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 films. A User can visit the profiles of other users and see the pages related to films. In those pages, the User, will find either all the post written by the user or the most recent posts which tag the film. You can follow films and users in order to view the posts that concern them on the home page. New users and movies will be continually suggested, based on your friends and their favourite movies. Finally, a mechanism of privileges guarantee the quality of the contents, uploaded only by trusted users, and the possibility of moderate the posts present within the application.

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 interaction*.
- **Social Moderator**: a trusted user with the possibility to *moderate* the posts.
- **Moderator**: a verified user with the possibility to add and *modify* elements in the application, like film pages.
- **Admin:** an *administrator* of the application, with possibility of a *complete interaction*.

FUNCTIONAL

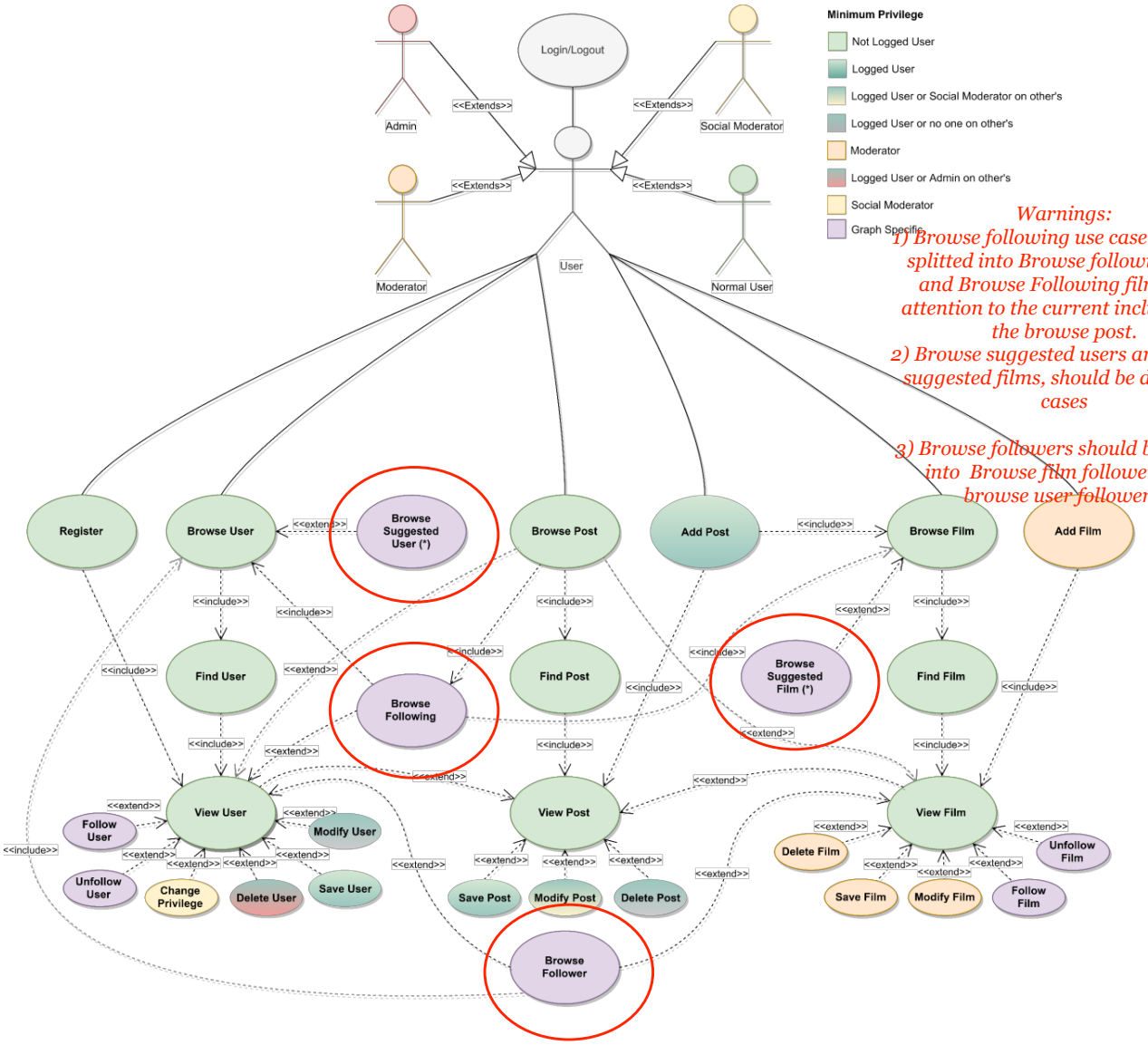
1. *Users* can **view** the list of **Movies** available on the platform.
2. *Users* can **view** the posts about a specific *Movie*.
3. *Users can **view** the list of **Users** on the platform.*
4. *Users can **view** the posts of a specific User.*
   a. *Users* can **register** an account on the platform.
5. *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 ***view*** Posts of his following *Movies* and *Users* on the *home page*.
   e. If logged a *Normal user* can ***view*** suggestions on *Movies* to follow.
   f. If logged a *Normal user* can ***view*** suggestions on *Users* to follow.
   g. If logged a *Normal user* can **write** a ***Post.***
   h. If logged a *Normal user* can **modify** his ***Posts***.
   i. A *Normal user* can **modify/delete** his account.
6. *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.
7. *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*.
   b. If logged as *Moderator* can **recruit** other *Moderator*s
8. *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 focus of the application is the *quality* of the information provided to the 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 in which they 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



