UNIVERSITY OF PISA

School of Engineering

Master of Science in Computer Engineering

Large-Scale and Multi-Structured Databases Course

TASK 3 PROJECT

CINEMASPACE RECOMMENDATION SYSTEM

WORKGROUP:

Marie Giannoni

Shirley Caillere

Francisco Payés Erroa

Diego Casu

ACADEMIC YEAR 2019/2020

INDEX

1 APPLICATION DESCRIPTION

To improve the user experience offered by CinemaSpace[1], the introduction of a recommendation system, which will be able to suggest films to the subscribed users, is needed.
Since the original archive of the application is optimized for storing and retrieving large collections of data, it is necessary to build an ad hoc parallel system that can store, identify and quickly analyse relationships between users and rated films. As a result, the registered user will be able to access a new section in her personal page, where she will find:

- a selection of recommended films based on genres, identified starting from the recent positive ratings she gave;
- a selection of recommended films based on the ones rated by other users with common interests.

The original application and the new extension will operate on different views and details of the same data, coordinating in order to use different archives and preserve the overall consistency of the dataset.

[1] CinemaSpace is the application developed for the Task 2 project.

2 MAIN ACTORS AND REQUIREMENTS

2.1 Main actors

- The client user
- The administrator of the community.

2.2 Requirements

The following requirements are in addition to the ones defined and implemented in CinemaSpace.

2.2.1 Functional requirements

The system must allow the registered client user:

- to see on her personal page a selection of recommended films based on genres. The latter are identified taking into account her last positive ratings — the ones with a value equal or higher than 2.5 — and the associated film genres.
- to see on her personal page a selection of recommended films based on the ones rated by other users with common interests. In this case, "users with common interests" refers to a set of users who rated a similar list of films.

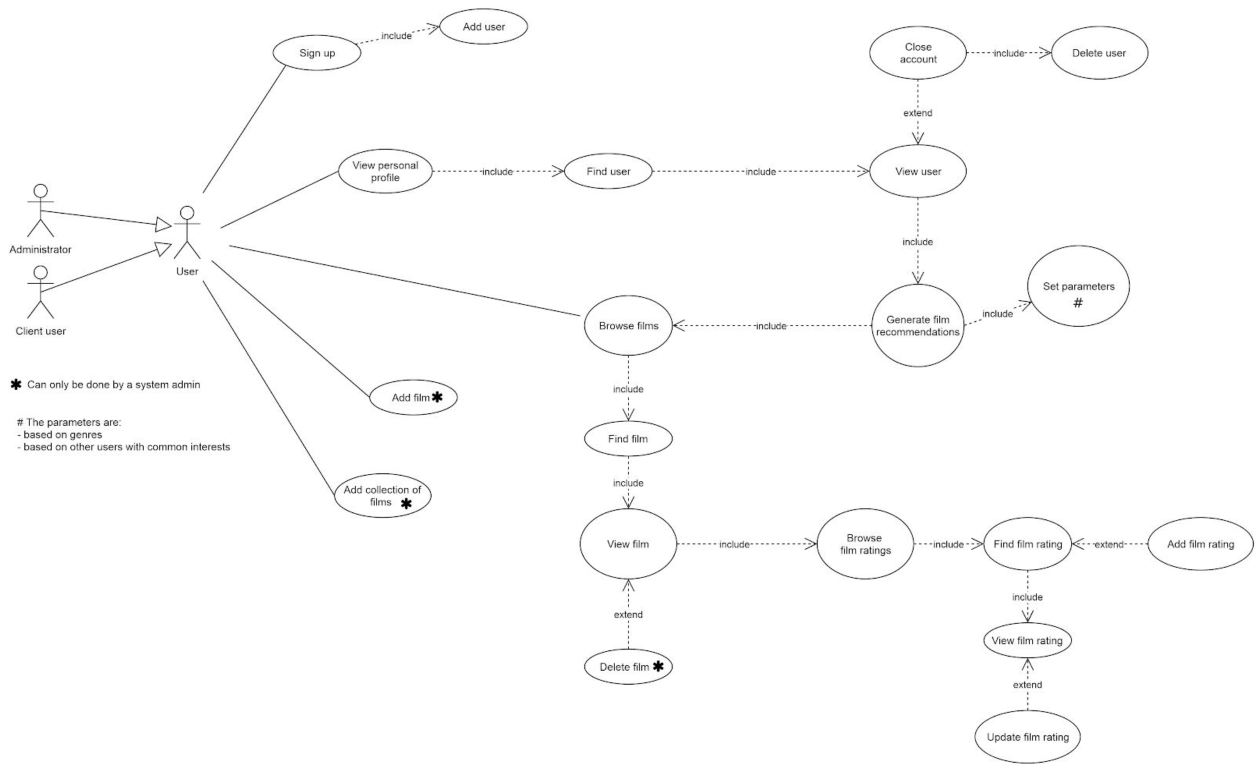
2.2.2 Non-functional requirements

The overall system must ensure:

- quick response to the recommendation requests;
- the duplication of only the strictly necessary information inside the recommendation system's archive;
- the consistency of the databases, where each significant insert, update or delete operation on the main archive must be propagated to the recommendation system's archive to be considered as completed. In case of failure, a rollback of the operation must be performed.

3 USE CASES

The use cases that involve an insert, update or delete operation are the same of the original CinemaSpace application, but they were replicated to underline the fact that their effects must be properly propagated in the recommendation system's archive.



See film recommendations based on genres

Brief description: A user wants to see a selection of films recommended by the system.

Preconditions: The user must be logged in and must have rated at least one film.

Basic flow of events:
1. The user enters her profile page.
2. The system generates a list of suggested films based on the genres retrieved from her recent positive ratings.
3. The system displays the films in the associated section of the page.

Extensions:

2a. The system fails to generate the suggested films
- The system logs information about the system's error and no modification is made to the current state of the application.

Post-conditions: The suggestion section is populated by the identified films and can be browsed by the user.

See film suggestions based on other users with common interests

Brief description: A user wants to see a selection of films recommended by the system.

Preconditions: The user must be logged in and must have rated at least one film.

Basic flow of events:
4. The user enters her profile page.
5. The system generates a list of suggested films based on the films rated by other users with common interests.
6. The system displays the films in the associated section of the page.

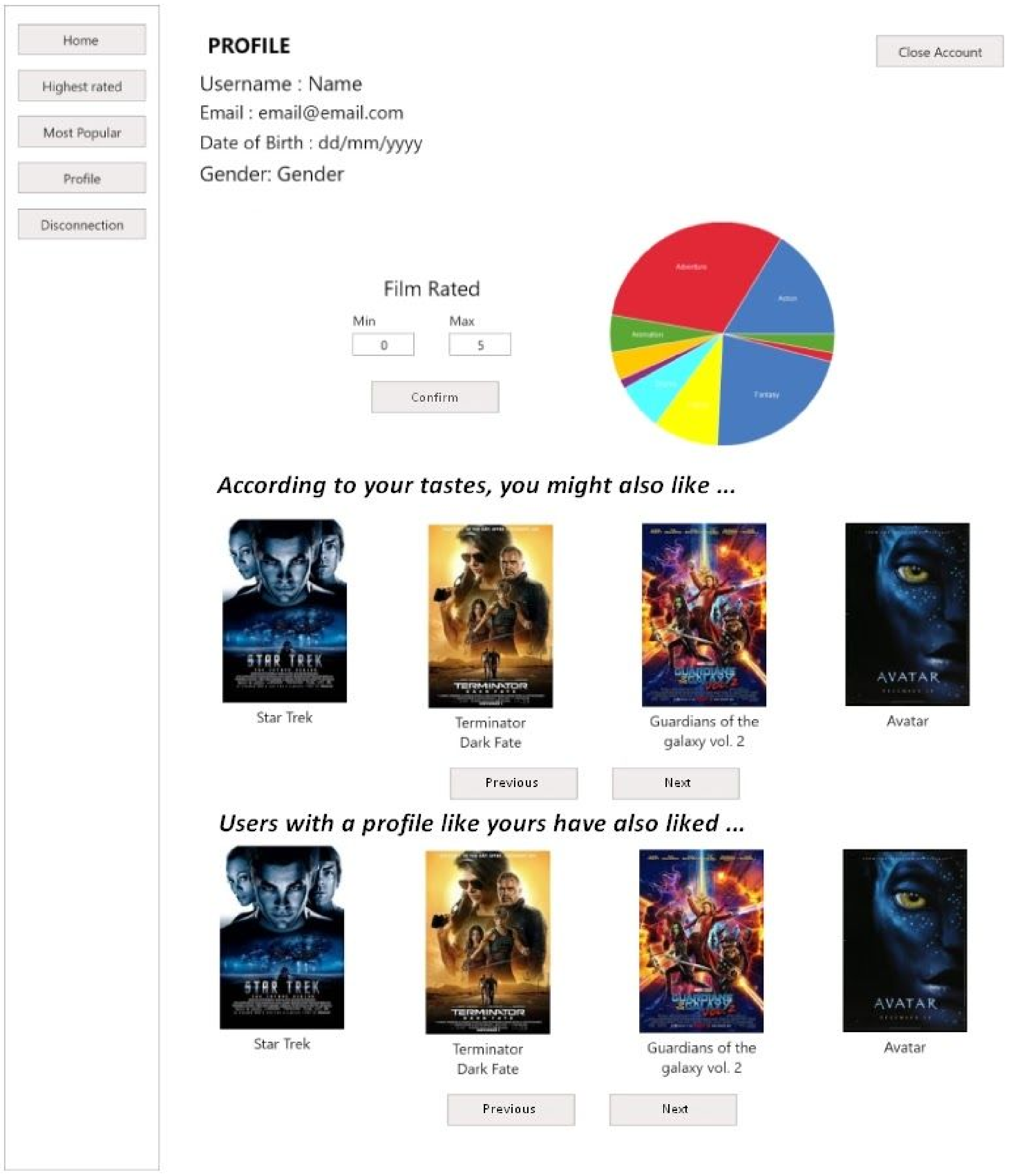
Extensions:

2a. The system fails to generate the suggested films
- The system logs information about the system's error and no modification is made to the current state of the application.

Post-conditions: The suggestion section is populated by the identified films and can be browsed by the user.

4 INTERFACE MOCKUP

Personal profile page (user)



Username : Name
Email : email@email.com
Date of Birth : dd/mm/yyyy
Gender: Gender

According to your tastes, you might also like ...

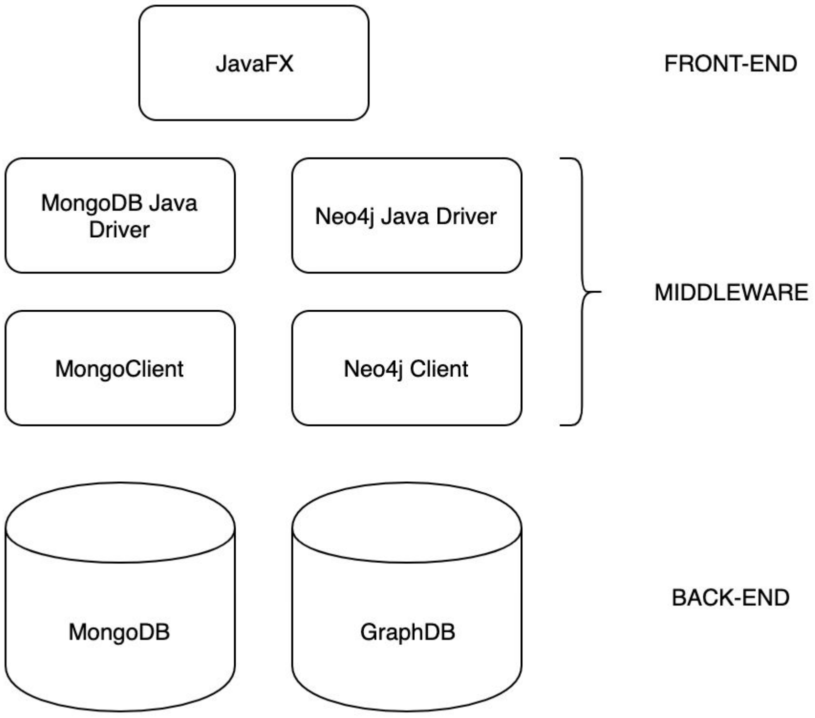
Users with a profile like yours have also liked ...

5 PLATFORM ARCHITECTURE

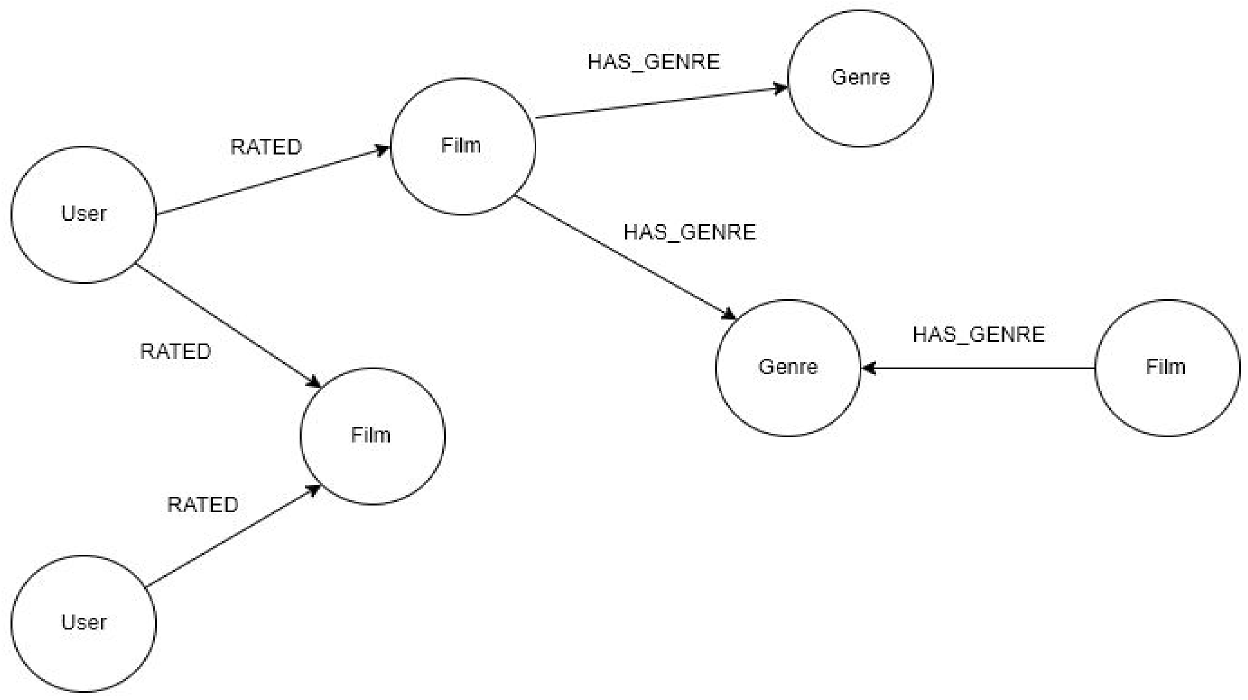
5.1 Overview

Client user and administrator will access the same application through a client-server paradigm, with different privileges. The client application is built in Java, using JavaFX and FXML to build the graphical interface, and it will connect:

- to the MongoDB server to reply to requests about the films, ratings and users;
- to the Neo4j server to elaborate the film recommendations.



