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

DESCRIPTION

PisaFlix 3.0 is a social network, oriented to the discussion of film. An user can follow other users, to see their posts, or a film, to see the post of other user on that film.

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.
- **Admin:** an administrator of the application, with possibility of a complete interaction.

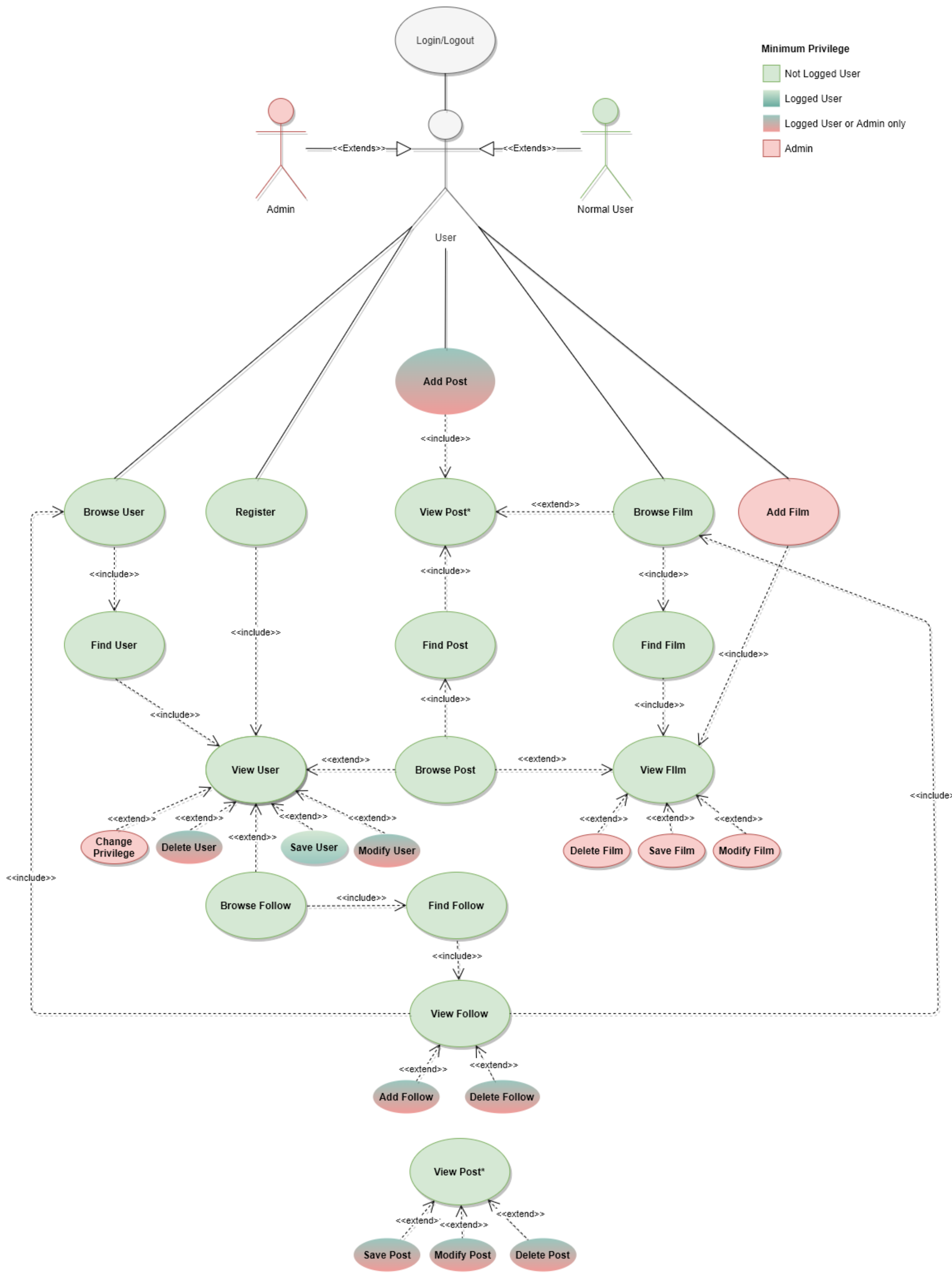
FUNCTIONAL

1. *Users* can **view** the list of **Movies** available on the platform.
2. *Users* can **view** the posts about a specific *Movie*.
3. *Users can **view** the list of **Users** on the platform.*
4. *Users can **view** the posts of a specific 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 other operations:
   a. If logged a *Normal user* can **follow/unfollow** a *Movie*.
   b. If logged a *Normal user* can **follow/unfollow** a User.
   c. If logged a *Normal user* can **write** a Post on a *Movie*.
   d. If logged a *Normal user* can **modify** his ***Posts***.
   e. A *Normal user* can **modify/delete** his account.
7. *Users* that can **log in** as *Admin* can do:
   a. If logged an *Admin* can **delete** another user's account.
   b. If logged as *Admin* can **recruit** other *Admin*s.
   c. If logged as Admin can **add/remove** a *Movie*.
   d. If logged as *Admin* can **delete** *Posts* of users.

