

Pisa University  
  
  
TASK 1  
LARGE-SCALE AND MULTI-STRUCTURED DATABASES

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

Summary

[Analysis Document 3](#_Toc23496033)

[Introduction 3](#_Toc23496034)

[Requirements 3](#_Toc23496035)

[Functional 3](#_Toc23496036)

[Non-Functional 3](#_Toc23496037)

[Software Architecture 3](#_Toc23496038)

[Use Cases Diagram 4](#_Toc23496039)

[Class Diagram 4](#_Toc23496040)

[Database Main Entities Diagram 4](#_Toc23496041)

[Project Document 5](#_Toc23496042)

[Software Architecture 5](#_Toc23496043)

[Classes 5](#_Toc23496044)

[Test Document 6](#_Toc23496045)

# Analysis Document

## Introduction

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 experience, PisaFlix is what you need.

PisaFlix is a platform in which you’ll find all of the information regarding movies and cinemas in the Pisa area. It gives you the possibility to know which cinema is available, which film you could watch and at what time all of the projections are due. PisaFlix has also a comment section both for cinemas and movies. This allows people to express their opinion, and, by doing so, providing others some really valuable information. Everyone who’s still unsure about what to do next will receive a great deal of help by this functionality. We believe PisaFlix offers a complete package of services, that will have a huge impact on the quality of the decisions made by our customers. Proving you everything you need to have a well informed choice is not only our goal, but also a pleasure.

