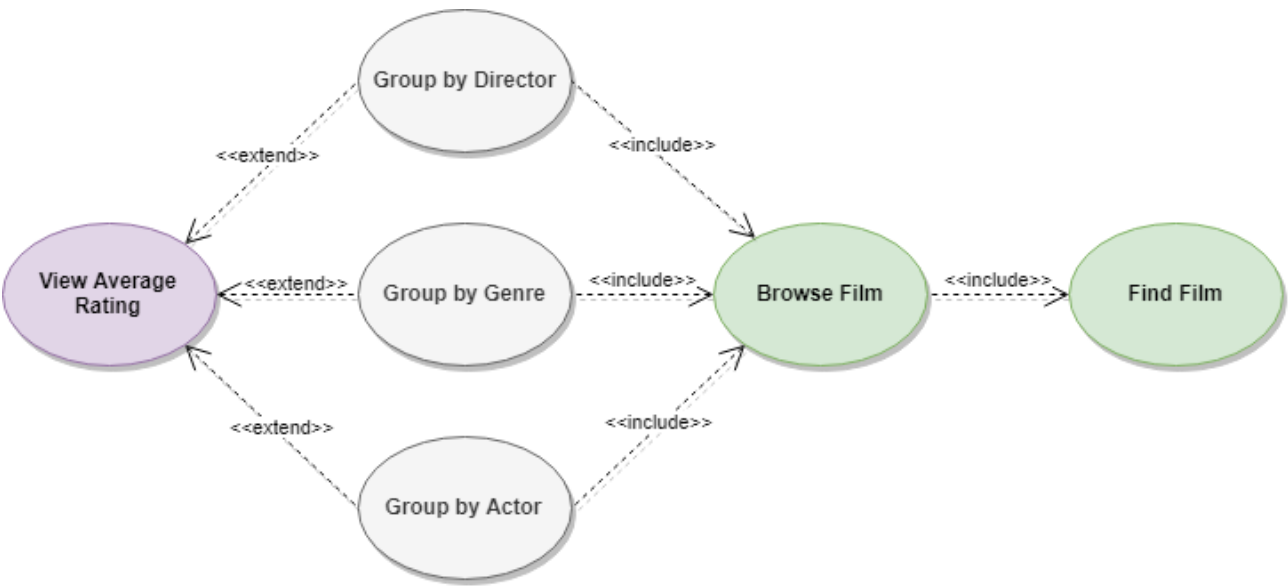
USE CASES



ANALYTICS USE CASES

Let us see the use cases of the analytics reported in the diagram above.

VIEW AVERAGE RATING



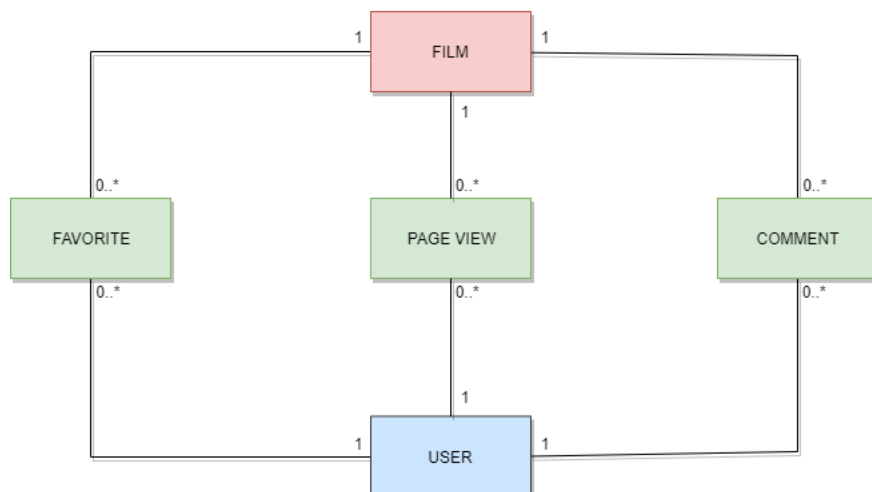
VIEW RANKING



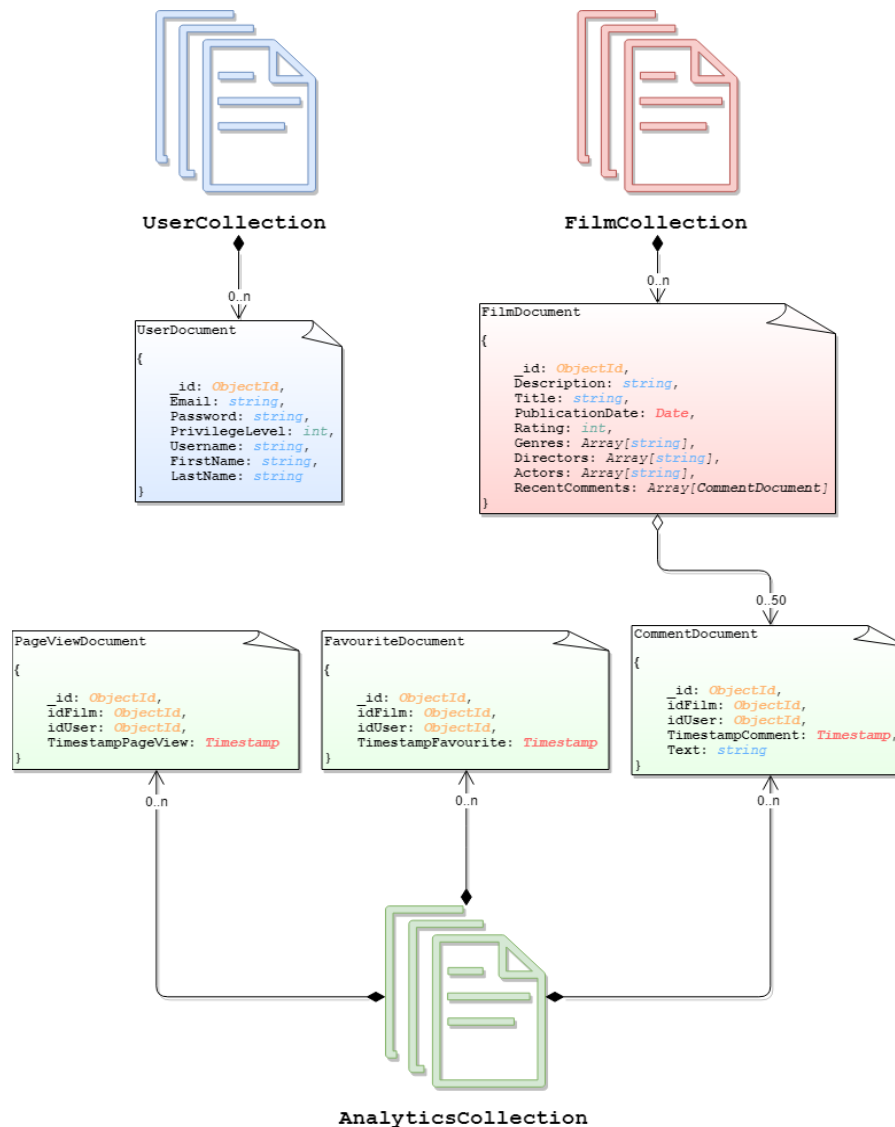
VIEW ENGAGEMENT ACTIVITY



ANALYSIS CLASSES



DATA MODEL



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 fofsr 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.