5.2 Database structure



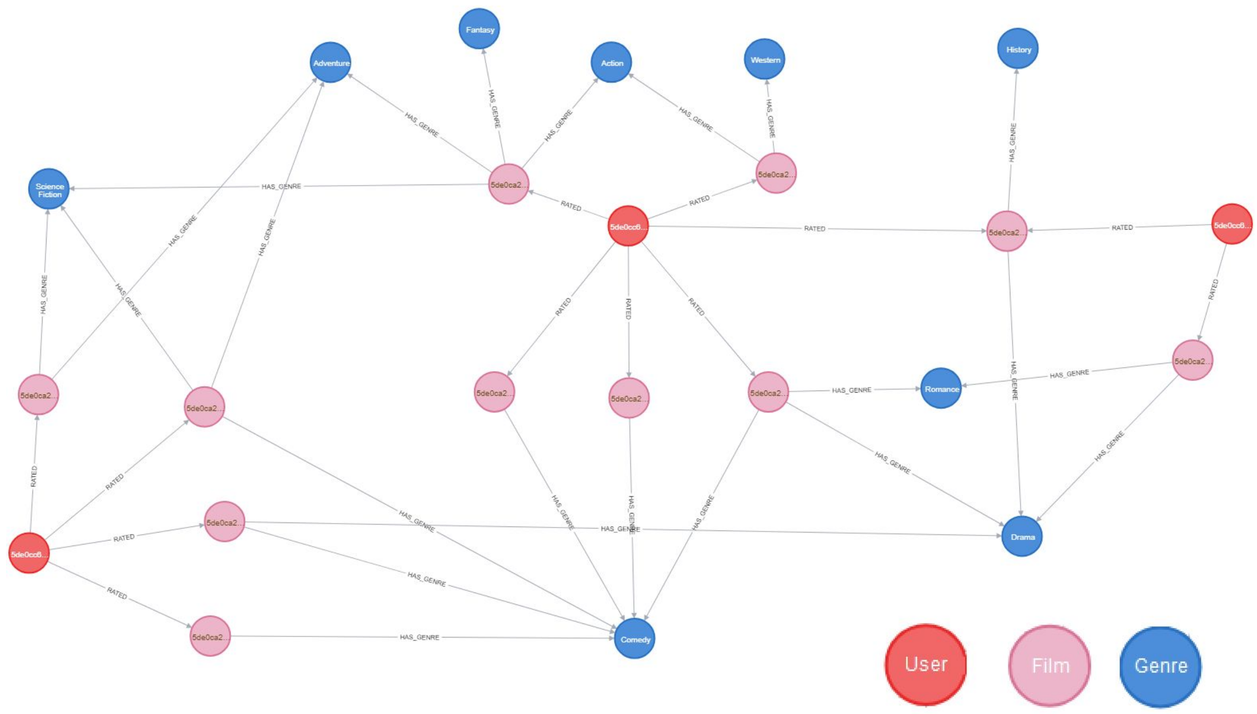
The graph recommendation database is composed of three types of nodes:

- User, with the property *objectId*, which is the hexadecimal string representation of the associated user id stored in the main MongoDB archive;
- Film, with the property *objectId*, which is the hexadecimal string representation of the associated film id stored in the main MongoDB archive;
- Genre, with the property *name*, which is the string representing the name of the film genre (ex. “Action”).

The relationships that connect the nodes are:

- RATED, which connects a User to a Film, with the properties *rating* and *timestamp*. The former represents the value of the rating, while the latter represents a timestamp in the Unix time format (milliseconds);
- HAS_GENRE, which connects a Film to a Genre, without properties.

An example of a graph subset is the following:



5.3 Main queries and indexes

Generate film recommendations based on genres

The recommended films are selected starting from the last positive ratings of the user, i.e. the ratings with a value equal or higher than 2.5, and the associated films. The query returns a random list of films which share at least one genre with the previously retrieved ones and that the user didn't rate yet.

```
MATCH (user:User {objectId: "5de0cc65bab5a6cf3157658f"})-[rated:RATED]->(film:Film)
WHERE rated.rating >= 2.5
WITH rated, film, user
ORDER BY rated.timestamp DESC
LIMIT 3
MATCH (film)-[:HAS_GENRE]->(:Genre)<-[:HAS_GENRE]-(suggestedFilm:Film)
WHERE NOT ((user)-[:RATED]->(suggestedFilm))
RETURN DISTINCT suggestedFilm.objectId
ORDER BY rand()
LIMIT 20
```

The response time of the query can be improved defining an index on the objectId property of User:

```
CREATE INDEX ON :User(objectId)
```

Generate film recommendations based on other users with common interests

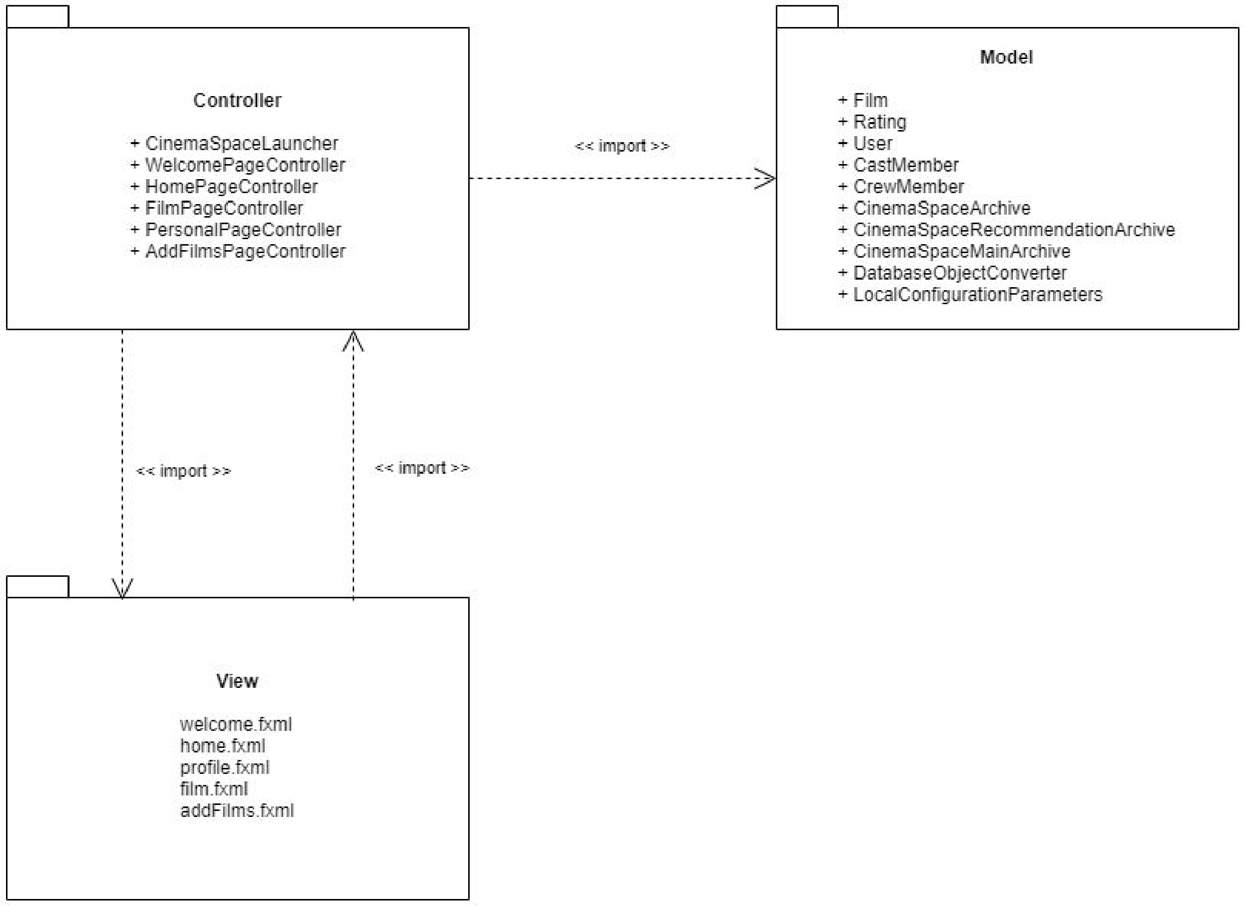
The recommended films are selected starting from a random subset of the users which have at least one rated film in common with the person that made the request; the subset is limited to 1000 users to make the following computations quicker. Then, the Jaccard similarity between the requesting user and each user of the identified subset is generated, taking into account the list of rated films of each one. Finally, starting from the users with the highest similarity coefficient and their positive ratings, a list of random films, that the user hasn't rated yet, is generated.

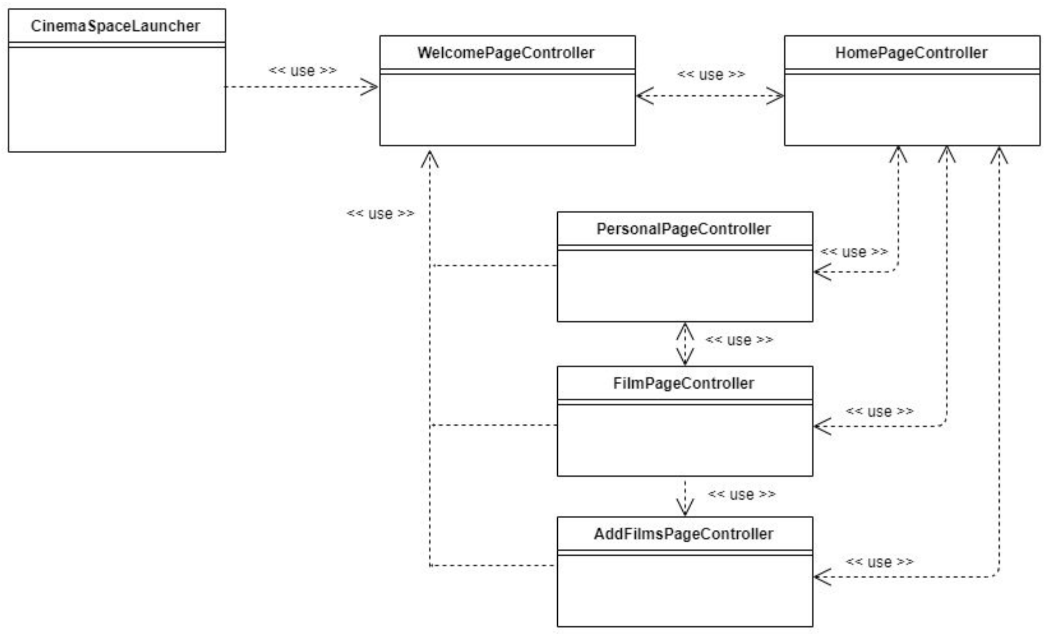
```
MATCH (user:User {objectId: "5de0cc65bab5a6cf3157658f"})
                    -[:RATED]->(:Film)<-[:RATED]-(otherUser:User)
WITH DISTINCT user, otherUser
ORDER BY rand()
LIMIT 1000
MATCH (user:User {objectId: "5de0cc65bab5a6cf3157658f"})-[:RATED]->(film:Film)
WITH user, collect(id(film)) as userFilms, otherUser
MATCH (otherUser:User)-[:RATED]->(film:Film)
WITH user, userFilms, otherUser, collect(id(film)) as otherUserFilms
WITH user, otherUser, algo.similarity.jaccard(userFilms, otherUserFilms) AS similarity
ORDER BY similarity DESC
LIMIT 5
MATCH (otherUser:User)-[rated:RATED]->(film:Film)
WHERE rated.rating >= 2.5 AND NOT((film)<-[:RATED]-(user:User))
RETURN DISTINCT film.objectId
ORDER BY rand()
LIMIT 20
```

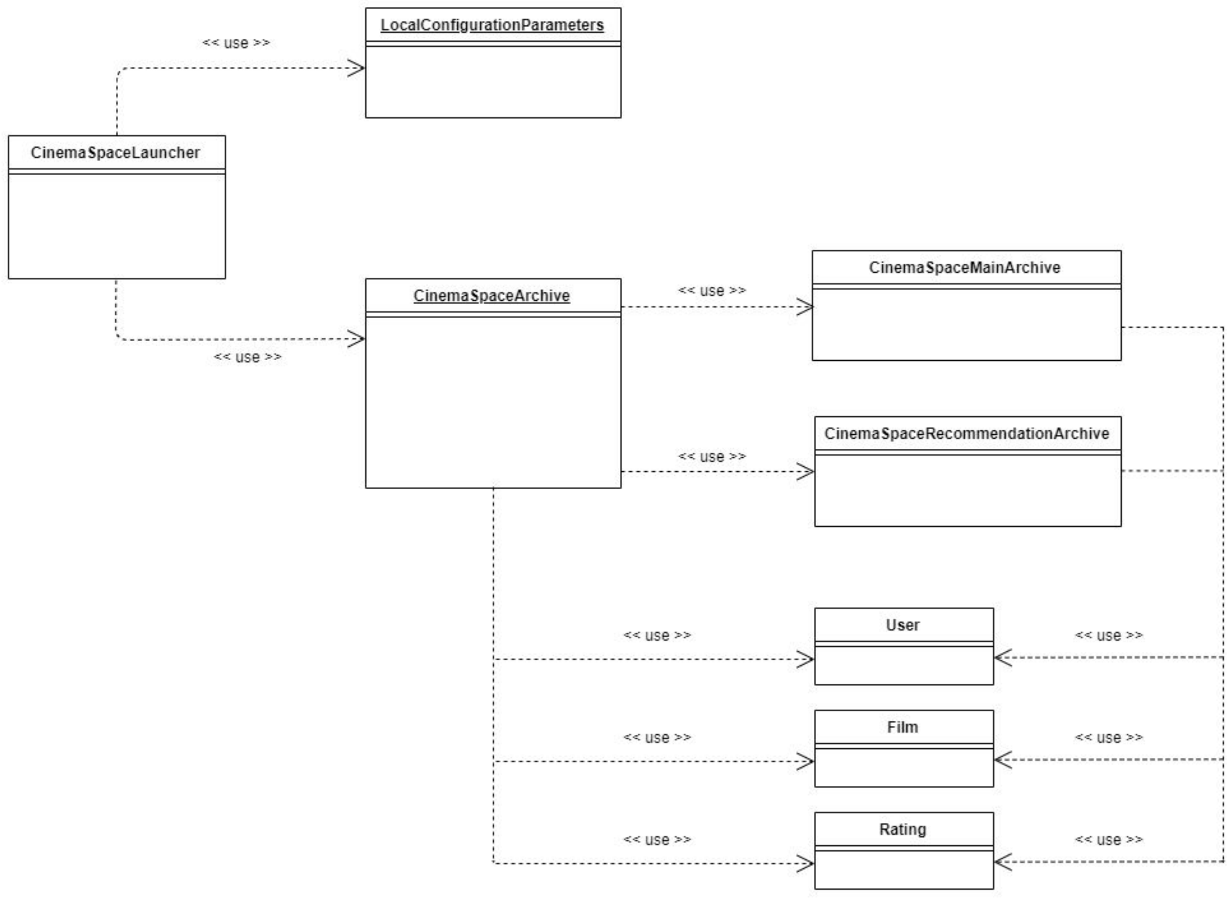
The query benefits automatically of the index previously defined on the objectId property of User.

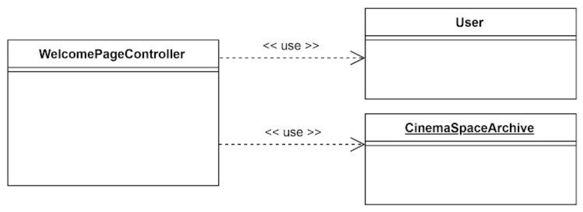
6 IMPLEMENTATION

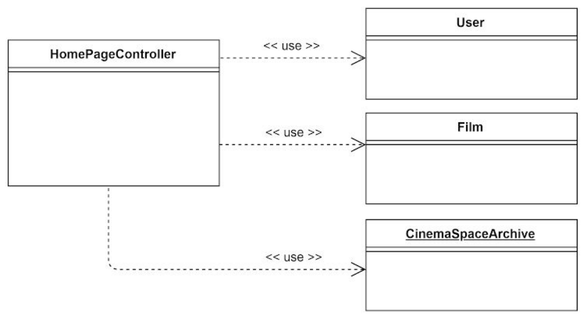
6.1 Class diagram

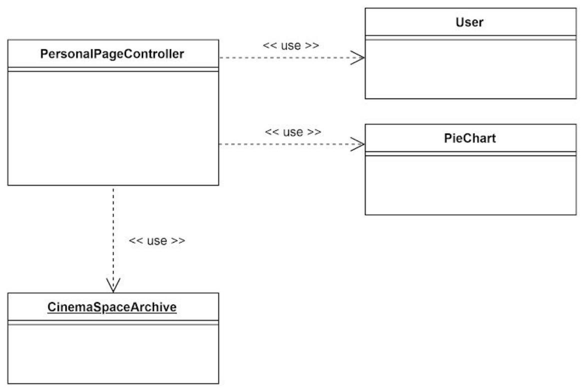


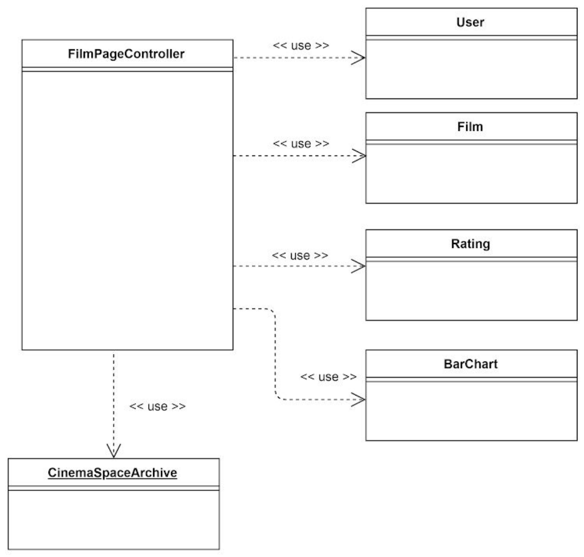

CinemaSpaceLauncher
WelcomePageController
HomePageController
PersonalPageController
FilmPageController
AddFilmsPageController
<< use >>

