University of Pisa

SCUOLA DI INGEGNERIA
Corso di Laurea in Artificial Intelligence and Data Engineering

Task1 documentation

Candidati
Alice Nannini
Giacomo Mantovani
Marco Parola
Stefano Poleggi

Relatore
Prof. Pietro Ducange

Anno Accademico 2019–2020

Contents

1 Introduction **1**

2 Analysis and workflow **3**

2.1 Requirements . . . 3

2.1.1 Functional requirement . . . 3

2.1.2 Non-functional requirements . . . 3

2.2 Use Cases . . . 4

2.2.1 Use Cases Description . . . 4

2.3 Analysis of entities . . . 6

3 Design **7**

3.1 Software architecture . . . 7

3.2 Populating the database . . . 7

3.3 Analytics and statistics . . . 9

3.3.1 View the top production houses . . . 9

3.3.2 Top rated movies by year and country . . . 9

3.4 MongoDb index . . . 9

4 Implementation **10**

4.1 Used technologies . . . 10

4.2 Replica setup . . . 10

4.3 Java class description . . . 11

4.4 GUI . . . 11

5 User Manual **12**

1 Introduction

The **Cine-Valutami** application offers a search and information service in the field of cinema. When the application starts, the system requires authentication to use the service. The logged-in user can perform a search by entering the first characters of a movie title in the search bar, obtaining a list of 10 movies in the database. After that you can select one of the proposed titles or carry out a more in-depth search, adding characters. Selecting a row, the system allows you to view more information in the right section, including cover, title, director and ratings. The user can leave a mark from 1 to 5 for the selected movie. There is also a module for the system administrator: when he logs in, he's redirected to a different activity than the user's one. In this page, he can view some statistics linked to the movies and the searches carried out by the users of the application, such as the ranking of 10 most voted films or the 10 most sought after films. Moreover, the system admin can add new movies to the application database or delete those already present, by searching by title.

