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

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! If you live within the Pisan suburb and you want to enjoy the best cinematic experience, PisaFlix is what you need.

PisaFlix is a platform in which users can find quality and updated information regarding movies and cinemas in the Pisa area. It gives the possibility to know which movies are available, in which cinema and at what time all the projections are scheduled, there is also the possibility to book a seat for a specific projection. PisaFlix has also a comment section both for cinemas and movies. The comment section gives at the users the possibility to create a community around their favourite cinemas and movies, exchanging opinions and news regarding them. It is also possible to add cinemas and 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 film and cinema pages, useful both for normal users and for cinema owners.

PisaFlix offers services that will change the way users approach the world of cinema, 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/Cinemas** available on the platform.
2. *Users* can **view** the information about a specific *Movie*.
3. *Users* can **view** the information about a specific *Cinema*.
4. *Users* can **view** the *Projections* scheduled in a *Cinema*.
5. *Users* can **view** the *Projections* scheduled for a *Film*.
6. *Users* can **view** the ***statistics*** of a *Cinema* or *Film* page.
7. *Users* can **view** the list of ***favourites*** a user.
8. *Users* can **register** an account on the platform.
9. *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/Cinema*.
    b. If logged a *Normal user* can **comment** a *Movie*/*Cinema*.
    c. If logged a *Normal user* can **modify** his *Movie*/*Cinema Comment*.

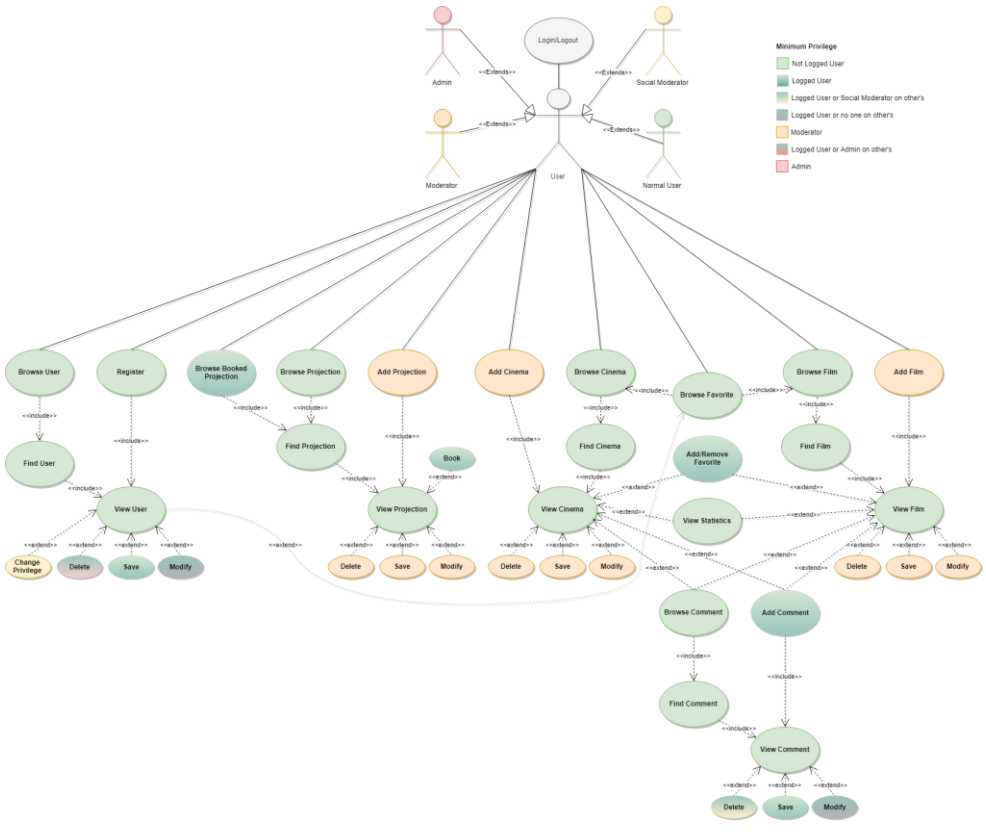
d. If logged a *Normal user* can **book** a seat for a *projection*.
e. If logged a *Normal user* can **view** the list of ***booked*** *Projections*.
f. A *Normal user* can **modify/delete** his account.

10. *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 moderator*s.
11. *Users* that can **log in** as M*oderator* can do all operation of a *Social moderator* plus:
    a. If logged a *Moderator* can **add/delete/modify** a *Movie/Cinema/Projection*.
    b. If logged as *Moderator* can **recruit** other *Moderator*s
12. *Users* that can **log in** as *Admins* can do all operation of a M*oderator* plus:
    a. If logged an *Admin* can **delete** another user's account.
    b. If logged as *Admin* can **recruit** other *Admin*s.