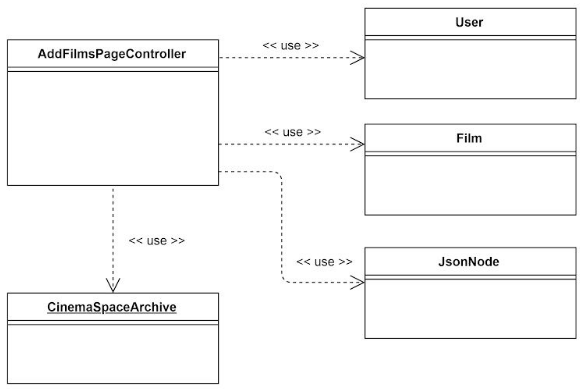

LocalConfigurationParameters
CinemaSpaceLauncher
CinemaSpaceArchive
CinemaSpaceMainArchive
CinemaSpaceRecommendationArchive
User
Film
Rating
<< use >>

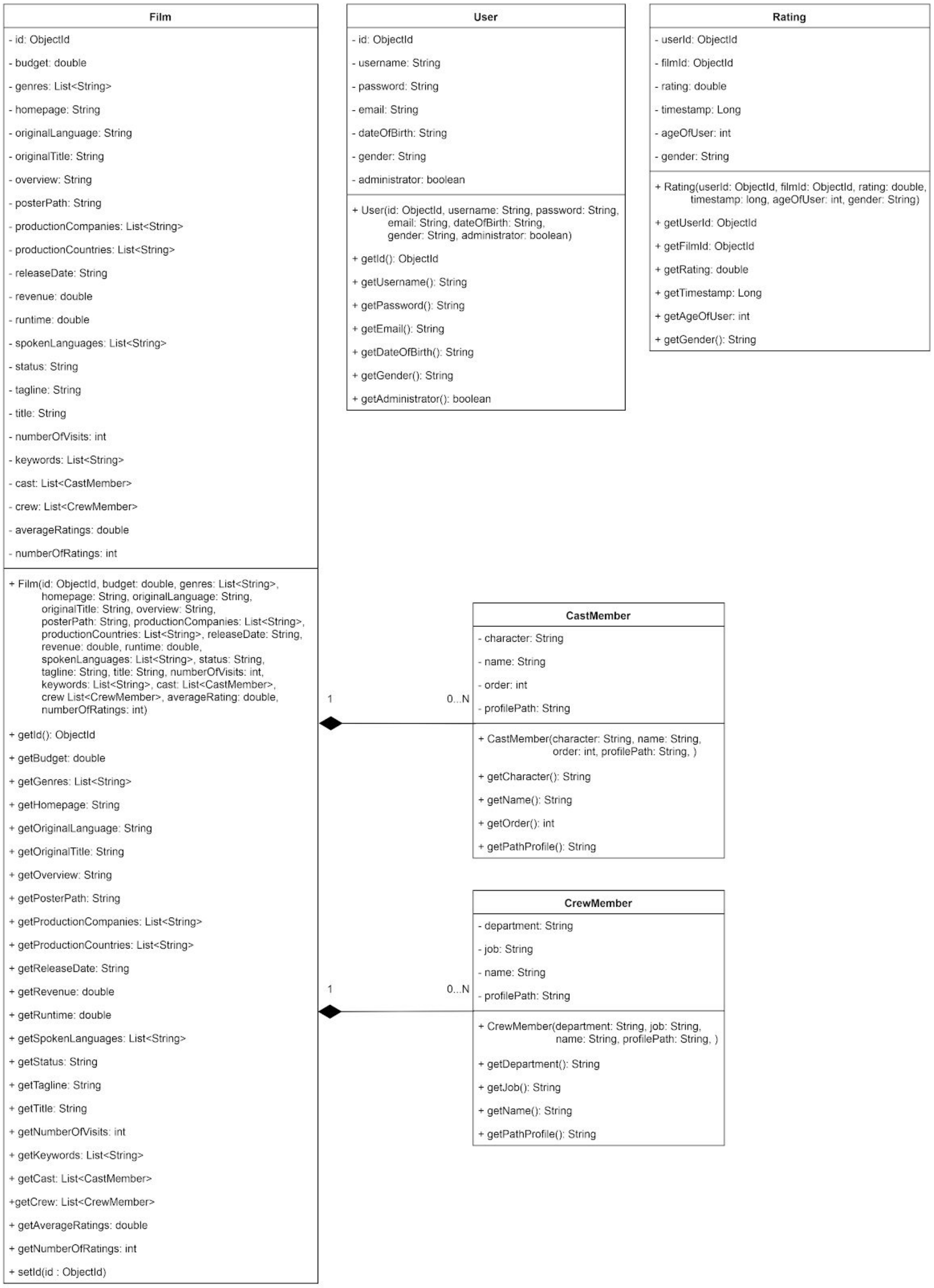

WelcomePageController
<< use >>
User
<< use >>
CinemaSpaceArchive

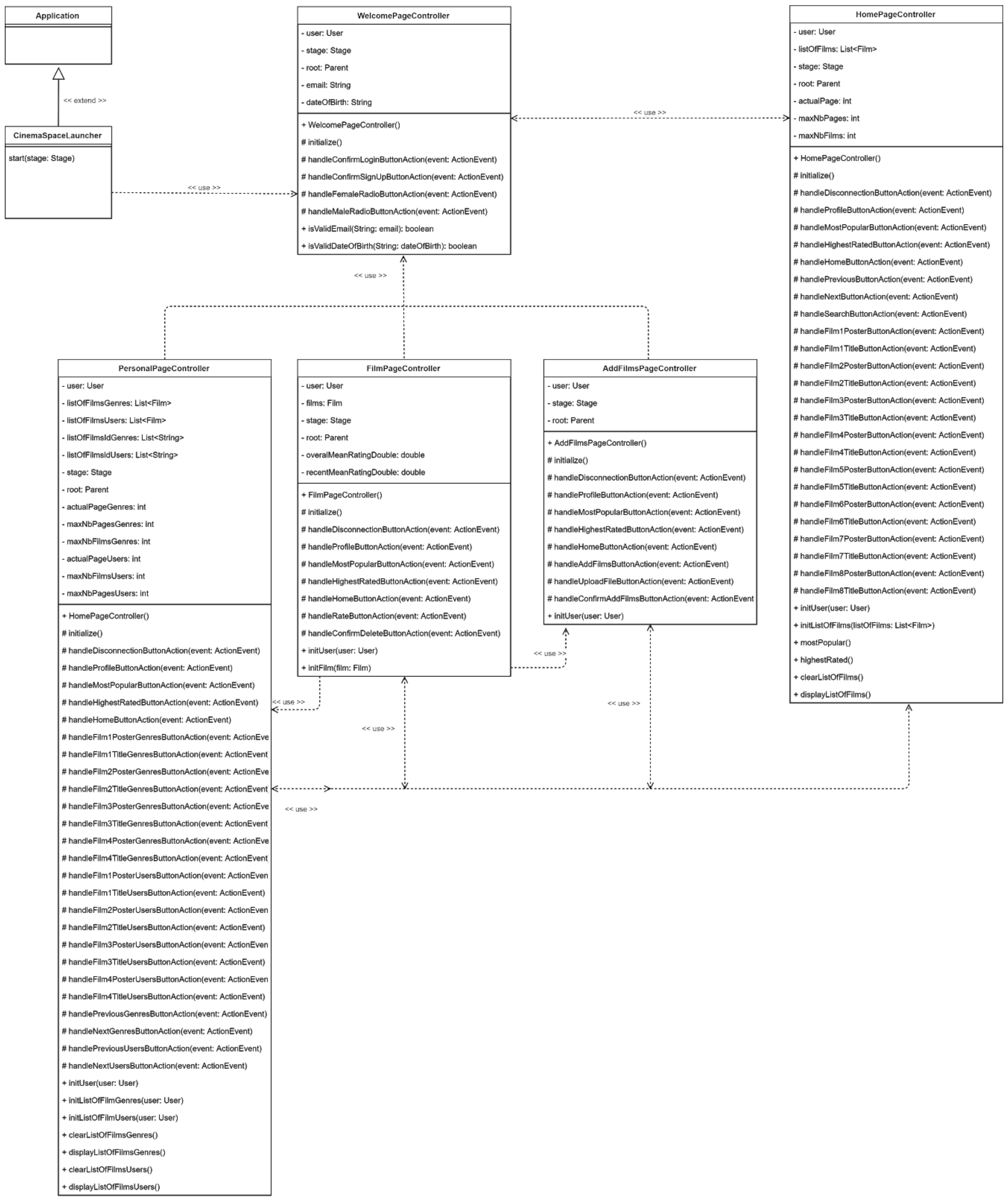

HomePageController
<< use >>
User
<< use >>
Film
<< use >>
CinemaSpaceArchive

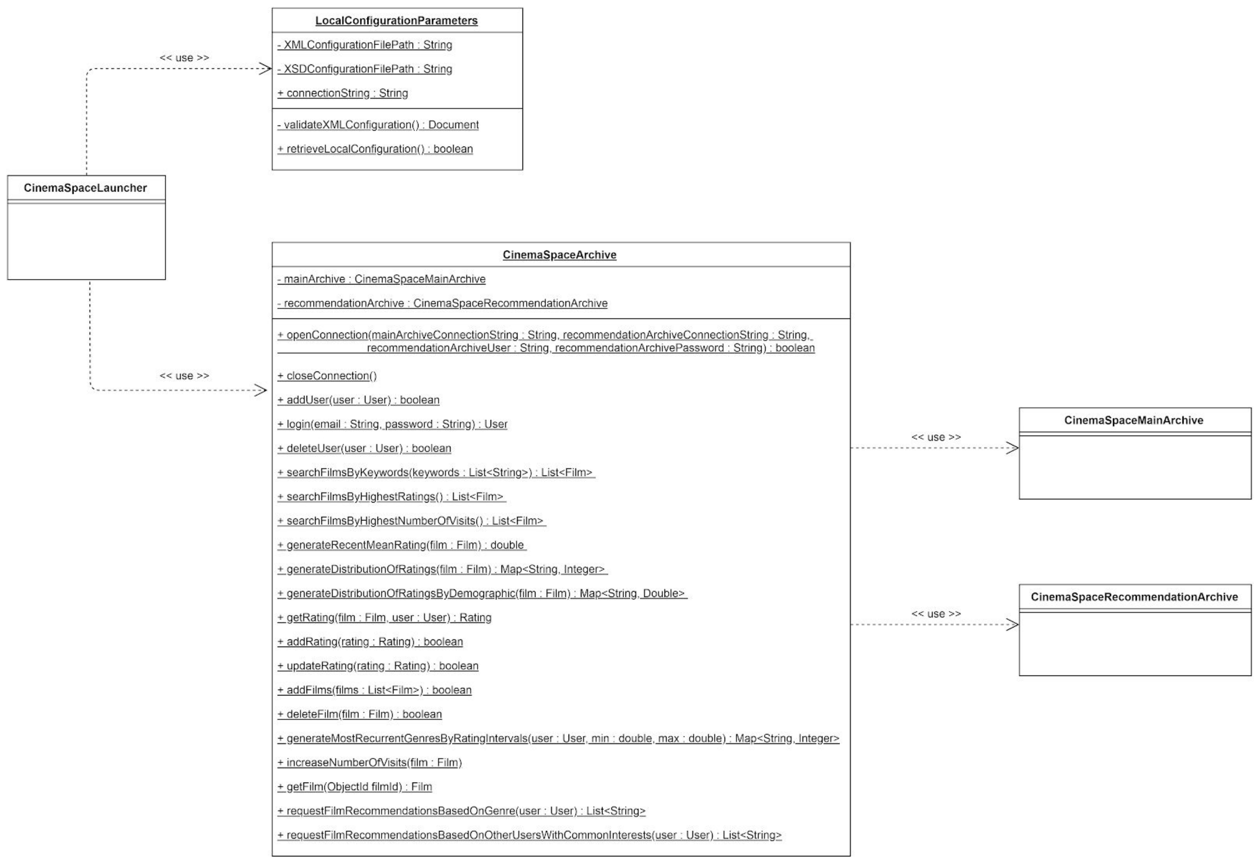

PersonalPageController
<< use >>
User
<< use >>
PieChart
<< use >>
CinemaSpaceArchive

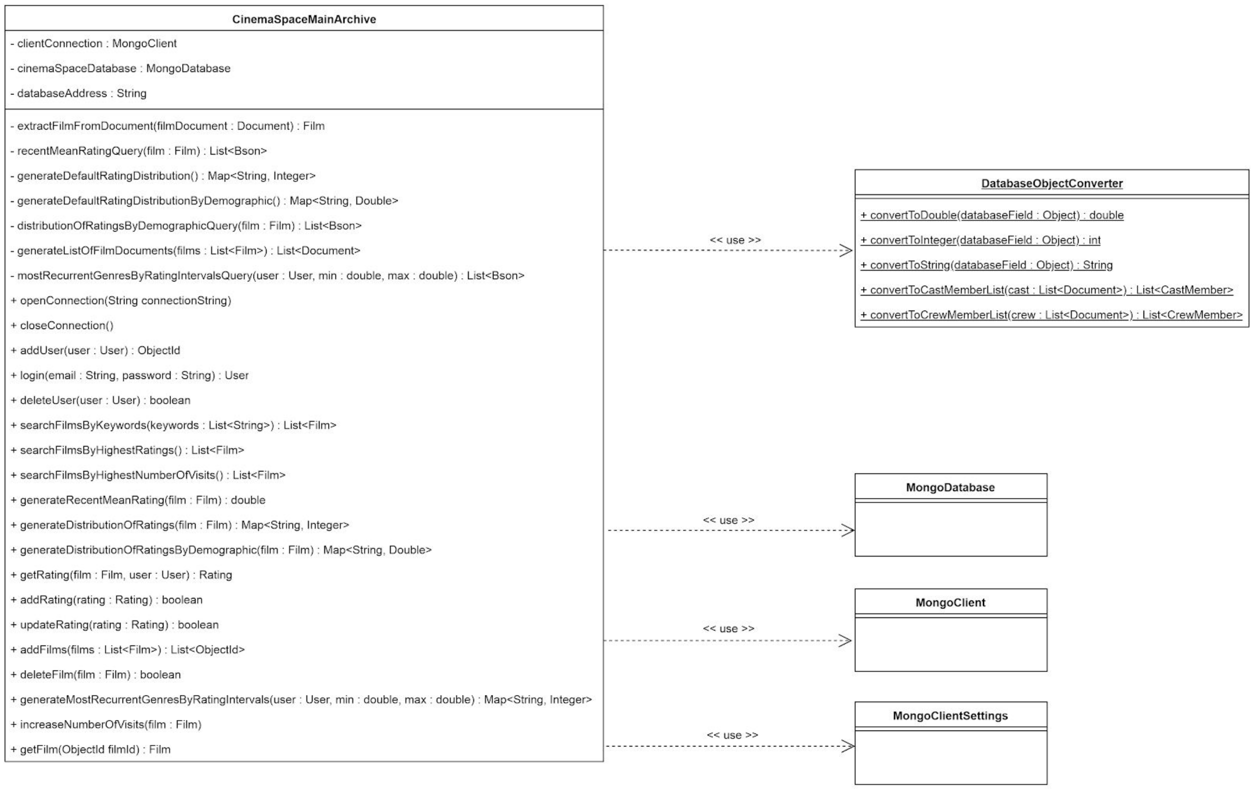

FilmPageController
<< use >>
User
<< use >>
Film
<< use >>
Rating
<< use >>
BarChart
<< use >>
CinemaSpaceArchive