SUGGESTIONS

The suggestions are shown only if the user is logged in. The suggestions can be found in the initial pages of the *browsers*, the page is filled with the suggestions from the highest priority to the lowest until exhaustion. If the suggestions are not enough to fill the page, the most recent films\users, that have not been already suggested, are chosen to complete it.

SUGGESTED FILMS

There are three levels of suggestions with different priorities:

- **Very Suggested:** They have the highest priority, given a user **U1** if **U1** is following user **U2** and **U2** is following a movie **F** and has also posted on **F**, then **F** is *very suggested* to **U1**.
- **Suggested:** They have the second priority level, if a user **U1** is following user **U2** and **U2** is following a film **F**, then **F** is *suggested* to **U1**.
- **Commented by Friend:** They have the lowest priority level, if a user **U1** follows a user **U2** who posted on a movie **F**, then **F** is suggested as "*commented by a friend*" at **U1**.

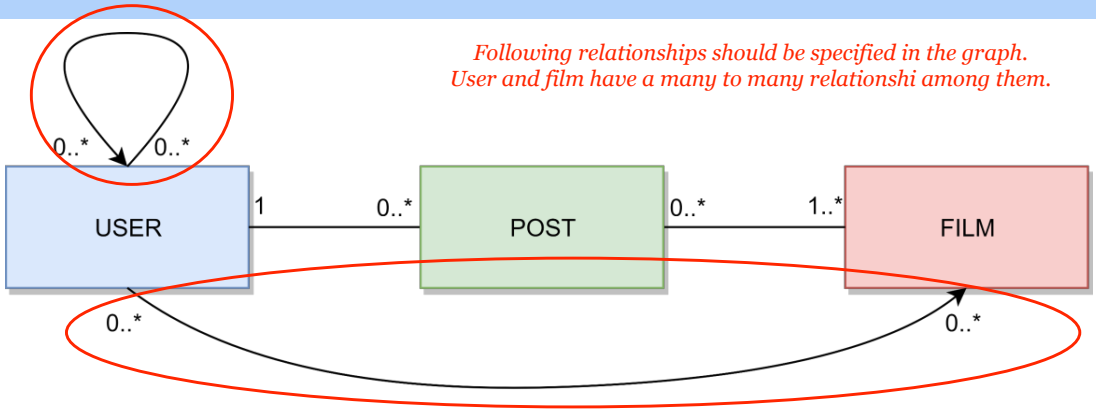
SUGGESTED USERS

There are two levels of suggestions with different priorities:

- **Very Suggested:** They have the highest priority, given a user **U1** if **U1** is following user **U2** and **U2** is following user **U3**, then **U3** is *very suggested* to **U1**.
- **Suggested:** They have the lowest priority level, if a user **U1** is following user **U2** and **U2** is following a user **U3** and **U3** is following a user **U4**, then **U4** is *suggested* to **U1**.

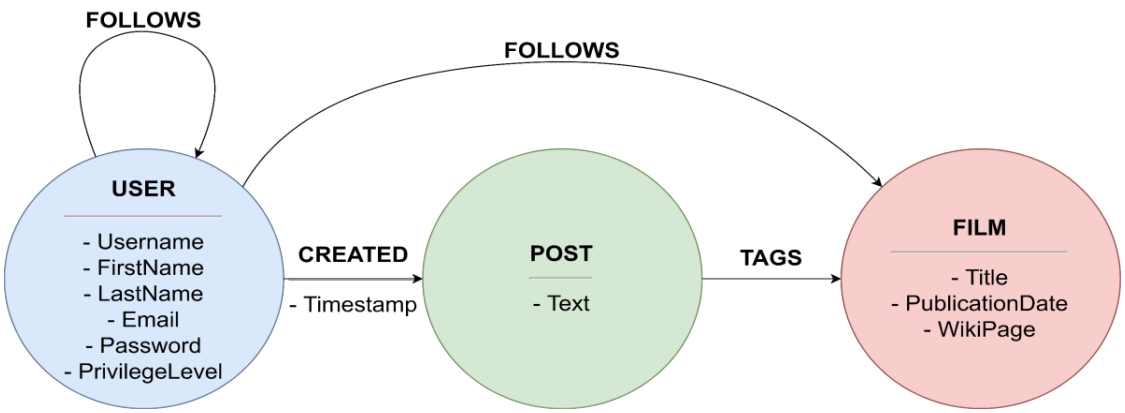
The description of the queries related to these suggestions is present in the Suggested Users and Movies section.