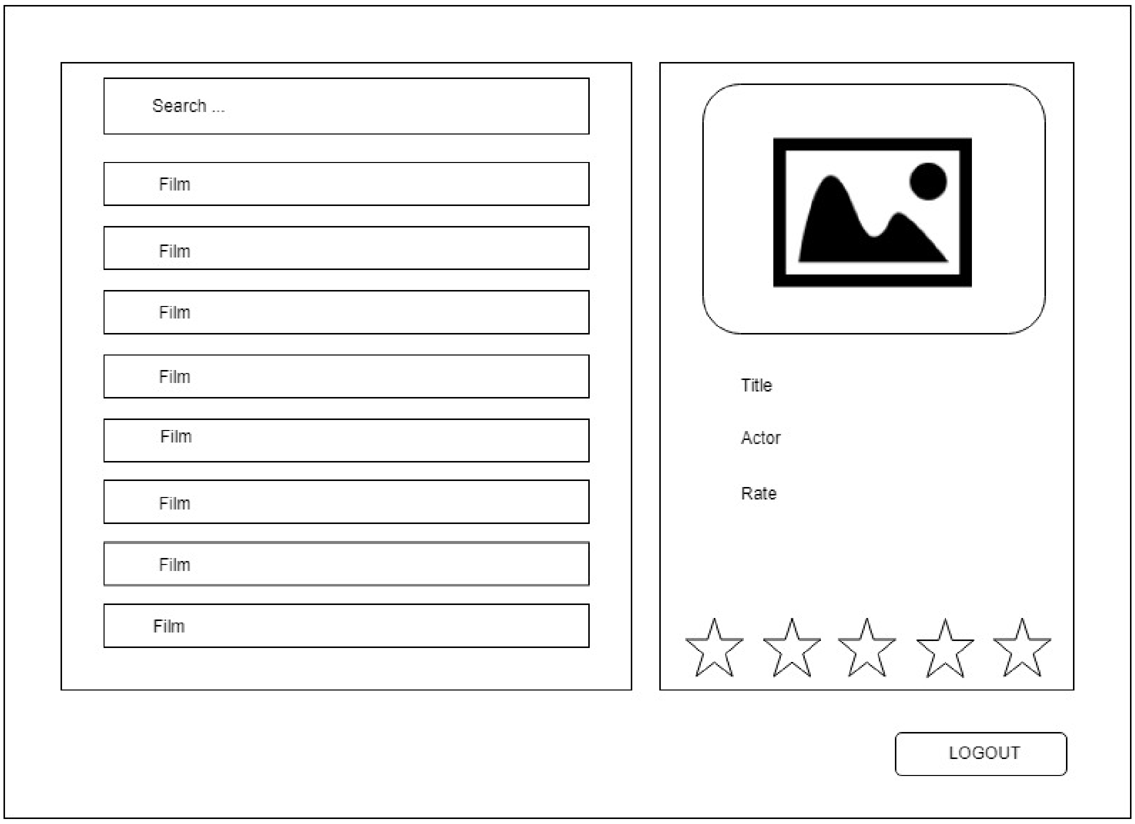


Figure 1: User Mockup

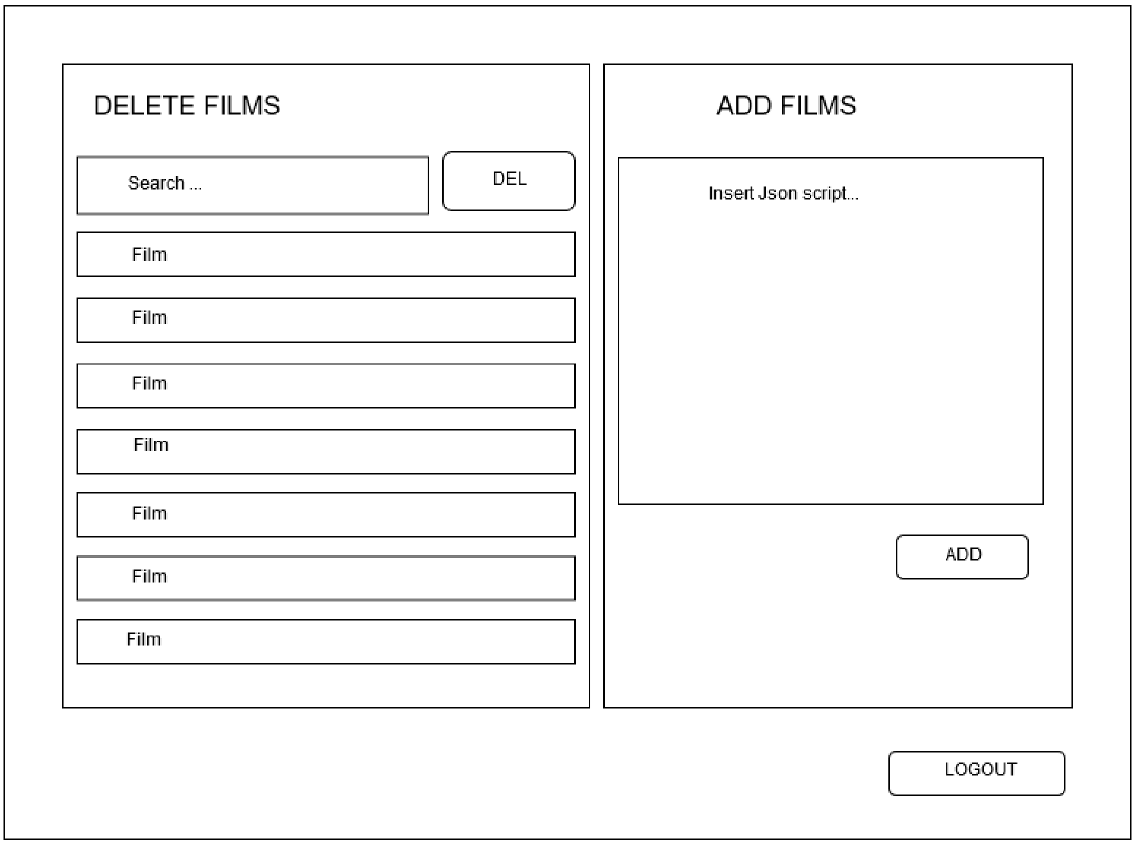


Figure 2: Admin Mockup

2 Analysis and workflow

2.1 Requirements

2.1.1 Functional requirement

The system has to allow the user to carry out basic functions such as:

- To sign up into the system.
- To login to the system.
- To search for a movie.
- To vote a movie.
- To view a list of the top rated movies, both in general and filtered by a selected country.
- To view a list of the top production houses.
- To view the top countries in making movies, filtered by a selected year.
- To view the most active users in the application, i.e. the users that have given most votes.

The system has to allow the administrator to carry out basic functions such as:

- To login to the system.
- To add a movie.
- To update a movie.
- To delete a movie.

2.1.2 Non-functional requirements

- Usability, ease of use and intuitiveness of the application by the user.
- Avaliablility, with the service guaranteed h24, using replicas.
- The system should support simultaneous users.
- The system should provide access to the database with a few seconds of latency.
- Enforced consistency.

2.2 Use Cases

Actors

- User: this actor represents a user of the application
- Admin: this actor represents the administrator of the application

2.2.1 Use Cases Description

Event	**UseCase**	**Actor(s)**	**Description**
Log in, Log out	Login, Logout	Admin, User	The user logs in/out the application.
Display all the Films	Browse, Find, Display Films	User, Admin	The user chooses that he wants to view the list of Films. The system browses the data on the db and returns them on the interface.
View Statistics	View Top Rated Films, View Top Productions, View Top Film-maker Countries, View Most Active Users	Admin	The Admin clicks on the button to view the statistics. The system browses on the db the informations used in the calculation and display the result.
Add a film	Add Film	Admin	The admin submits the Film information. The system updates the db and the interface.
Update a film	Update Film	Admin	The admin selects the film and commits the new informations. The system updates the db and the interface.
Delete a film	Delete Film	Admin	The admin selects the film and submits the delete. The system updates the db and the interface.
View the film informations	Select Film, Display Film Info	User, Admin	The user selects the film. The system shows the film informations on the interface.
Vote a film	Vote Film	User, Admin	The user submits the vote on a selected film. The system updates the db and the interface.