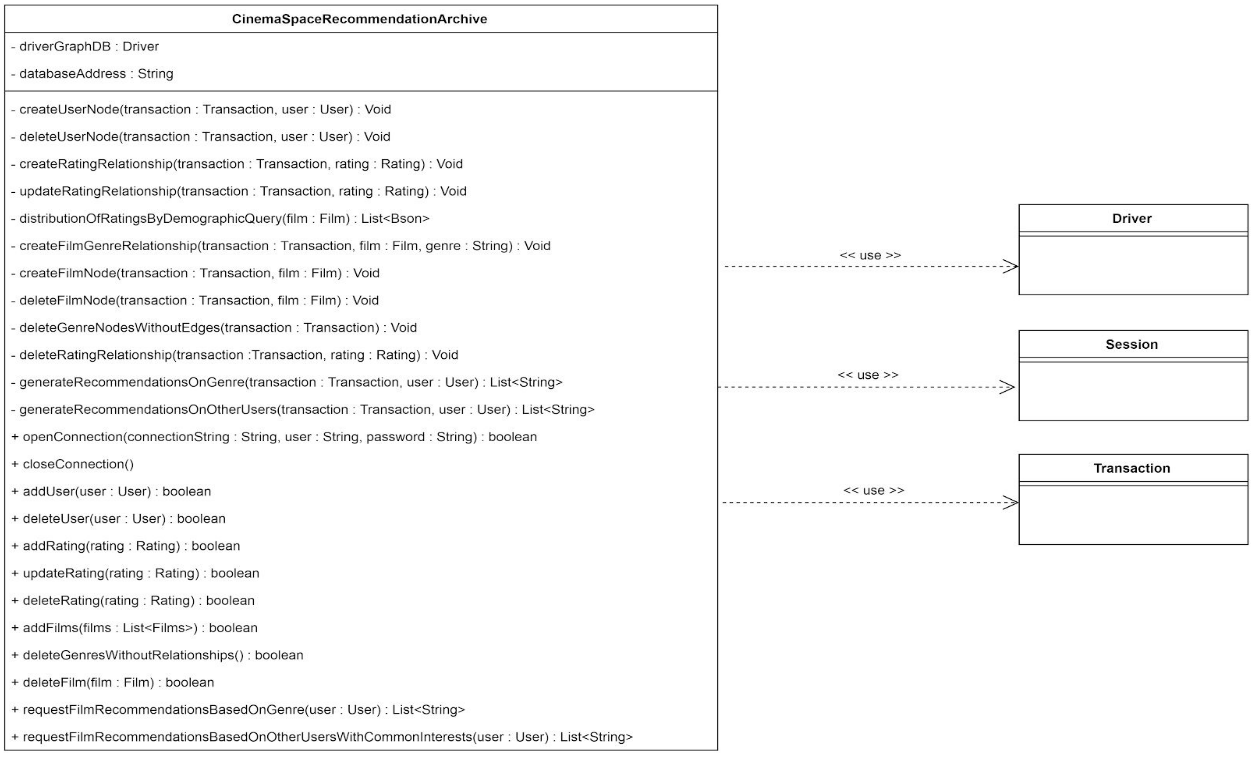

AddFilmsPageController
<< use >>
User
<< use >>
Film
<< use >>
JsonNode
<< use >>
CinemaSpaceArchive


Film
- id: ObjectId
- budget: double
- genres: List<String>
- homepage: String
- originalLanguage: String
- originalTitle: String
- overview: String
- posterPath: String
- productionCompanies: List<String>
- productionCountries: List<String>
- releaseDate: String
- revenue: double
- runtime: double
- spokenLanguages: List<String>
- status: String
- tagline: String
- title: String
- numberOfVisits: int
- keywords: List<String>
- cast: List<CastMember>
- crew: List<CrewMember>
- averageRatings: double
- numberOfRatings: int
+ Film(id: ObjectId, budget: double, genres: List<String>, homepage: String, originalLanguage: String, originalTitle: String, overview: String, posterPath: String, productionCompanies: List<String>, productionCountries: List<String>, releaseDate: String, revenue: double, runtime: double, spokenLanguages: List<String>, status: String, tagline: String, title: String, numberOfVisits: int, keywords: List<String>, cast: List<CastMember>, crew List<CrewMember>, averageRating: double, numberOfRatings: int)
+ getId(): ObjectId
+ getBudget: double
+ getGenres: List<String>
+ getHomepage: String
+ getOriginalLanguage: String
+ getOriginalTitle: String
+ getOverview: String
+ getPosterPath: String
+ getProductionCompanies: List<String>
+ getProductionCountries: List<String>
+ getReleaseDate: String
+ getRevenue: double
+ getRuntime: double
+ getSpokenLanguages: List<String>
+ getStatus: String
+ getTagline: String
+ getTitle: String
+ getNumberOfVisits: int
+ getKeywords: List<String>
+ getCast: List<CastMember>
+getCrew: List<CrewMember>
+ getAverageRatings: double
+ getNumberOfRatings: int
+ setId(id : ObjectId)
User
- id: ObjectId
- username: String
- password: String
- email: String
- dateOfBirth: String
- gender: String
- administrator: boolean
+ User(id: ObjectId, username: String, password: String, email: String, dateOfBirth: String, gender: String, administrator: boolean)
+ getId(): ObjectId
+ getUsername(): String
+ getPassword(): String
+ getEmail(): String
+ getDateOfBirth(): String
+ getGender(): String
+ getAdministrator(): boolean
Rating
- userId: ObjectId
- filmId: ObjectId
- rating: double
- timestamp: Long
- ageOfUser: int
- gender: String
+ Rating(userId: ObjectId, filmId: ObjectId, rating: double, timestamp: long, ageOfUser: int, gender: String)
+ getUserId: ObjectId
+ getFilmId: ObjectId
+ getRating: double
+ getTimestamp: Long
+ getAgeOfUser: int
+ getGender(): String
1
0...N
CastMember
- character: String
- name: String
- order: int
- profilePath: String
+ CastMember(character: String, name: String, order: int, profilePath: String,)
+ getCharacter(): String
+ getName(): String
+ getOrder(): int
+ getPathProfile(): String
1
0...N
CrewMember
- department: String
- job: String
- name: String
- profilePath: String
+ CrewMember(department: String, job: String, name: String, profilePath: String,)
+ getDepartment(): String
+ getJob(): String
+ getName(): String
+ getPathProfile(): String


Application
<< extend >>
CinemaSpaceLauncher
start(stage: Stage)
<< use >>
WelcomePageController
- user: User
- stage: Stage
- root: Parent
- email: String
- dateOfBirth: String
+ WelcomePageController()
# initialize()
# handleConfirmLoginButtonAction(event: ActionEvent)
# handleConfirmSignUpButtonAction(event: ActionEvent)
# handleFemaleRadioButtonAction(event: ActionEvent)
# handleMaleRadioButtonAction(event: ActionEvent)
+ isValidEmail(String: email): boolean
+ isValidDateOfBirth(String: dateOfBirth): boolean
<< use >>
HomePageController
- user: User
- listOfFilms: List<Film>
- stage: Stage
- root: Parent
- actualPage: int
- maxNbPages: int
- maxNbFilms: int
+ HomePageController()
# initialize()
# handleDisconnectionButtonAction(event: ActionEvent)
# handleProfileButtonAction(event: ActionEvent)
# handleMostPopularButtonAction(event: ActionEvent)
# handleHighestRatedButtonAction(event: ActionEvent)
# handleHomeButtonAction(event: ActionEvent)
# handlePreviousButtonAction(event: ActionEvent)
# handleNextButtonAction(event: ActionEvent)
# handleSearchButtonAction(event: ActionEvent)
# handleFilm1PosterButtonAction(event: ActionEvent)
# handleFilm1TitleButtonAction(event: ActionEvent)
# handleFilm2PosterButtonAction(event: ActionEvent)
# handleFilm2TitleButtonAction(event: ActionEvent)
# handleFilm3PosterButtonAction(event: ActionEvent)
# handleFilm3TitleButtonAction(event: ActionEvent)
# handleFilm4PosterButtonAction(event: ActionEvent)
# handleFilm4TitleButtonAction(event: ActionEvent)
# handleFilm5PosterButtonAction(event: ActionEvent)
# handleFilm5TitleButtonAction(event: ActionEvent)
# handleFilm6PosterButtonAction(event: ActionEvent)
# handleFilm6TitleButtonAction(event: ActionEvent)
# handleFilm7PosterButtonAction(event: ActionEvent)
# handleFilm7TitleButtonAction(event: ActionEvent)
# handleFilm8PosterButtonAction(event: ActionEvent)
# handleFilm8TitleButtonAction(event: ActionEvent)
+ initUser(user: User)
+ initListOfFilms(listOfFilms: List<Film>)
+ mostPopular()
+ highestRated()
+ clearListOfFilms()
+ displayListOfFilms()
<< use >>
PersonalPageController
- user: User
- listOfFilmsGenres: List<Film>
- listOfFilmsUsers: List<Film>
- listOfFilmsIdGenres: List<String>
- listOfFilmsIdUsers: List<String>
- stage: Stage
- root: Parent
- actualPageGenres: int
- maxNbPagesGenres: int
- maxNbFilmsGenres: int
- actualPageUsers: int
- maxNbFilmsUsers: int
- maxNbPagesUsers: int
+ HomePageController()
# initialize()
# handleDisconnectionButtonAction(event: ActionEvent)
# handleProfileButtonAction(event: ActionEvent)
# handleMostPopularButtonAction(event: ActionEvent)
# handleHighestRatedButtonAction(event: ActionEvent)
# handleHomeButtonAction(event: ActionEvent)
# handleFilm1PosterGenresButtonAction(event: ActionEve
# handleFilm1TitleGenresButtonAction(event: ActionEvent
# handleFilm2PosterGenresButtonAction(event: ActionEve
# handleFilm2TitleGenresButtonAction(event: ActionEvent
# handleFilm3PosterGenresButtonAction(event: ActionEve
# handleFilm3TitleGenresButtonAction(event: ActionEvent
# handleFilm4PosterGenresButtonAction(event: ActionEve
# handleFilm4TitleGenresButtonAction(event: ActionEvent
# handleFilm1PosterUsersButtonAction(event: ActionEven
# handleFilm1TitleUsersButtonAction(event: ActionEvent)
# handleFilm2PosterUsersButtonAction(event: ActionEven
# handleFilm2TitleUsersButtonAction(event: ActionEvent)
# handleFilm3PosterUsersButtonAction(event: ActionEven
# handleFilm3TitleUsersButtonAction(event: ActionEvent)
# handleFilm4PosterUsersButtonAction(event: ActionEven
# handleFilm4TitleUsersButtonAction(event: ActionEvent)
# handlePreviousGenresButtonAction(event: ActionEvent)
# handleNextGenresButtonAction(event: ActionEvent)
# handlePreviousUsersButtonAction(event: ActionEvent)
# handleNextUsersButtonAction(event: ActionEvent)
+ initUser(user: User)
+ initListOfFilmGenres(user: User)
+ initListOfFilmUsers(user: User)
+ clearListOfFilmsGenres()
+ displayListOfFilmsGenres()
+ clearListOfFilmsUsers()
+ displayListOfFilmsUsers()
FilmPageController
- user: User
- films: Film
- stage: Stage
- root: Parent
- overalMeanRatingDouble: double
- recentMeanRatingDouble: double
+ FilmPageController()
# initialize()
# handleDisconnectionButtonAction(event: ActionEvent)
# handleProfileButtonAction(event: ActionEvent)
# handleMostPopularButtonAction(event: ActionEvent)
# handleHighestRatedButtonAction(event: ActionEvent)
# handleHomeButtonAction(event: ActionEvent)
# handleRateButtonAction(event: ActionEvent)
# handleConfirmDeleteButtonAction(event: ActionEvent)
+ initUser(user: User)
+ initFilm(film: Film)
AddFilmsPageController
- user: User
- stage: Stage
- root: Parent
+ AddFilmsPageController()
# initialize()
# handleDisconnectionButtonAction(event: ActionEvent)
# handleProfileButtonAction(event: ActionEvent)
# handleMostPopularButtonAction(event: ActionEvent)
# handleHighestRatedButtonAction(event: ActionEvent)
# handleHomeButtonAction(event: ActionEvent)
# handleAddFilmsButtonAction(event: ActionEvent)
# handleUploadFileButtonAction(event: ActionEvent)
# handleConfirmAddFilmsButtonAction(event: ActionEvent
+ initUser(user: User)
<< use >>
<< use >>
<< use >>
<< use >>
<< use >>
<< use >>


LocalConfigurationParameters
- XMLConfigurationFilePath : String
- XSDConfigurationFilePath : String
+ connectionString : String
- validateXMLConfiguration() : Document
+ retrieveLocalConfiguration() : boolean
<< use >>
CinemaSpaceLauncher
<< use >>
CinemaSpaceArchive
- mainArchive : CinemaSpaceMainArchive
- recommendationArchive : CinemaSpaceRecommendationArchive
+ openConnection(mainArchiveConnectionString : String, recommendationArchiveConnectionString : String, recommendationArchiveUser : String, recommendationArchivePassword : String) : boolean
+ closeConnection()
+ addUser(user : User) : boolean
+ login(email : String, password : String) : User
+ deleteUser(user : User) : boolean
+ searchFilmsByKeywords(keywords : List<String>) : List<Film>
+ searchFilmsByHighestRatings() : List<Film>
+ searchFilmsByHighestNumberOfVisits() : List<Film>
+ generateRecentMeanRating(film : Film) : double
+ generateDistributionOfRatings(film : Film) : Map<String, Integer>
+ generateDistributionOfRatingsByDemographic(film : Film) : Map<String, Double>
+ getRating(film : Film, user : User) : Rating
+ addRating(rating : Rating) : boolean
+ updateRating(rating : Rating) : boolean
+ addFilms(films : List<Film>) : boolean
+ deleteFilm(film : Film) : boolean
+ generateMostRecurrentGenresByRatingIntervals(user : User, min : double, max : double) : Map<String, Integer>
+ increaseNumberOfVisits(film : Film)
+ getFilm(ObjectId filmId) : Film
+ requestFilmRecommendationsBasedOnGenre(user : User) : List<String>
+ requestFilmRecommendationsBasedOnOtherUsersWithCommonInterests(user : User) : List<String>
<< use >>
CinemaSpaceMainArchive
<< use >>
CinemaSpaceRecommendationArchive


CinemaSpaceMainArchive
- clientConnection : MongoClient
- cinemaSpaceDatabase : MongoDatabase
- databaseAddress : String
- extractFilmFromDocument(filmDocument : Document) : Film
- recentMeanRatingQuery(film : Film) : List<Bson>
- generateDefaultRatingDistribution() : Map<String, Integer>
- generateDefaultRatingDistributionByDemographic() : Map<String, Double>
- distributionOfRatingsByDemographicQuery(film : Film) : List<Bson>
- generateListOfFilmDocuments(films : List<Film>) : List<Document>
- mostRecurrentGenresByRatingIntervalsQuery(user : User, min : double, max : double) : List<Bson>
+ openConnection(String connectionString)
+ closeConnection()
+ addUser(user : User) : ObjectId
+ login(email : String, password : String) : User
+ deleteUser(user : User) : boolean
+ searchFilmsByKeywords(keywords : List<String>) : List<Film>
+ searchFilmsByHighestRatings() : List<Film>
+ searchFilmsByHighestNumberOfVisits() : List<Film>
+ generateRecentMeanRating(film : Film) : double
+ generateDistributionOfRatings(film : Film) : Map<String, Integer>
+ generateDistributionOfRatingsByDemographic(film : Film) : Map<String, Double>
+ getRating(film : Film, user : User) : Rating
+ addRating(rating : Rating) : boolean
+ updateRating(rating : Rating) : boolean
+ addFilms(films : List<Film>) : List<ObjectId>
+ deleteFilm(film : Film) : boolean
+ generateMostRecurrentGenresByRatingIntervals(user : User, min : double, max : double) : Map<String, Integer>
+ increaseNumberOfVisits(film : Film)
+ getFilm(ObjectId filmId) : Film
<< use >>
DatabaseObjectConverter
+ convertToDouble(databaseField : Object) : double
+ convertToInteger(databaseField : Object) : int
+ convertToString(databaseField : Object) : String
+ convertToCastMemberList(cast : List<Document>) : List<CastMember>
+ convertToCrewMemberList(crew : List<Document>) : List<CrewMember>
<< use >>
MongoDatabase
<< use >>
MongoClient
<< use >>
MongoClientSettings


CinemaSpaceRecommendationArchive
- driverGraphDB : Driver
- databaseAddress : String
- createUserNode(transaction : Transaction, user : User) : Void
- deleteUserNode(transaction : Transaction, user : User) : Void
- createRatingRelationship(transaction : Transaction, rating : Rating) : Void
- updateRatingRelationship(transaction : Transaction, rating : Rating) : Void
- distributionOfRatingsByDemographicQuery(film : Film) : List<Bson>
- createFilmGenreRelationship(transaction : Transaction, film : Film, genre : String) : Void
- createFilmNode(transaction : Transaction, film : Film) : Void
- deleteFilmNode(transaction : Transaction, film : Film) : Void
- deleteGenreNodesWithoutEdges(transaction : Transaction) : Void
- deleteRatingRelationship(transaction :Transaction, rating : Rating) : Void
- generateRecommendationsOnGenre(transaction : Transaction, user : User) : List<String>
- generateRecommendationsOnOtherUsers(transaction : Transaction, user : User) : List<String>
+ openConnection(connectionString : String, user : String, password : String) : boolean
+ closeConnection()
+ addUser(user : User) : boolean
+ deleteUser(user : User) : boolean
+ addRating(rating : Rating) : boolean
+ updateRating(rating : Rating) : boolean
+ deleteRating(rating : Rating) : boolean
+ addFilms(films : List<Films>) : boolean
+ deleteGenresWithoutRelationships() : boolean
+ deleteFilm(film : Film) : boolean
+ requestFilmRecommendationsBasedOnGenre(user : User) : List<String>
+ requestFilmRecommendationsBasedOnOtherUsersWithCommonInterests(user : User) : List<String>
<< use >>
Driver
<< use >>
Session
<< use >>
Transaction

Class responsibilities

User: represents a user in the domain of CinemaSpace.