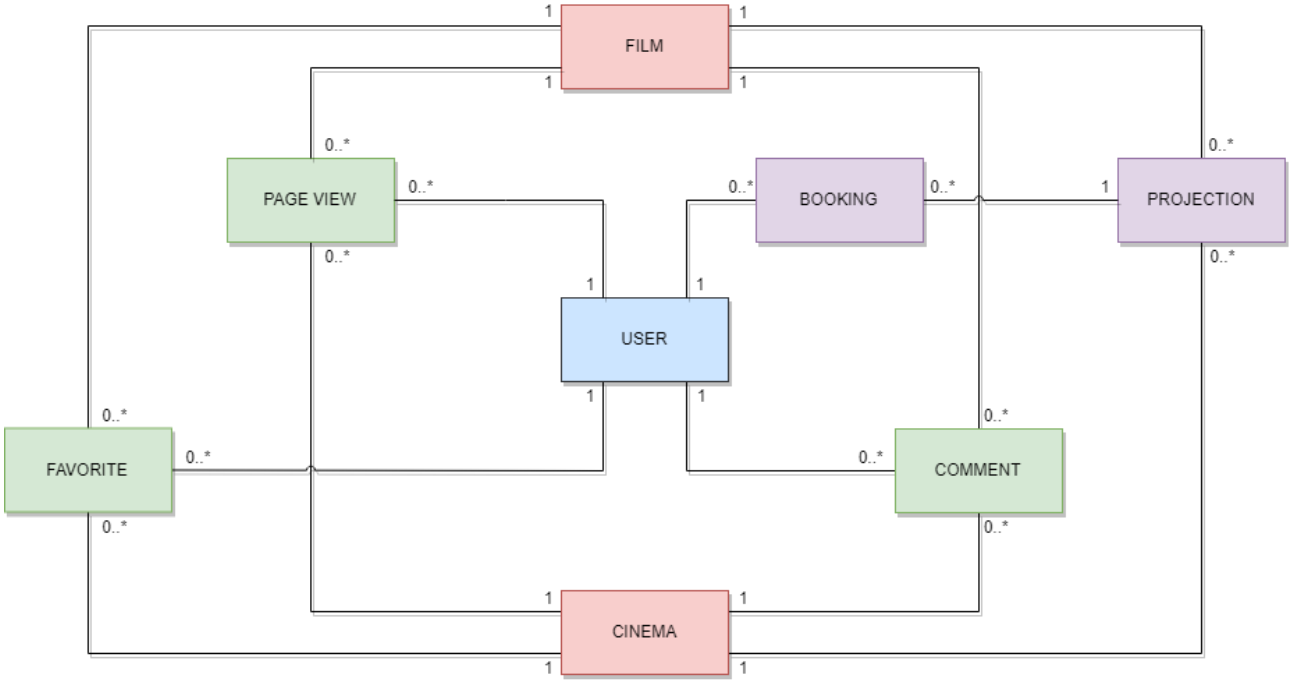
NON-FUNCTIONAL

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 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.

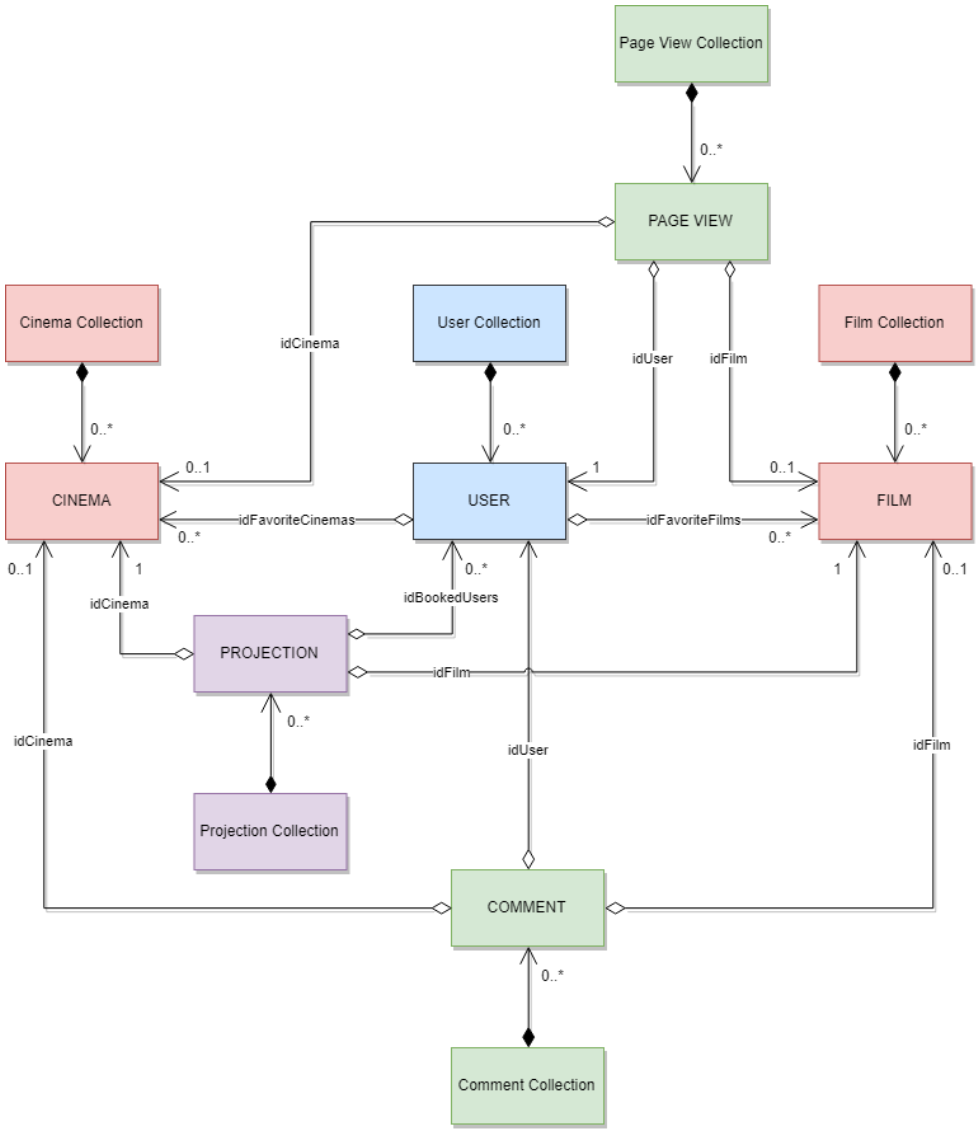
USE CASES



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 for 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.