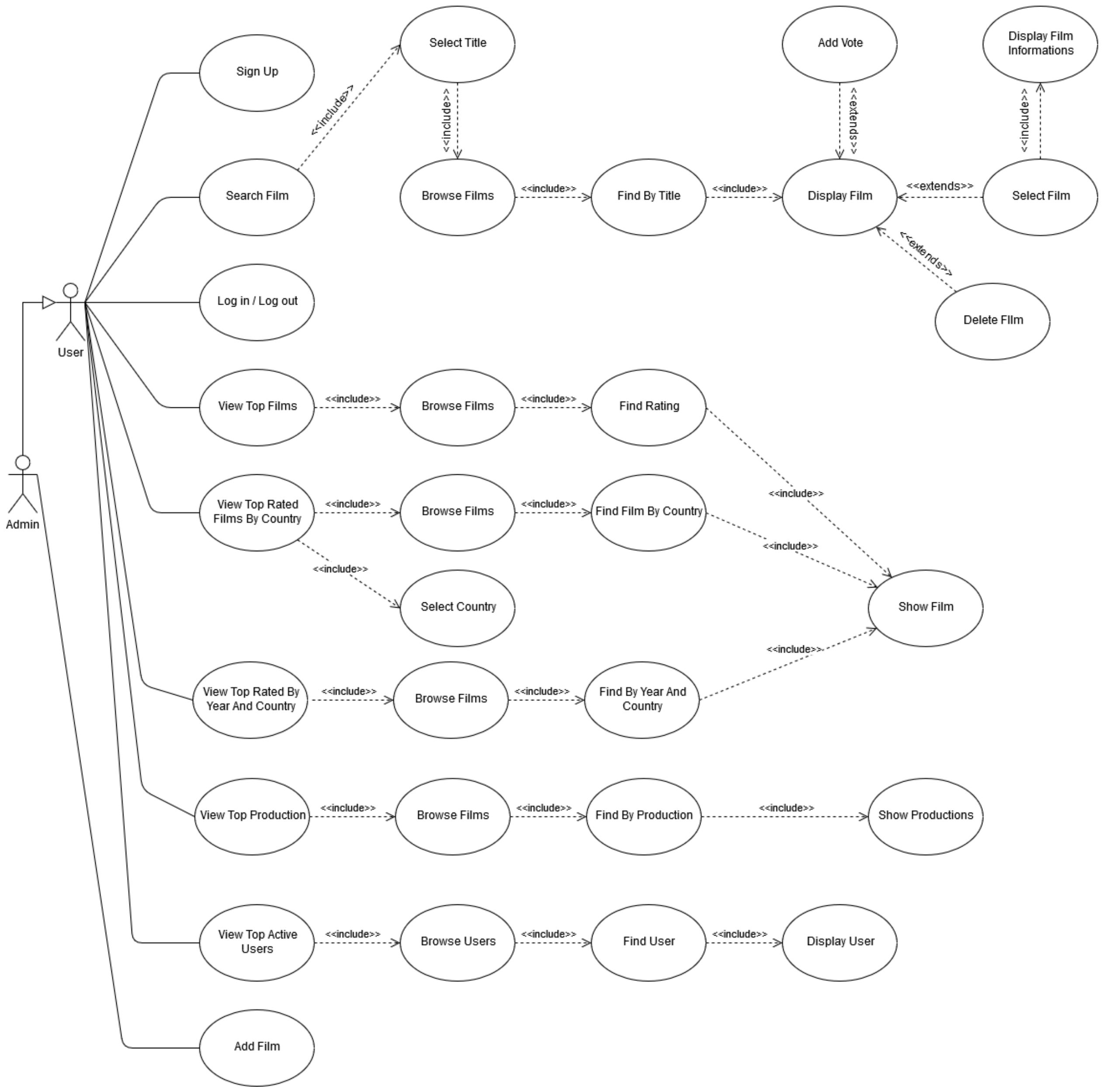


Figure 3: Use cases diagram

2.3 Analysis of entities

This diagram represents the main entities of the application and the relations between them.

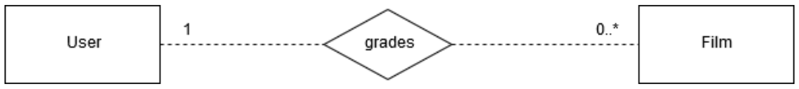


Figure 4: UML analysis diagram

3 Design

3.1 Software architecture

The application is designed over 2 different layers, see figure 5:

- Front-end
- Back-end

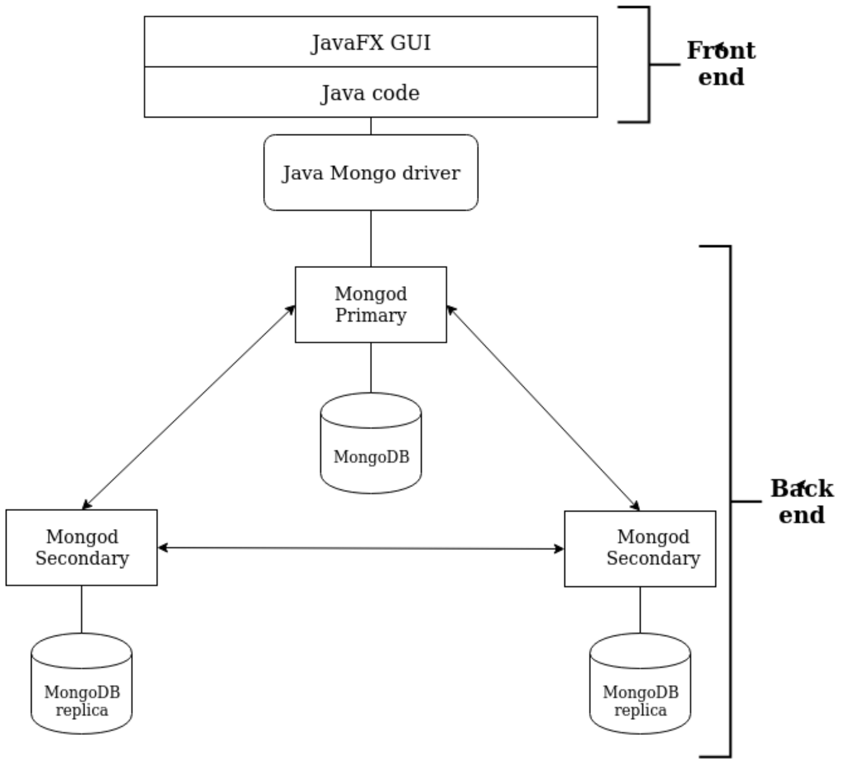
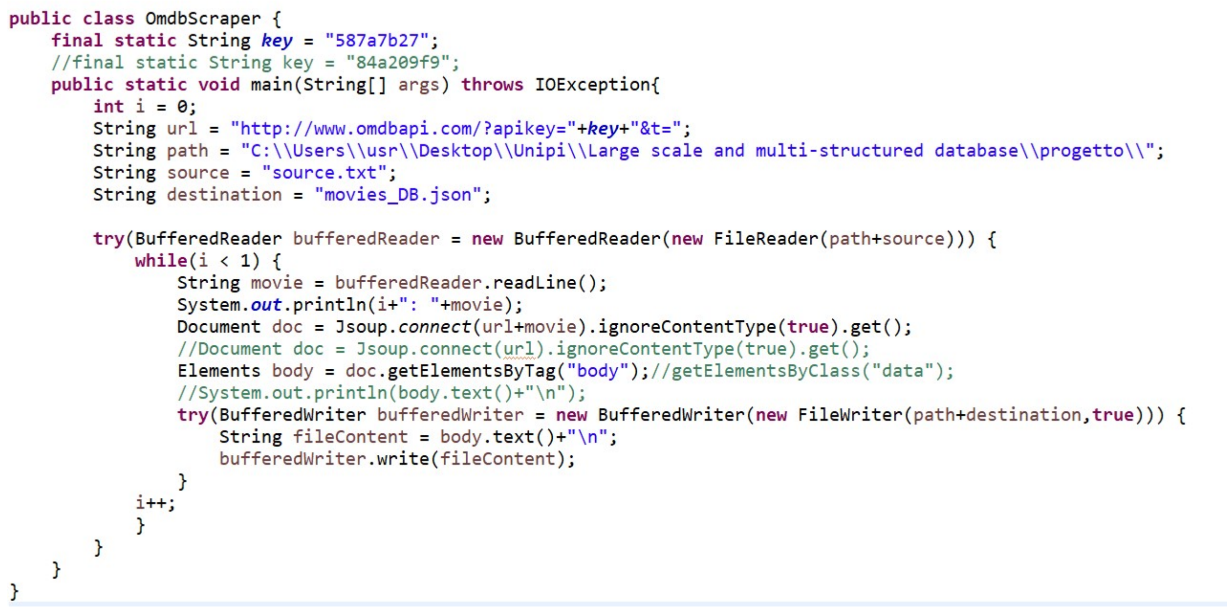


Figure 5: Software architecture diagram

3.2 Populating the database

The dataset used in the MoviesEvaluation application was created by scraping the Open Movie Database, using their API (http://www.omdbapi.com/). Before started building up the dataset using the API, it was necessary to obtain a list of movie titles, to be passed to the API; the list of titles was built up by scraping Wikipedia page. Once the first 9000 titles' list was ready, the only thing left to do was to obtain the key requested by the OMDb API to download the films' information. The keys provided by the database allow users to download 1000 film's information per day. Once the movies' list and the keys were ready, we could start to use our scraping application.