Film: represents a film in the domain of CinemaSpace.

Rating: represents a rating in the domain of CinemaSpace.

CinemaSpaceLauncher: manages the launch of the application, coordinating the retrieval of the configuration file, the connection to the database and the initialization of the GUI. After this phase, it leaves the control of the application flow to the WelcomePageController.

WelcomePageController: manages the welcome page, coordinating the signup and login services offered to the user. If a login is successfully requested, it leaves the control of the application flow to the HomePageController.

HomePageController: manages the home page and it is responsible for the visualization of the films searched by keywords, by highest ratings and by number of visits. It leaves the control of the application to the FilmPageController, PersonalPageController, AddFilmsPageController or WelcomePageController according to the user requests.

FilmPageController: manages the film page, showing the associated information and statistics, and gives the possibility to use the rating system of the community. It leaves the control of the application to the HomePageController, PersonalPageController, AddFilmsPageController or WelcomePageController according to the user requests.

PersonalPageController: manages the personal profile page of a user, showing the associated information/statistics, displaying the film recommendations and giving the possibility to close the account. It leaves the control of the application to the HomePageController, FilmPageController or WelcomePageController according to the user requests.

AddFilmsPageController: manages the administrator page for adding films to the database from JSON files. It leaves the control of the application to the HomePageController or WelcomePageController according to the user requests.

CinemaSpaceArchive: manages and coordinates the connection to both the main archive and the recommendation archive, with particular attention to the consistency of the overall dataset, using opportune instances of CinemaSpaceMainArchive and CinemaSpaceRecommendationArchive. It is responsible for responding to the specific requests of the different page controllers and it returns opportune objects of type List<Film>, User, Rating and Map< > (the latter in the case of statistic distribution requests).

CinemaSpaceMainArchive: manages the connection to the main archive, using the MongoDB API.

CinemaSpaceRecommendationArchive: manages the connection to the recommendation archive, using the Neo4j API.

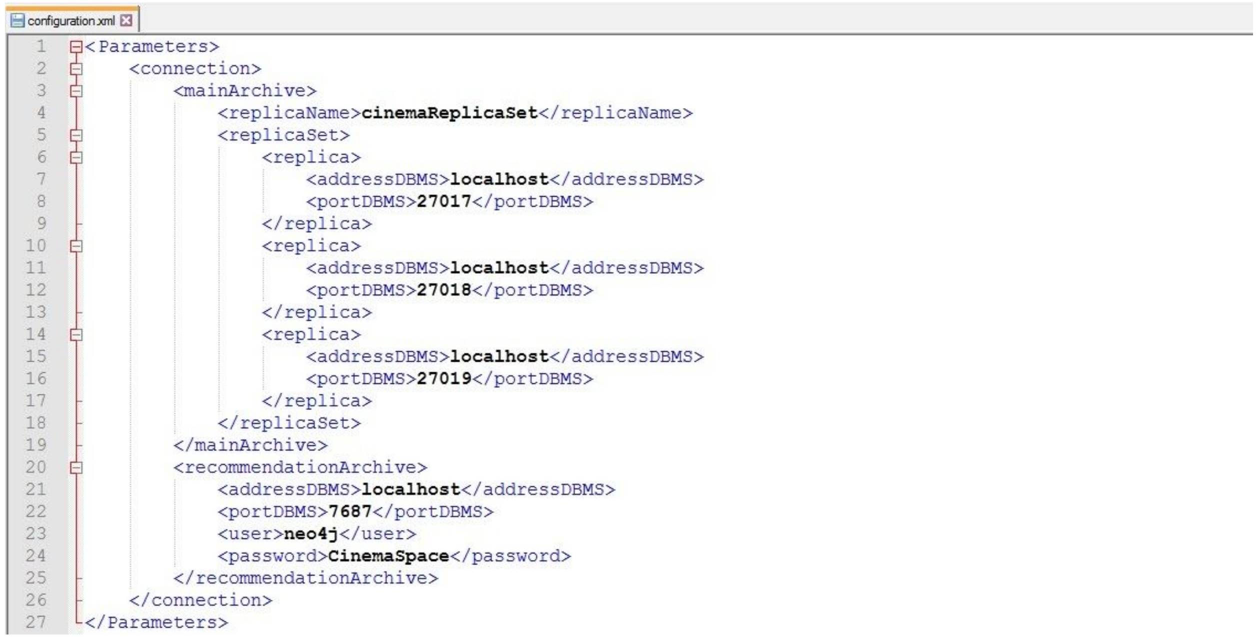
DatabaseObjectConverter: offers utility methods for the conversion of types. It is used by CinemaSpaceMainArchive to manage the retrieval of documents with non-fixed types from the MongoDB database.

LocalConfigurationParameters: retrieves and validates the local configuration stored in an XML file, making available its parameters to all the classes.

Note: the connections to the databases are managed by connection pools, which are initialized at the start of the application and closed at the end: in particular, the latter is ensured by binding the opportune method to the *onCloseRequest* window event. Both these steps are executed in the CinemaSpaceLauncher class.

6.2 Test manual

To be able to connect the application with the database, an XML configuration file with the addresses of the replica set members must be specified, with the following format:



```xml
<Parameters>
    <connection>
        <mainArchive>
            <replicaName>cinemaReplicaSet</replicaName>
            <replicaSet>
                <replica>
                    <addressDBMS>localhost</addressDBMS>
                    <portDBMS>27017</portDBMS>
                </replica>
                <replica>
                    <addressDBMS>localhost</addressDBMS>
                    <portDBMS>27018</portDBMS>
                </replica>
                <replica>
                    <addressDBMS>localhost</addressDBMS>
                    <portDBMS>27019</portDBMS>
                </replica>
            </replicaSet>
        </mainArchive>
        <recommendationArchive>
            <addressDBMS>localhost</addressDBMS>
            <portDBMS>7687</portDBMS>
            <user>neo4j</user>
            <password>CinemaSpace</password>
        </recommendationArchive>
    </connection>
</Parameters>
```

The configuration is validated at the start using a pre-defined XSD file.

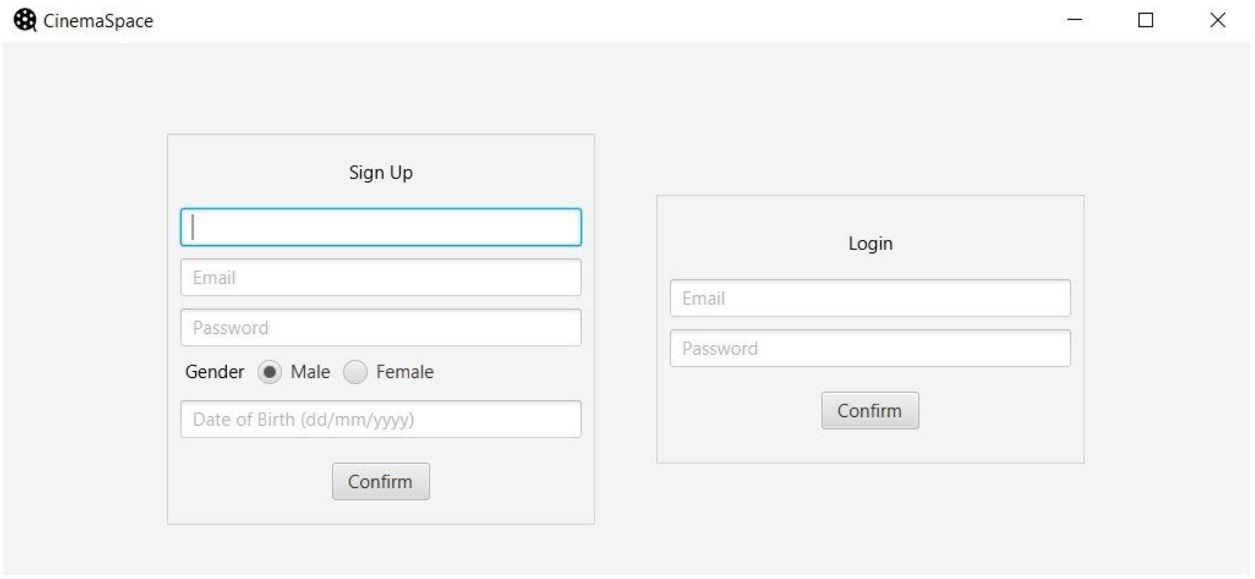
6.2.1 Client user

Sign Up

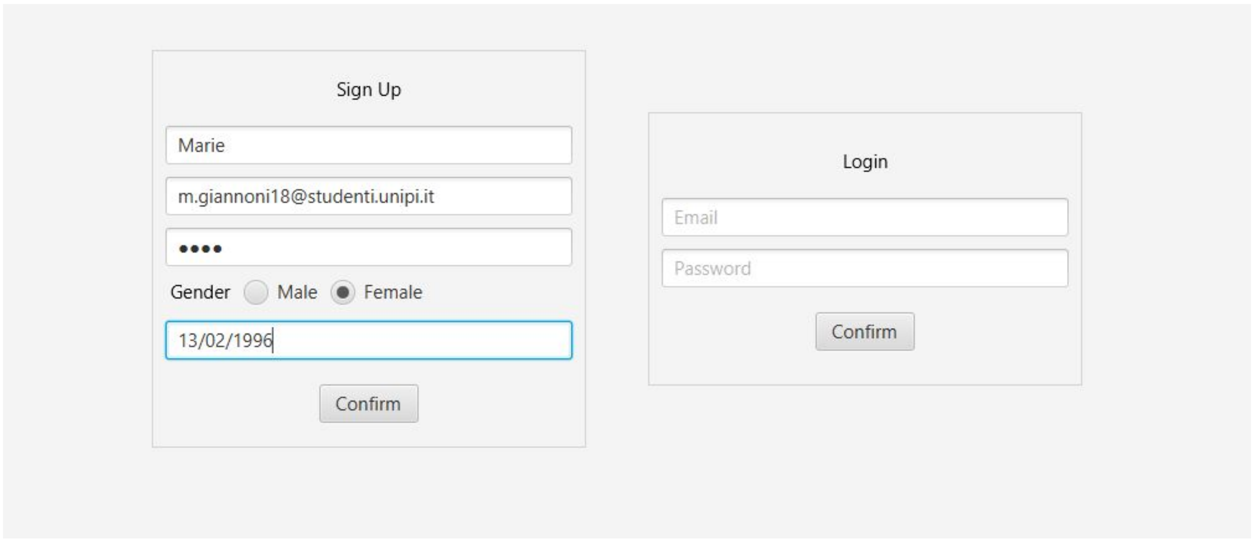
Preconditions: None

Steps:

1. The user opens the application.



2. The user fills in all the fields in the Sign Up form and click on the "Confirm" button.



Marie

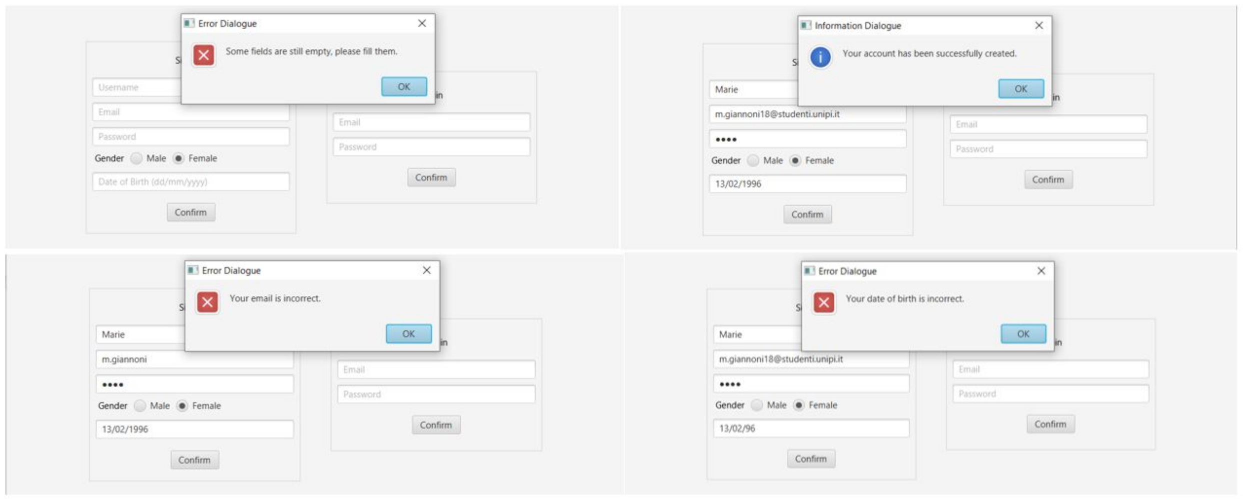
m.giannoni18@studenti.unipi.it

••••

Gender ○ Male ● Female

13/02/1996

The email must respect the pattern abc@efg.h, while the date of birth must be in the format dd/mm/yyyy. If these patterns are not respected, the application will show an error message.


Error Dialogue
Some fields are still empty, please fill them.

Username, Email, Password, Gender: Female, Date of Birth (dd/mm/yyyy)

Information Dialogue
Your account has been successfully created.

Marie
m.giannoni18@studenti.unipi.it
••••
Gender: Female
13/02/1996

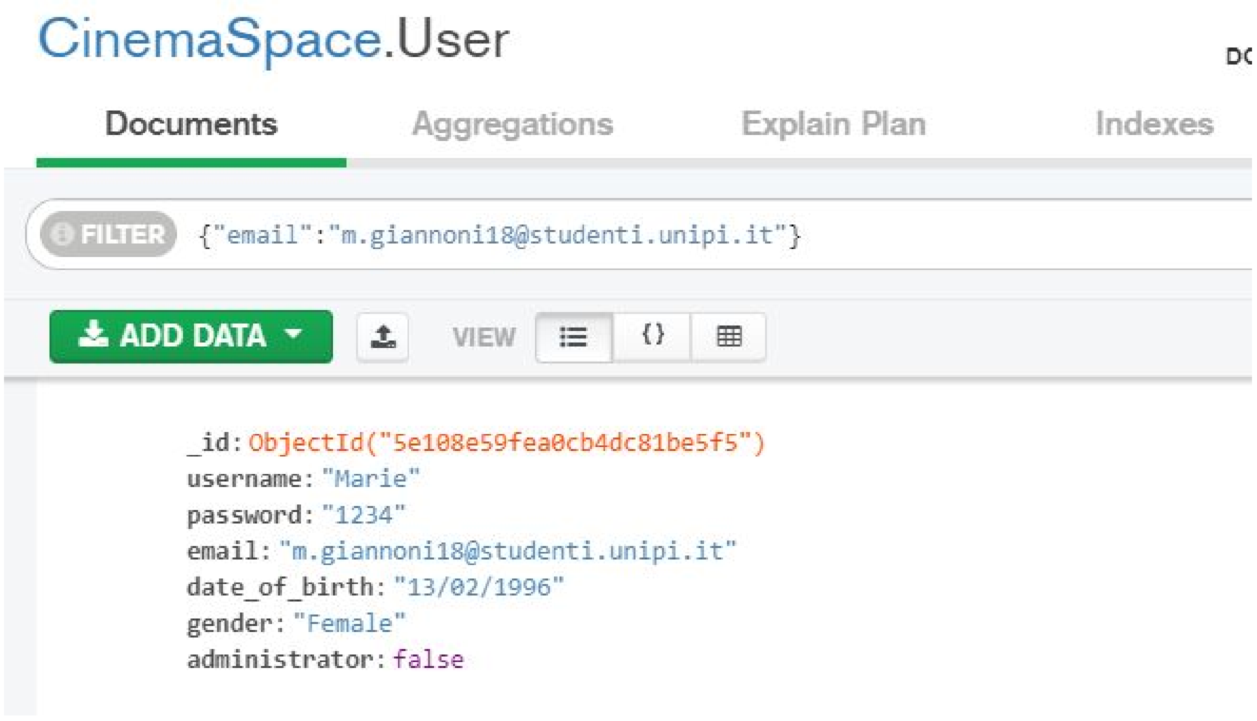
Error Dialogue
Your email is incorrect.