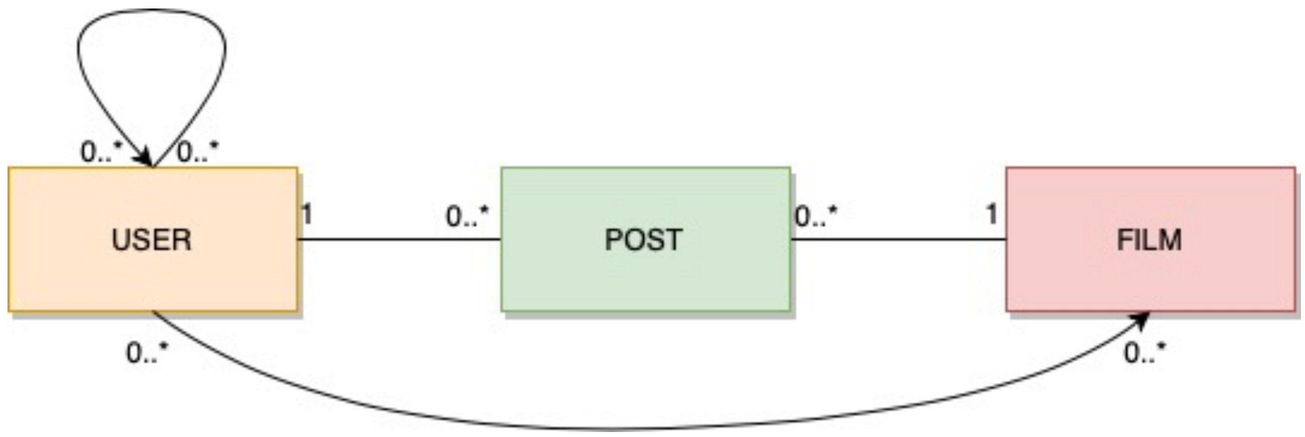
NON-FUNCTIONAL

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 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.
4. The application needs to be *usable* and *enjoyable* for the user, therefore the system needs **limited response times**.
5. The *password* must be protected and stored *encrypted* for privacy issues.