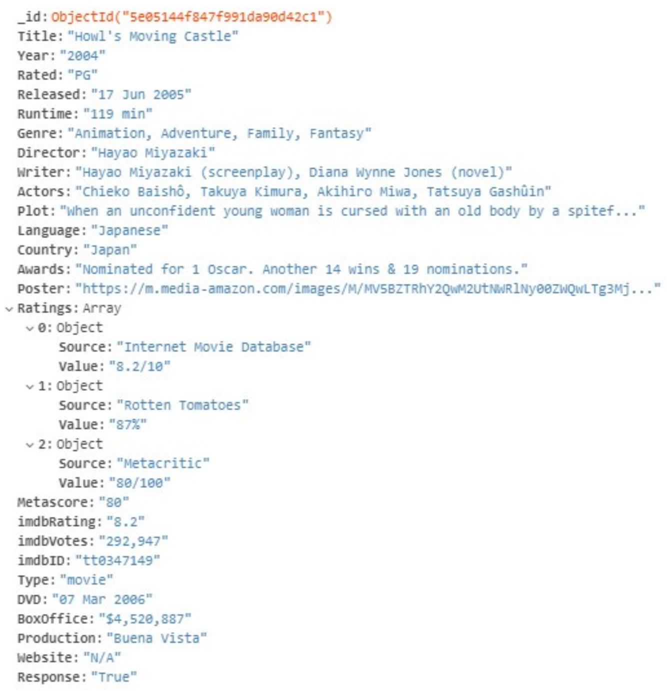


```java
public class OmdbScraper {
    final static String key = "587a7b27";
    //final static String key = "84a209f9";
    public static void main(String[] args) throws IOException{
        int i = 0;
        String url = "http://www.omdbapi.com/?apikey="+key+"&t=";
        String path = "C:\\Users\\usr\\Desktop\\Unipi\\Large scale and multi-structured database\\progetto\\";
        String source = "source.txt";
        String destination = "movies_DB.json";

        try(BufferedReader bufferedReader = new BufferedReader(new FileReader(path+source))) {
            while(i < 1) {
                String movie = bufferedReader.readLine();
                System.out.println(i+": "+movie);
                Document doc = Jsoup.connect(url+movie).ignoreContentType(true).get();
                //Document doc = Jsoup.connect(url).ignoreContentType(true).get();
                Elements body = doc.getElementsByTag("body");//getElementsByClass("data");
                //System.out.println(body.text()+"\n");
                try(BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(path+destination,true))) {
                    String fileContent = body.text()+"\n";
                    bufferedWriter.write(fileContent);
                }
            i++;
            }
        }
    }
}
```

Figure 6: Java class of OMDBScraper

Using our "OmdbScraper" application, we managed to obtain the collection of movies' information in a JSON format.
The format of our JSON elements are the following:



```
 _id: ObjectId("5e05144f847f991da90d42c1")
 Title: "Howl's Moving Castle"
 Year: "2004"
 Rated: "PG"
 Released: "17 Jun 2005"
 Runtime: "119 min"
 Genre: "Animation, Adventure, Family, Fantasy"
 Director: "Hayao Miyazaki"
 Writer: "Hayao Miyazaki (screenplay), Diana Wynne Jones (novel)"
 Actors: "Chieko Baishô, Takuya Kimura, Akihiro Miwa, Tatsuya Gashûin"
 Plot: "When an unconfident young woman is cursed with an old body by a spitef..."
 Language: "Japanese"
 Country: "Japan"
 Awards: "Nominated for 1 Oscar. Another 14 wins & 19 nominations."
 Poster: "https://m.media-amazon.com/images/M/MV5BZTRhY2QwM2UtNWRlNy00ZWQwLTg3Mj..."
⌄Ratings: Array
  ⌄0: Object
      Source: "Internet Movie Database"
      Value: "8.2/10"
  ⌄1: Object
      Source: "Rotten Tomatoes"
      Value: "87%"
  ⌄2: Object
      Source: "Metacritic"
      Value: "80/100"
 Metascore: "80"
 imdbRating: "8.2"
 imdbVotes: "292,947"
 imdbID: "tt0347149"
 Type: "movie"
 DVD: "07 Mar 2006"
 BoxOffice: "$4,520,887"
 Production: "Buena Vista"
 Website: "N/A"
 Response: "True"
```