Marie
m.giannoni
••••
Gender: Female
13/02/1996

Error Dialogue
Your date of birth is incorrect.

Marie
m.giannoni18@studenti.unipi.it
••••
Gender: Female
13/02/96

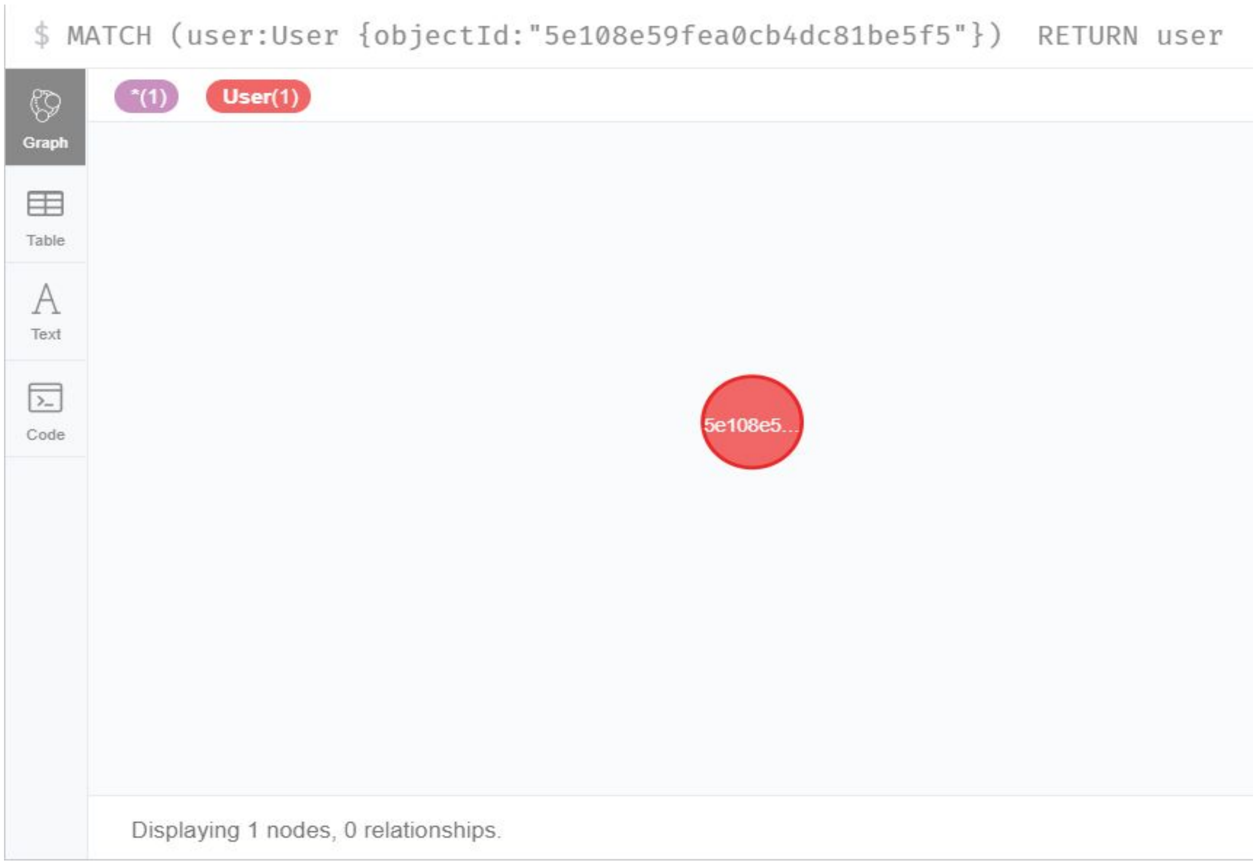
Databases status after the insertion - MongoDB



```
{"email":"m.giannoni18@studenti.unipi.it"}
```

```
_id: ObjectId("5e108e59fea0cb4dc81be5f5")
username: "Marie"
password: "1234"
email: "m.giannoni18@studenti.unipi.it"
date_of_birth: "13/02/1996"
gender: "Female"
administrator: false
```

Databases status after the insertion - Neo4j



```
$ MATCH (user:User {objectId:"5e108e59fea0cb4dc81be5f5"}) RETURN user
```

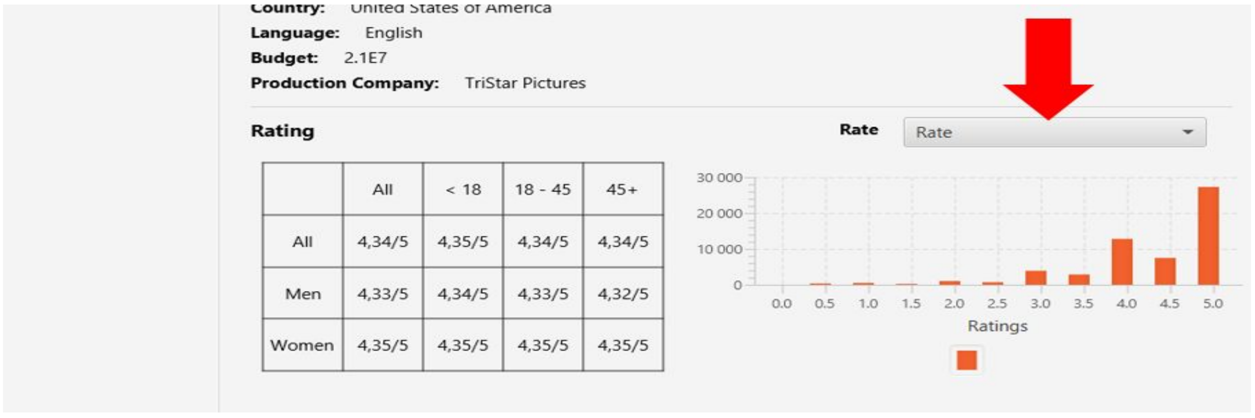
Displaying 1 nodes, 0 relationships.

Rate a film

Preconditions: The user must have an account. The user must be logged in and on the film page.

Steps:

1. The user clicks on the combo box named “Rate”.



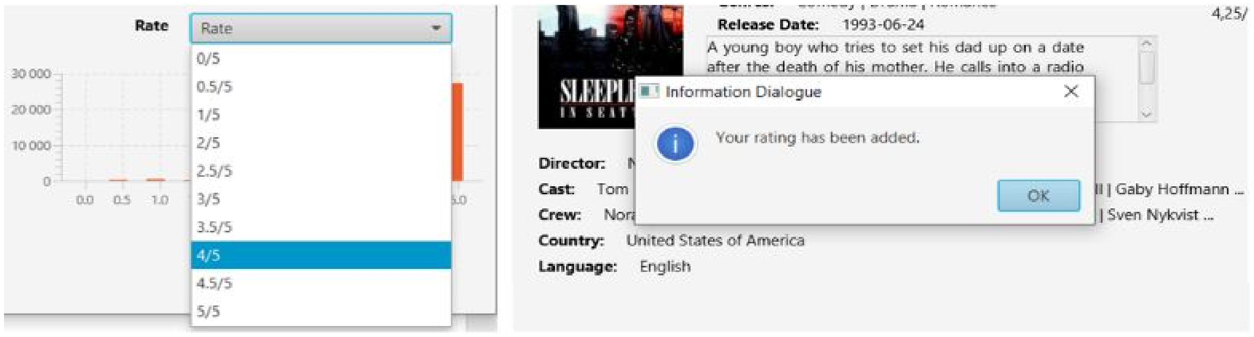
Country: United States of America
Language: English
Budget: 2.1E7
Production Company: TriStar Pictures

Rating

Rate Rate

	All	< 18	18 - 45	45+
All	4,34/5	4,35/5	4,34/5	4,34/5
Men	4,33/5	4,34/5	4,33/5	4,32/5
Women	4,35/5	4,35/5	4,35/5	4,35/5

2. The user selects a rating in the combo box.



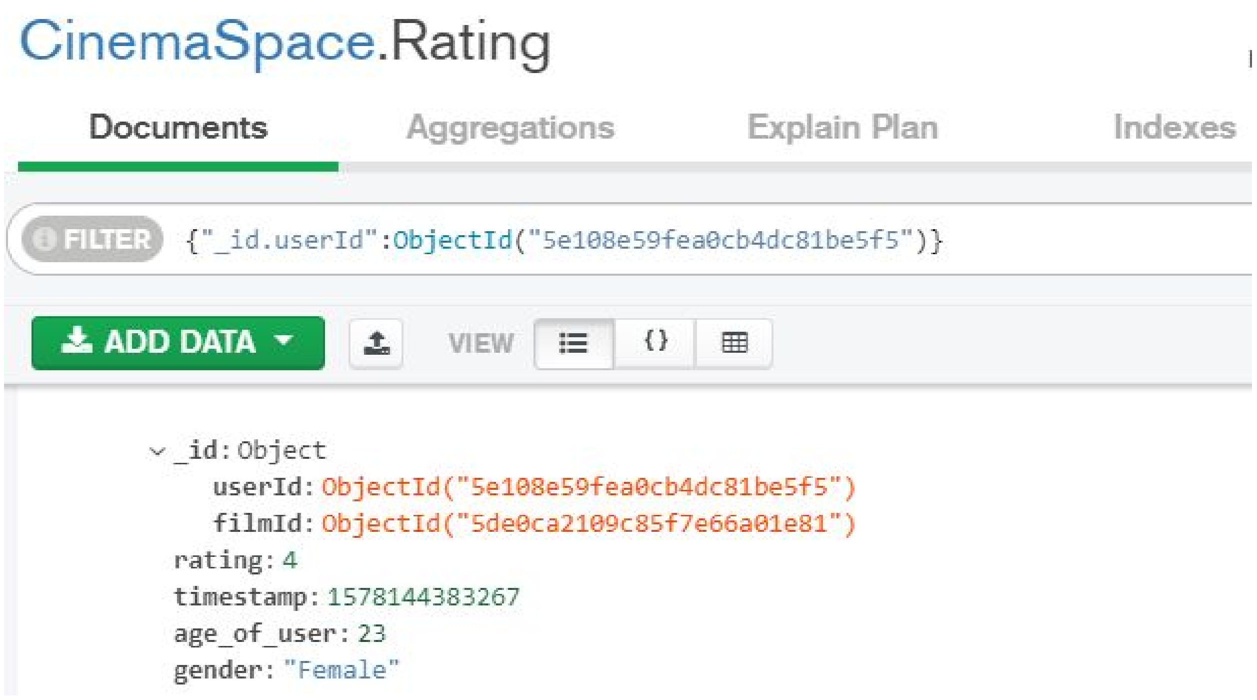
A young boy who tries to set his dad up on a date after the death of his mother. He calls into a radio

Information Dialogue

Your rating has been added.

OK

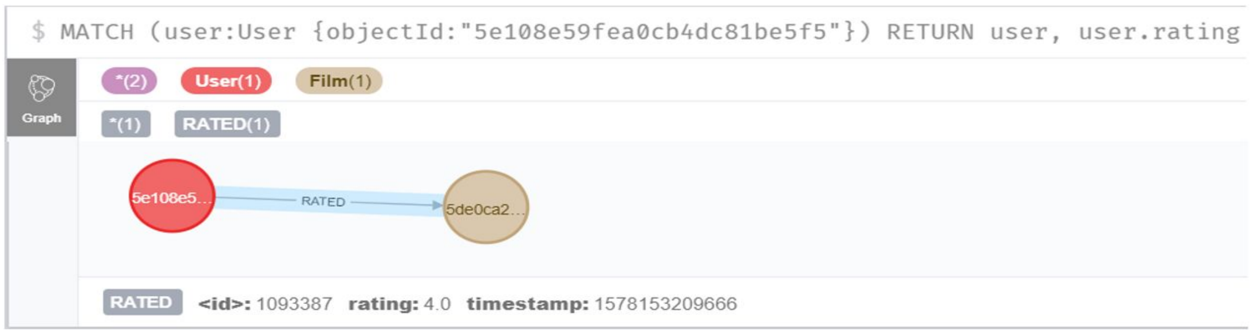
Database status after the insertion - MongoDB



```
{"_id.userId":ObjectId("5e108e59fea0cb4dc81be5f5")}
```

```
_id: Object
   userId: ObjectId("5e108e59fea0cb4dc81be5f5")
   filmId: ObjectId("5de0ca2109c85f7e66a01e81")
rating: 4
timestamp: 1578144383267
age_of_user: 23
gender: "Female"
```

Database status after the insertion - Neo4j



```
$ MATCH (user:User {objectId:"5e108e59fea0cb4dc81be5f5"}) RETURN user, user.rating
```

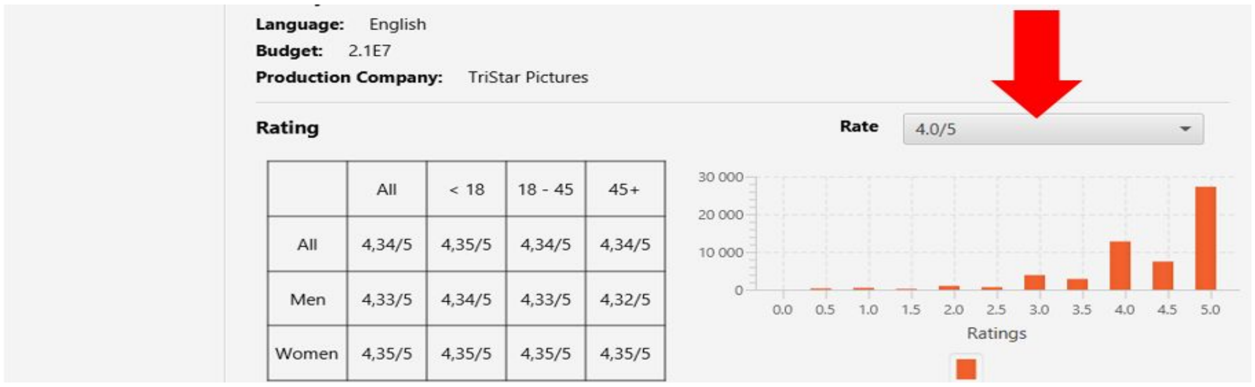
RATED <id>: 1093387 rating: 4.0 timestamp: 1578153209666

Update a rating of a film

Preconditions: The user must have an account. The user must be logged in and on the page of a film she rated.