ANALYSIS CLASSES



DATA MODEL

We have basically three entities, User, Film, and Post. The relation between Users is of type “follows”, 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”.



RELEVANT QUERIES

COUNT FOLLOWING

This requirement was missing in the functional requirements

This query count how many users and films are followed by a specific user and is used to fill up the stats of that user when is loaded his profile page:

Domain-specific	Graph-centric
How many users/films are followed by a specific user?	How many outgoing (Follows) edges does the (User) node take into consideration has?

```
1.  MATCH (u1:User)-[:FOLLOWS]->(u2:User), (u1:User)-[:FOLLOWS]->(f:Film)
2.  WHERE ID(u1) = $userId
3.  RETURN count(DISTINCT u2) AS followingUsers, count(DISTINCT f) AS followingFilms
```

For sake of brevity descriptions of further counting queries have been omitted, which have the same structure, including:

- How many Posts have been written by a specific User?
- How many Users follow a specific Film?
- How many Posts tag a specific Film?
- How many followers have a specific User?

These queries are all used to generate the statistics of the pages of the films and users and concern the counting of outgoing and ingoing relations from the node under consideration.

SUGGESTED USERS AND MOVIES

The behavior of this suggestions have been described in the chapters *Suggested Users* and *Suggested Films*.

This are the queries to get the *suggested* and *very suggested* users.

Domain-specific	Graph-centric
What are the *suggested* users for a specific user? (The users that are followed by a followed user of a followed users of the user take into consideration)	What are the nodes that have an exact distance of three (*Follow*) ingoing hopes from the node take into consideration and are of the same type (*User*)?

```
1.  MATCH (u1:User)-[:FOLLOWS]->(u2:User)-[:FOLLOWS]->(:User)-[:FOLLOWS]->(u:User)
2.  WHERE ID(u1) = $userId
3.  AND NOT (u1)-[:FOLLOWS]->(u)
4.  AND NOT (u2)-[:FOLLOWS]->(u)
5.  RETURN u
```

Domain-specific	Graph-centric
What are the *very suggested* users for a specific user? (The users that are followed by a followed user of the user take into consideration)	What are the nodes that have an exact distance of two (*Follow*) ingoing hopes from the node take into consideration and are of the same type (*User*)?

```
1.  MATCH (u1:User)-[:FOLLOWS]->(u2:User)-[:FOLLOWS]->(u:User)
2.  WHERE ID(u1) = $userId
```

```
3.  AND NOT (u1)-[:FOLLOWS]->(u)
4.  RETURN u
```

This are the queries to get the *commented by a friend*, *suggested* and *very suggested* films.

Domain-specific	Graph-centric
What films have been *commented by a friend* of a specific user? (The films that have a post that tags them, created by a followed user of the user taken into consideration)	What are the nodes that have an ingoing (*Tags*) edge from a node that has an ingoing (*Created*) edge from a (*User*) node that has an ingoing (*Follows*) edge from the node taken into consideration and don't have an ingoing (*Follow*) edge from the (*User*) nodes?

```
1.  MATCH (u1:User)-[:FOLLOWS]->(u2:User)-[:CREATED]->(:Post)-[:TAGS]->(f:Film)
2.  WHERE ID(u1) = $userId
3.  AND NOT (u1)-[:FOLLOWS]->(f)
4.  AND NOT (u2)-[:FOLLOWS]->(f)
5.  RETURN f
```

Domain-specific	Graph-centric
What are the *suggested* films for a specific user? (The films that are followed by a followed user of the user take into consideration)	What are the (*Film*) nodes that have an exact distance of two (*Follow*) ingoing hopes from the (*User*) node take into consideration?

```
1.  MATCH (u1:User)-[:FOLLOWS]->(u2:User)-[:FOLLOWS]->(f:Film)
2.  WHERE ID(u1) = $userId
3.  AND NOT (u1)-[:FOLLOWS]->(f)
4.  RETURN f
```

Domain-specific	Graph-centric
What are the *very suggested* films for a specific user? (The films that are followed and have a post that tags them by a followed user of the user take into consideration)	What are the (*Film*) nodes that have two ingoing (*Follow* and *Tags*) edges from a same (*User*) node that has an ingoing (*Follows*) edge from the (*User*) node take into consideration and no ingoing (*Follows*) edges from that node?

```
1.  MATCH (u1:User)-[:FOLLOWS]->(u2:User)-[:FOLLOWS]->(f:Film)
2.  WHERE ID(u1) = $userId
3.  AND NOT (u1)-[:FOLLOWS]->(f)
4.  AND (u2)-[:CREATED]->(:Post)-[:TAGS]->(f)
5.  RETURN f
```

FOLLOW AND UNFOLLOW

These are examples of two queries for movie follow and unfollow, analogous queries for the users have been omitted for brevity:

Domain-specific	Graph-centric
A specific user follows a specific film.	Given two nodes (*User* and *Film*), creates a (*Follows*) edge outgoing from the first (*User*) node and ingoing in the second (*Film*) node.

```
1.  MATCH (u:User),(f:Film)
```