Figure 7: Example of JSON document

We integrated our movies dataset with a plot dataset, found on kaggle (wiki_plot). In order to do

so, we searched for matching between the plot's movie title and the movie title in our collection; once the matching was found, the plot field in the original element was replaced with the data presents in the plot dataset.
Moreover we manage some data types, because scraping OpenMovie platform, we obtain every fields in a string format, but to compute the analytics operations, describes on functional requirements section, we need to have some fields in a number format. We run the following javascript code on mongo shell, to perform the conversion:

```
db.collection.find().foerEach( function(x){
        x.ratingImdb = parseFloat( x.imdbRating);
        db.collection.save(x);
});
```

3.3 Analytics and statistics

3.3.1 View the top production houses

This analytic is performed to display all production houses.

3.3.2 Top rated movies by year and country

3.4 MongoDb index

In order to speeding up some operations, we introduce some indexes ...

4 Implementation

4.1 Used technologies

The application is developed in java programming language, version 11.0.4, and in JavaFX system to create the GUI, version 11, so it should run on each platform in which JVM is installed, but the application is tested and guardantee on Ubuntu 16 and Window OS. Moreover Maven is used to build and mantain the project, version 3.8.0.
The java driver for mongo manage the comunication between client application layer and mongo backend layer, version 3.11.2.
For the backend layer it is used MongoDB, version 4.2.
So this application is tested using these technologies, considering these particular versions: for other versions the correct execution isn't guaranteed .

4.2 Replica setup

The following code shows how it is realized the architecture in which 3 mongo server are running in back end layer:

```
# ------ PREPARAZIONE CARTELLE PER LOG E PER I DB ------
sudo mkdir -p /srv/mongodb/rs0-0 /srv/mongodb/rs0-1 /srv/mongodb/rs0-2
sudo mkdir -p /var/log/mongodb/rs0-0 /var/log/mongodb/rs0-1 /var/log/mongodb/rs0-2

# ------ RUN 3 ISTANCES OF MONGOD ------
sudo mongod --port 27017 --dbpath /srv/mongodb/rs0-0 --replSet rs0 --oplogSize 128
--logpath /var/log/mongodb/rs0-0/server.log --fork
sudo mongod --port 27018 --dbpath /srv/mongodb/rs0-1 --replSet rs0 --oplogSize 128
--logpath /var/log/mongodb/rs0-1/server.log --fork
sudo mongod --port 27019 --dbpath /srv/mongodb/rs0-2 --replSet rs0 --oplogSize 128
--logpath /var/log/mongodb/rs0-2/server.log --fork

# ------ CHECK LISTENING PROCESS ------
netstat -tulpn

# ------ CONNECT TO PRIMARY MONGOD ISTANCE ------
mongo --port 27017

/* JS SCRIPT */
var rsconf = {
    _id: "rs0",
    members: [
                {_id: 0, host: "127.0.0.1:27017"},
                {_id: 1, host: "127.0.0.1:27018"},
                {_id: 2, host: "127.0.0.1:27019"}
            ]
};
rs.initiate( rsconf );
rs.conf();
rs.status();

# ------ KILL PROCESS USING PORT ------
sudo fuser -k 27017/tcp
sudo fuser -k 27018/tcp
sudo fuser -k 27019/tcp

# ------ ELIMINA CARTELLE -------
sudo rm -r /srv/mongodb/rs0-0 /srv/mongodb/rs0-1 /srv/mongodb/rs0-2
sudo rm -r /var/log/mongodb/rs0-0 /var/log/mongodb/rs0-1 /var/log/mongodb/rs0-2
```

4.3 Java class description

4.4 GUI

5 User Manual