USE CASES

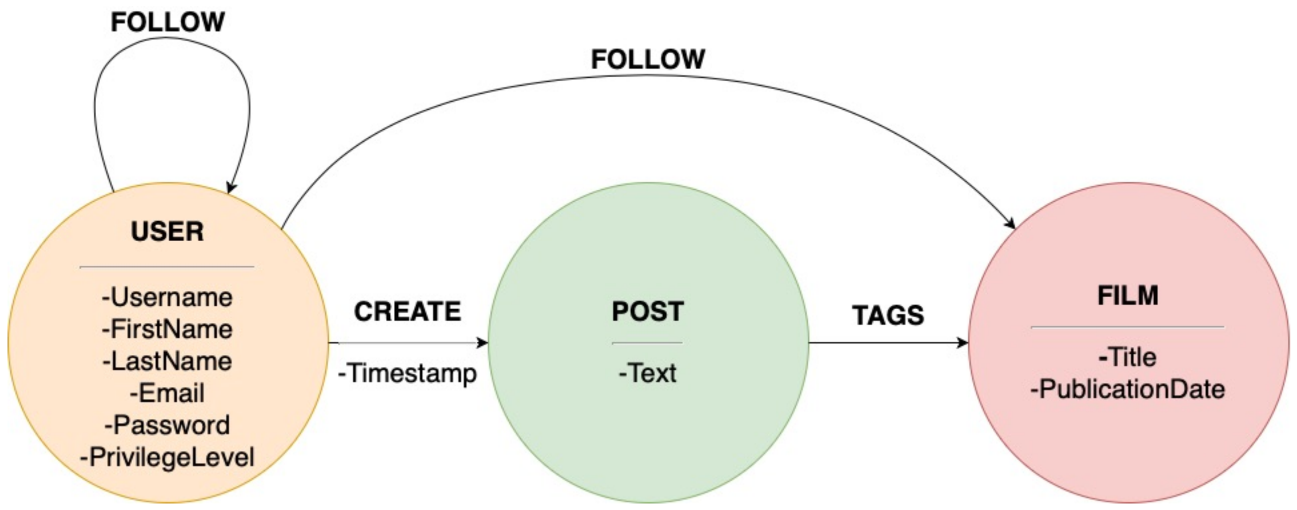

Login/Logout
Admin
<<Extends>>
<<Extends>>
Normal User
User
Minimum Privilege
Not Logged User
Logged User
Logged User or Admin only
Admin
Add Post
<<include>>
View Post*
<<extend>>
Browse User
Register
Browse Film
Add Film
<<include>>
Find User
Find Post
Find Film
<<include>>
View User
Browse Post
View Film
<<extend>>
Change Privilege
Delete User
Save User
Modify User
Delete Film
Save Film
Modify Film
Browse Follow
Find Follow
View Follow
Add Follow
Delete Follow
View Post*
Save Post
Modify Post
Delete Post

ANALYSIS CLASSES

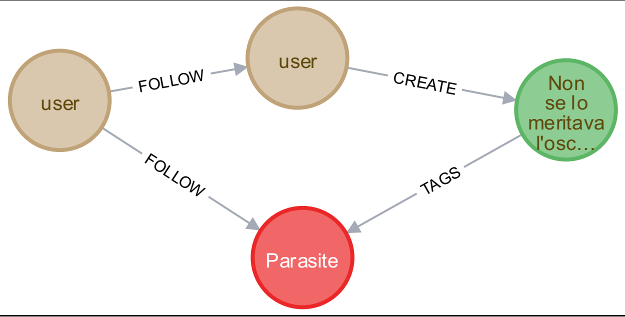


DATA MODEL

We have basically three entities, User, Film, and Post. The relation between Users is of type “follow”, such as the relation between User and Film. The relation between User and Post is of type “create” and contains a property Timestamp. The relation between Post and Film, is of type “Tags”.



EXAMPLE



```
{firstName:user, Email:user1@mail.com, LastName:1, Username:user1,
PrivilegeLevel:0, Password:pass}

{Title:Parasite, PublicationDate:"2019-11-07T00:00:00Z", WikiPage:url}

{firstName:user, Email:user2@mail.com, LastName:2, Username:user2,
PrivilegeLevel:0, Password:pass}

{Text:Non meritava l'Oscar}
```

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

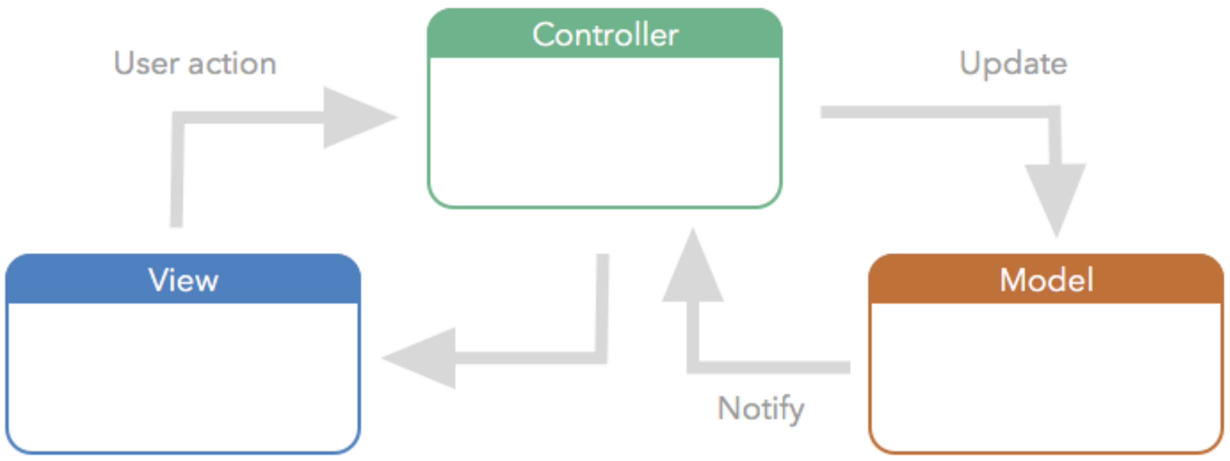
Services module represent the *model* and is the central component of the pattern. It is the application's dynamic data structure, independent of the user interface. It directly manages logic and rules of the application receiving inputs from the controller. The model is also responsible for managing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


User action
Controller
Update
View
Model
Notify