Steps:

1. The user clicks on the combo box with the given rating.



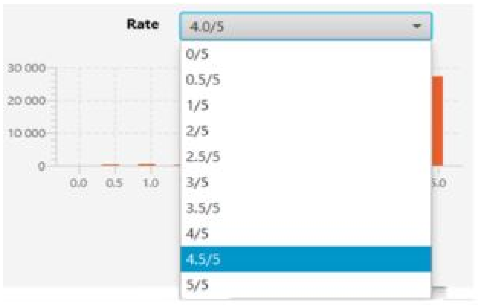
Language: English
Budget: 2.1E7
Production Company: TriStar Pictures

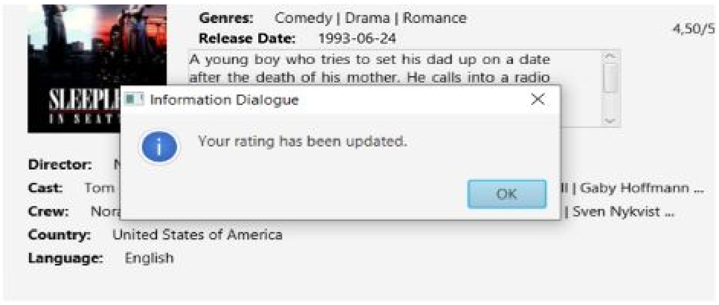
Rating

Rate 4.0/5

	All	< 18	18 - 45	45+
All	4,34/5	4,35/5	4,34/5	4,34/5
Men	4,33/5	4,34/5	4,33/5	4,32/5
Women	4,35/5	4,35/5	4,35/5	4,35/5

2. The user selects a rating in the combo box items.





Genres: Comedy | Drama | Romance
Release Date: 1993-06-24

4,50/5

A young boy who tries to set his dad up on a date after the death of his mother. He calls into a radio

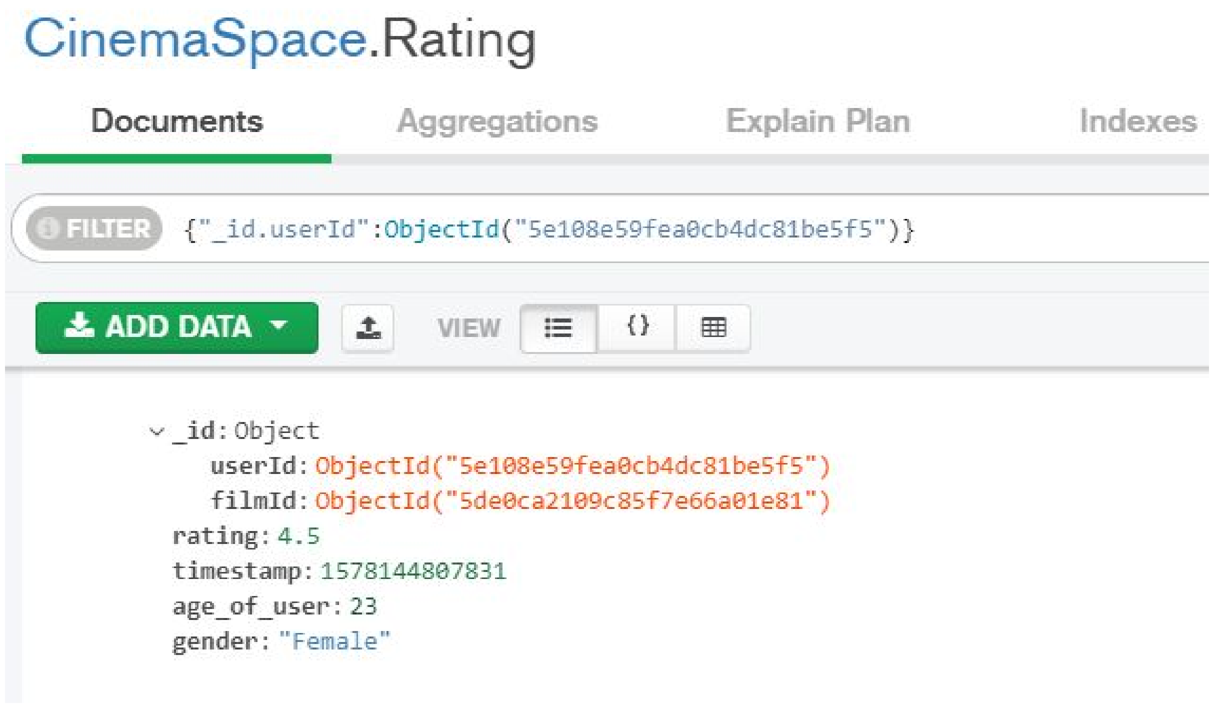
Information Dialogue

Your rating has been updated.

OK

Director: N
Cast: Tom ll | Gaby Hoffmann ...
Crew: Nora | Sven Nykvist ...
Country: United States of America
Language: English

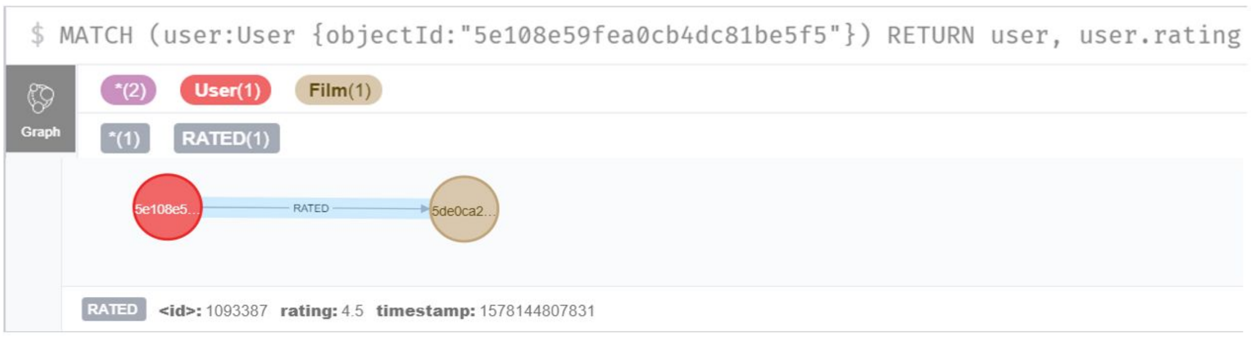
Database status after the insertion - MongoDB



```
{"_id.userId":ObjectId("5e108e59fea0cb4dc81be5f5")}
```

```
_id: Object
   userId: ObjectId("5e108e59fea0cb4dc81be5f5")
   filmId: ObjectId("5de0ca2109c85f7e66a01e81")
rating: 4.5
timestamp: 1578144807831
age_of_user: 23
gender: "Female"
```

Database status after the insertion - Neo4j



```
$ MATCH (user:User {objectId:"5e108e59fea0cb4dc81be5f5"}) RETURN user, user.rating
```

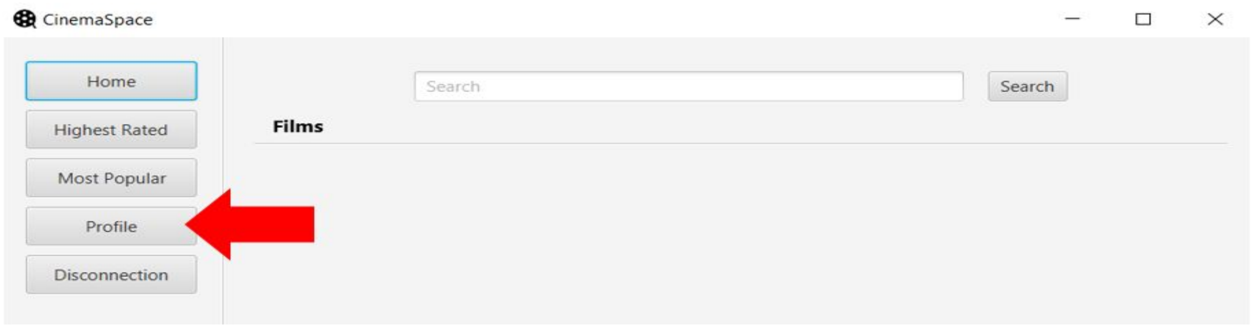
RATED <id>: 1093387 rating: 4.5 timestamp: 1578144807831

See film recommendations based on genres

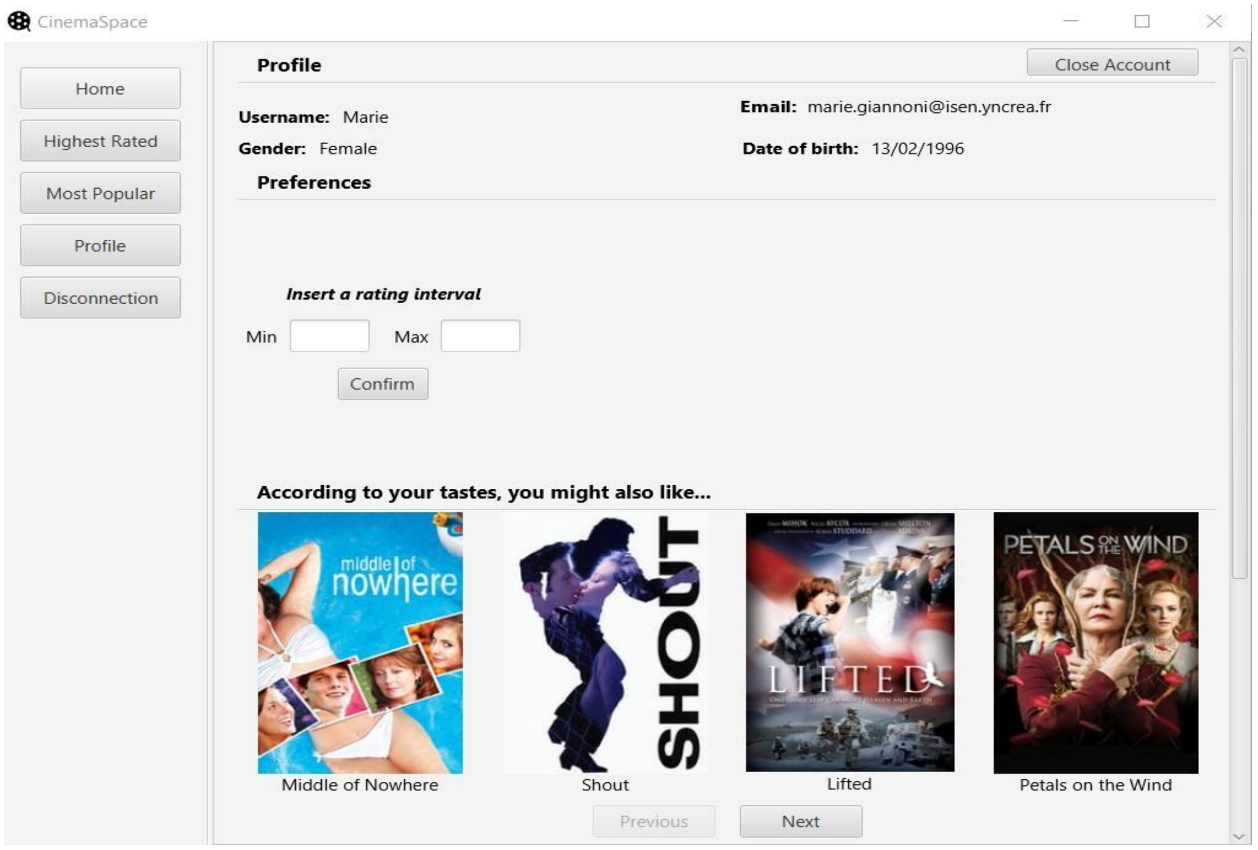
Preconditions: The user must have a profile. The user must be logged in. The user must have rated at least one film.

Steps:

1. The user clicks on the “Profile” button.

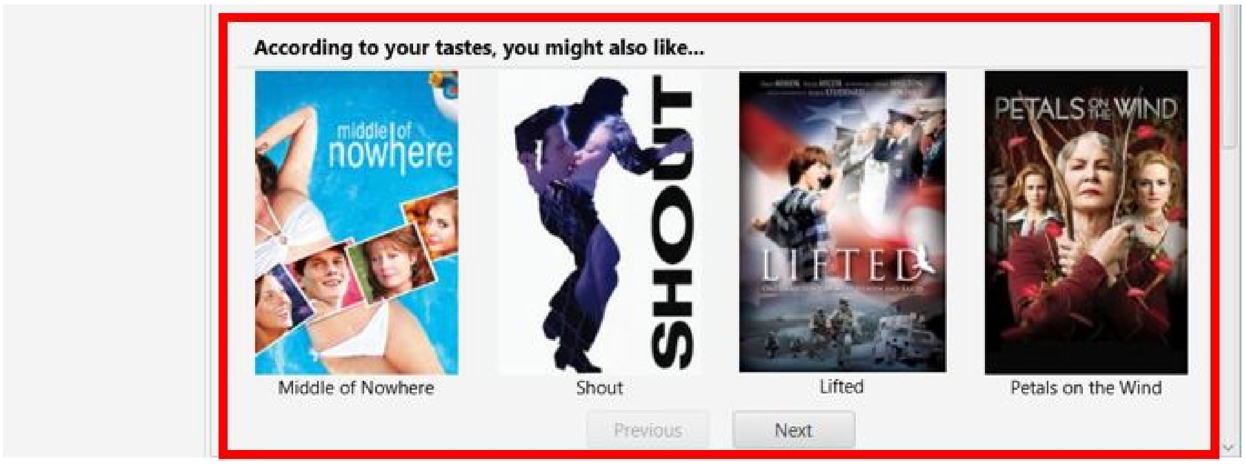


2. The “Profile” page is displayed.



Username: Marie
Gender: Female
Email: marie.giannoni@isen.yncrea.fr
Date of birth: 13/02/1996

3. The recommendations based on genres are displayed on the section "According to your tastes, you might also like…".

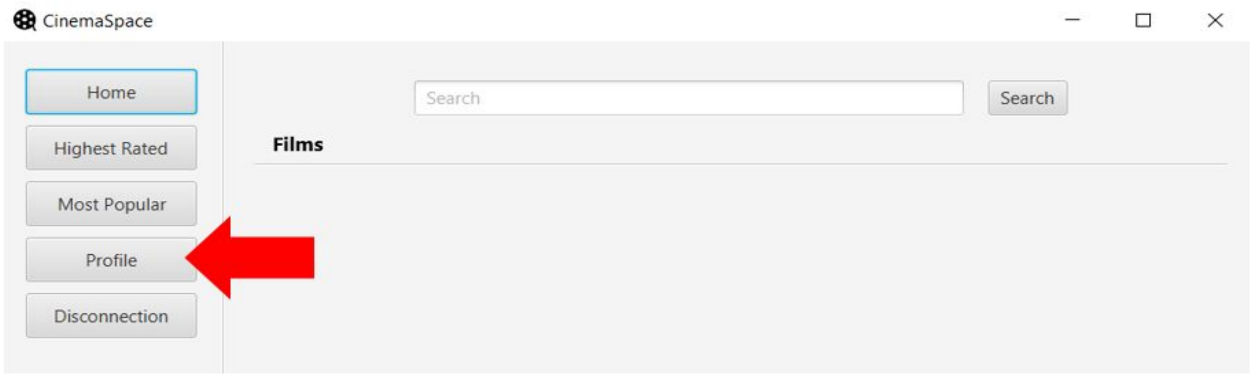


See film recommendations based on other users with common interests

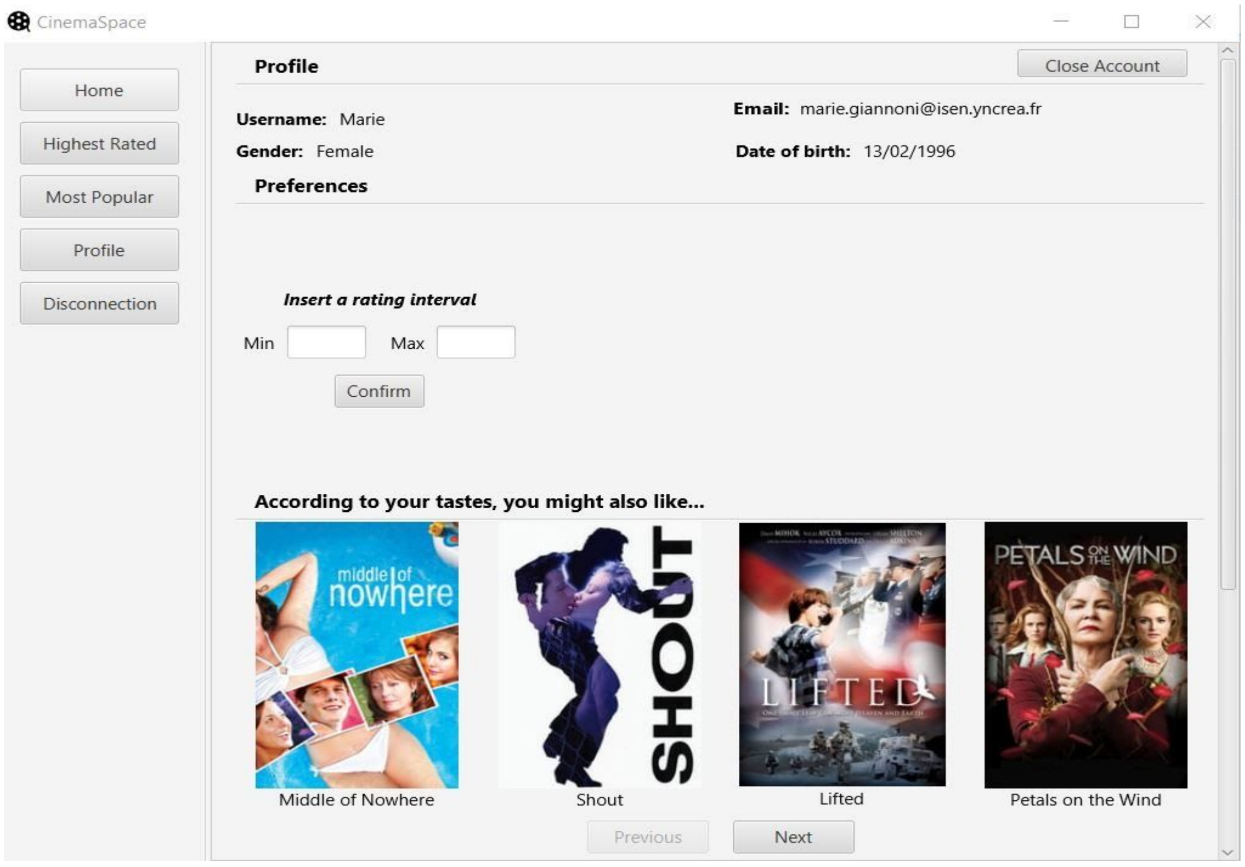
Preconditions: The user must have a profile. The user must be logged in. The user must have rated at least one film.