## Requirements

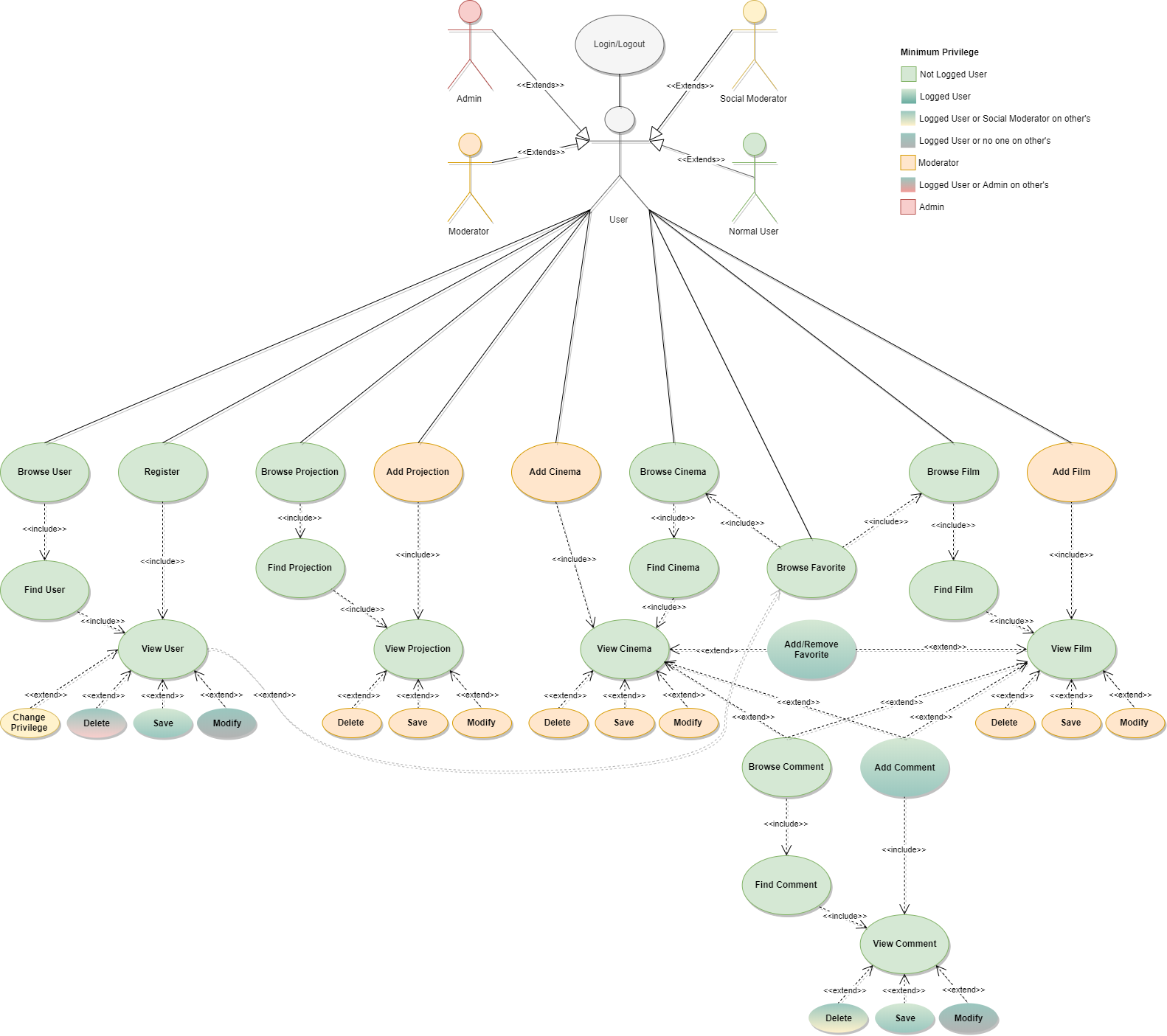
### Functional

1. *Users* can **view** the list of **Movies/Cinemas** available on the platform.
2. *Users* can **view** the specific information about a *Movie* (es. category, publish date ecc…).
3. *Users* can **view** the specific information about a *Cinema* (es. Name, Address).
4. *Users* can **view** the *Projections* scheduled in a *Cinema*.
5. *Users* can **view** the *Projections* scheduled for a *Film*.
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:
   1. If logged a *Normal user* can **add/remove to favorite** a *Movie/Cinema*.
   2. If logged a *Normal user* can **view** a *favorites*.
   3. If logged a *Normal user* can **comment** a *Movie*/*Cinema* and express his opinion about it.
   4. If logged a *Normal user* can **change** a *Movie* *Comment* that he expressed before or delete it.
   5. A *Normal user* can **modify/delete** his account.
8. *Users* can **log in** as *Social moderator* that can do all operation of a *Normal user* plus:
   1. If logged as *Social moderator*.
   2. If logged as *Social moderator* can **delete** others users comments.
   3. If logged as *Social moderator* can **recruit** other *Social moderator* giving his same privileges level or below.
9. *Users* can **log in** as M*oderator* that can do all operation of a *Social moderator* plus:
   1. If logged an *Moderator* can **add/delete/modify** a *Movie/Cinema/Projection*.
10. *Users* can **log in** as*Admins* that can do all operation of a M*oderator* plus:
    1. If logged an *Admin* can **delete** other user’s account.

### Non-Functional

1. The systems must be on 24/24.
2. The system must support hundred of concurrent access.
3. The response time must be in the order of 1-10 ms.
4. The password must be protected and stored encrypted for privacy issues.

## Use Cases Diagram



## Class Diagram



## Database Main Entities Diagram



# Project Document

## Software Architecture

The aim of this project is to build up the platform PisaFlix, a MySQL relational Database was chosen to store all the informations about movies, cinemas, users etc.

The Database has the following structure

### E-R DIAGRAM

Immagine che contiene screenshot

Descrizione generata automaticamente

**NOTE**: in the table *film/cinema\_has\_comment* the field *idComment* must be UNIQUE, the tables were made in order to make Hibernate work properly

Users instead can use a java application with a GUI for using all functionalities of the platform (register, see movies list etc…)

The client Application it’s made in Java using JavaFX framework for the GUI and Hibernate JPA for implementing data persistence

### GUI – MVC

The graphic user interface was build follow the software design pattern of Model View Controller

**Model** (PisaFlixServices)

The central component of the pattern. It is the application's dynamic data structure, independent of the user interface. It directly manages the data, logic and rules of the application.

The model is responsible for managing the data of the application. It receives user input from the controller.

**View** (FXML files)

All the graphic components (Pages, Buttons).

**Controller** (Contollers linked to FXML files)

Accepts input and converts it to commands for the model or view.

The controller responds to the user input and performs interactions on the data model objects. The controller receives the input, optionally validates it and then passes the input to the model.

Immagine che contiene screenshot

Descrizione generata automaticamente

## SOFTWARE Classes

In the next pages we will describe all classes presents in the application.

Let’s start with the main entities, but since they are self explanatory we will not see them in details.

Immagine che contiene testo, mappa

Descrizione generata automaticamente

The only interestring thing is that inside of java file there are indications for Hibernate in order to perform Queries on the database, let’s see an example

### E:\Stemma_unipi.png

**@Entity**

**@Table**(name = "Film")

**public** **class** **Film** **implements** Serializable {

**private** **static** **final** **long** serialVersionUID = **1L**;

**@Id**

**@GeneratedValue**(strategy = GenerationType.IDENTITY)

**@Basic**(optional = **false**)

**@Column**(name = "idFilm")

**private** Integer idFilm;

**@Basic**(optional = **false**)

**@Column**(name = "title")

**private** String title;

**@Basic**(optional = **false**)

**@Column**(name = "publicationDate")

**@Temporal**(TemporalType.DATE)

**private** Date publicationDate;

**@Lob**

**@Column**(name = "description")

**private** String description;

**@JoinTable**(name = "Favorite\_Film", joinColumns = {

**@JoinColumn**(name = "idFilm", referencedColumnName = "idFilm")}, inverseJoinColumns = {

**@JoinColumn**(name = "idUser", referencedColumnName = "idUser")})

**@ManyToMany**(fetch = FetchType.EAGER)

**private** Set<User> userSet = **new** LinkedHashSet<>();

**@ManyToMany**(mappedBy = "filmSet", fetch = FetchType.EAGER, cascade = CascadeType.ALL)

**@OrderBy**

**private** Set<Comment> commentSet = **new** LinkedHashSet<>();

**@OneToMany**(mappedBy = "idFilm", fetch = FetchType.EAGER, cascade = CascadeType.ALL)

**private** Set<Projection> projectionSet = **new** LinkedHashSet<>();

//GETTERS AND SETTERS

}