Steps:

1. The user clicks on the “Profile” button.

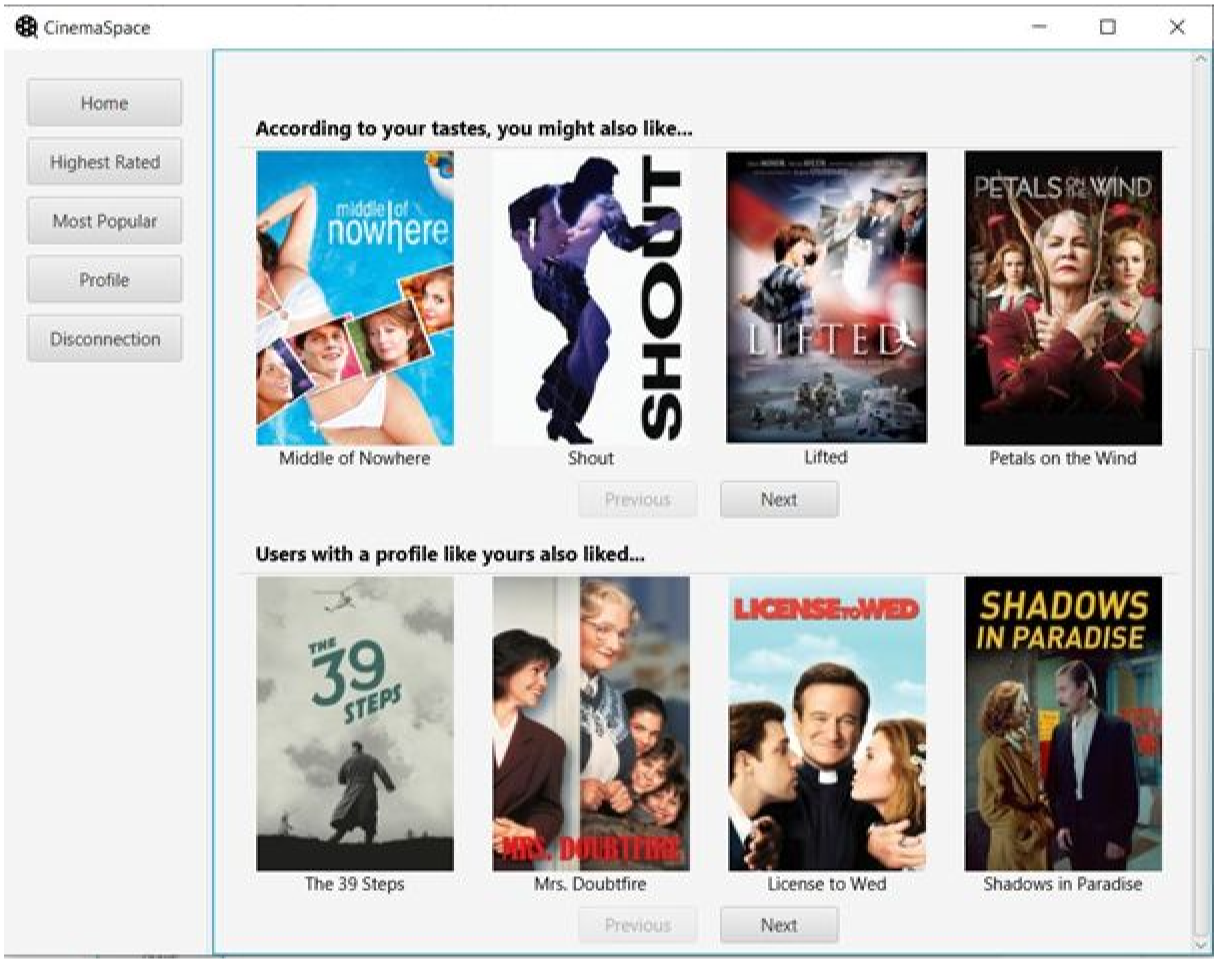


2. The “Profile” page is displayed.

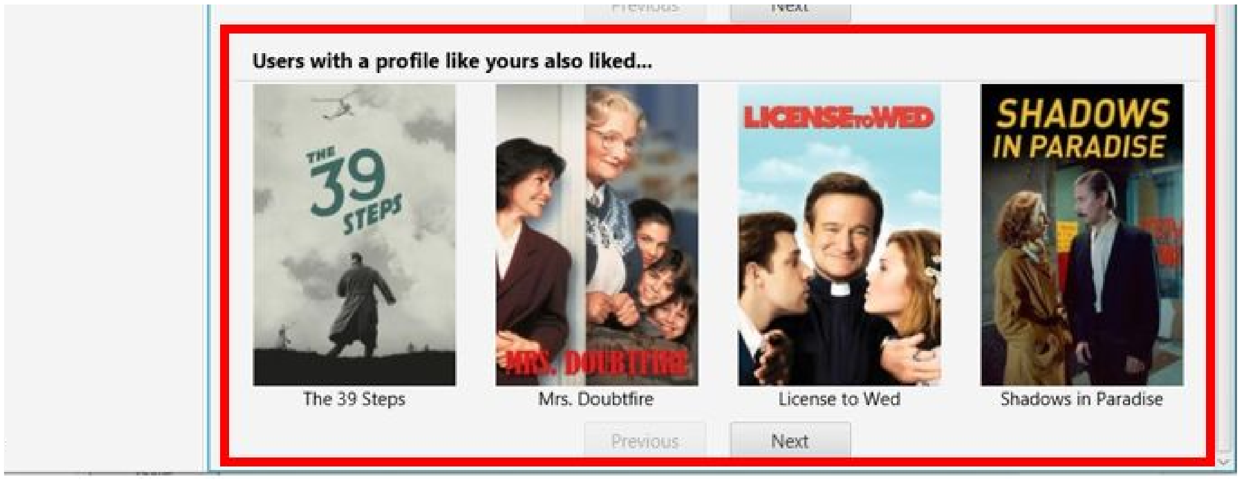


Username: Marie
Gender: Female
Email: marie.giannoni@isen.yncrea.fr
Date of birth: 13/02/1996

3. The user uses the scroll bar to go down in the page.



4. The recommendations based on genres are displayed on the section "Users with a profile like yours also liked...".



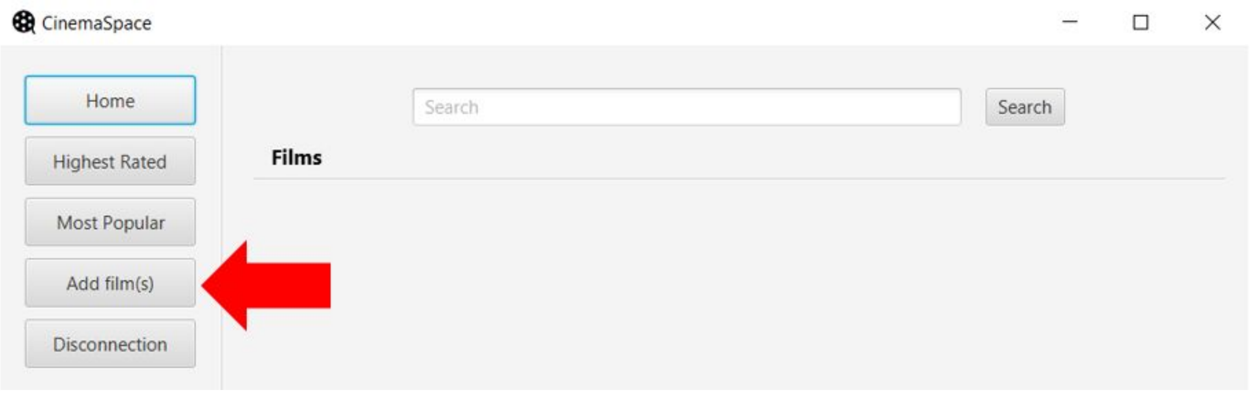
6.2.2 Administrator

Add a film or a collection of films

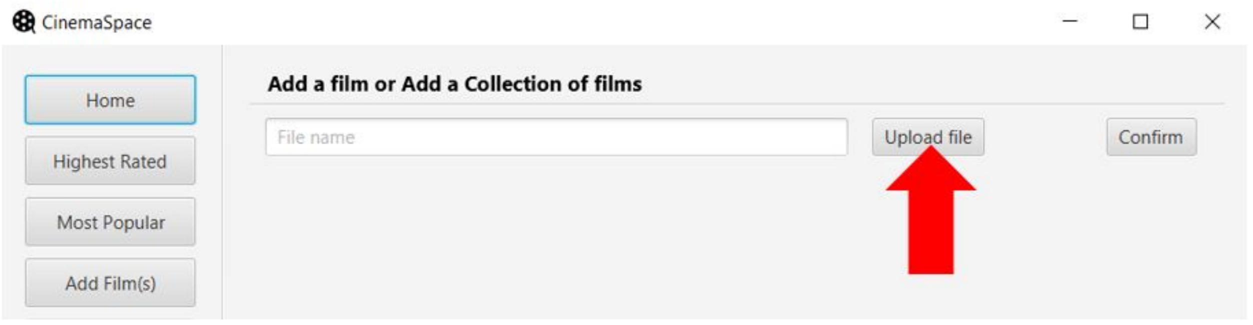
Preconditions: The user must have an account with the privilege of an administrator. The user must be logged in.

Steps:

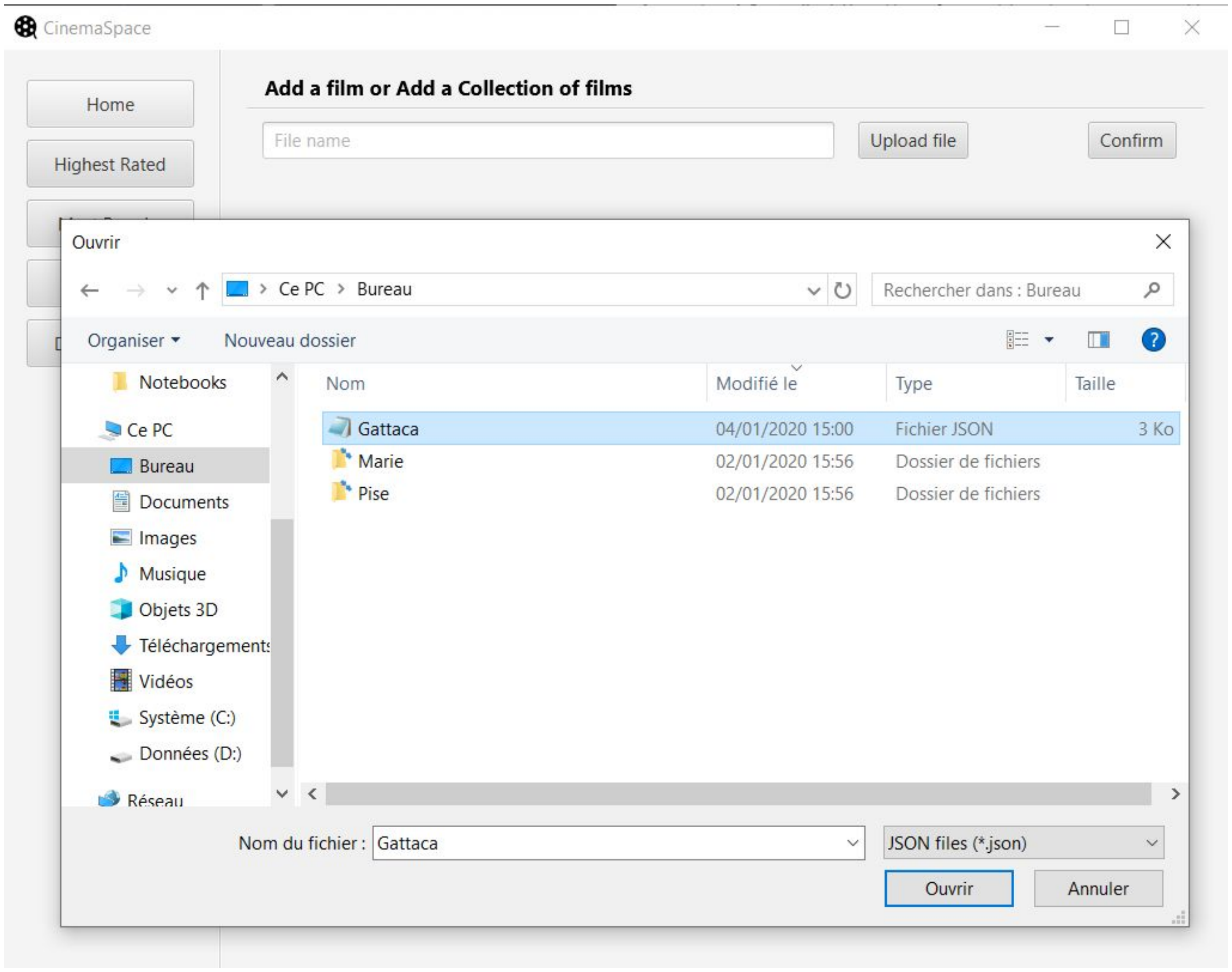
1. The user clicks on the "Add film(s)" button.



2. The user clicks on the "Upload file" button.



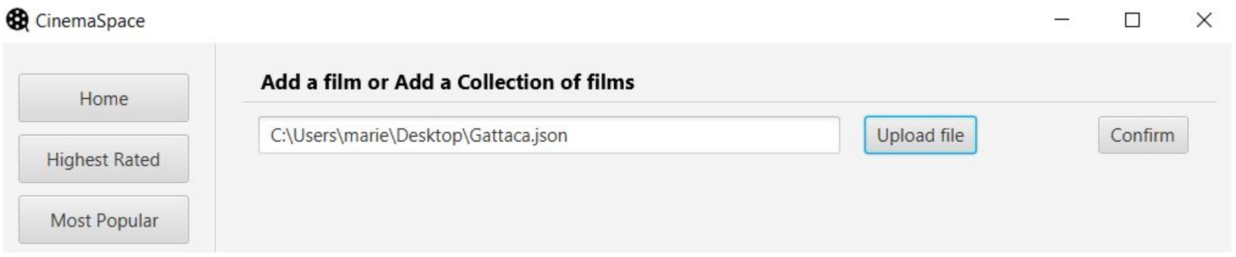
3. The user selects a JSON file with one or more films to add.



Nom	Modifié le	Type	Taille
Gattaca	04/01/2020 15:00	Fichier JSON	3 Ko
Marie	02/01/2020 15:56	Dossier de fichiers	
Pise	02/01/2020 15:56	Dossier de fichiers	

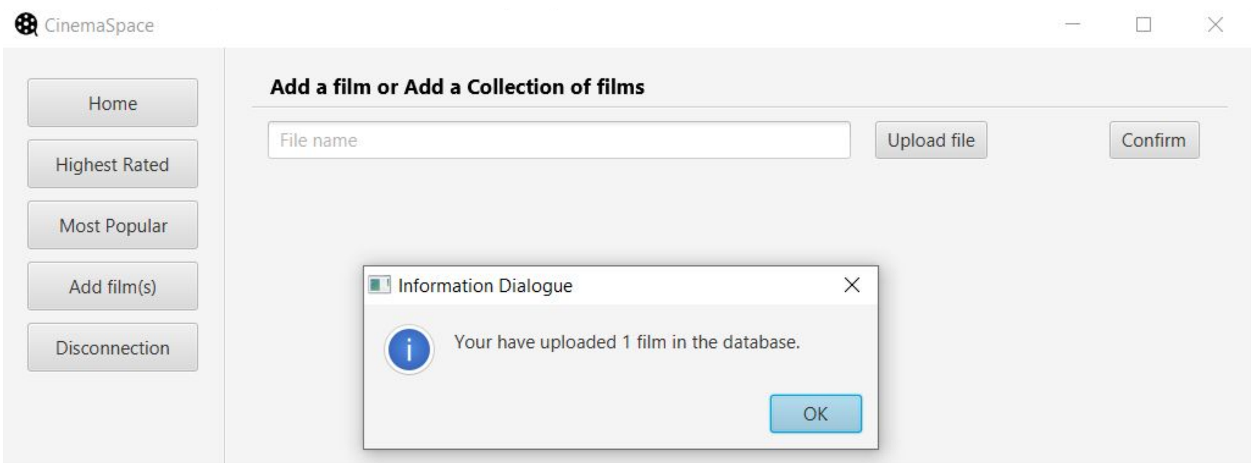
Nom du fichier : Gattaca

4. The user clicks on the “Confirm” button.



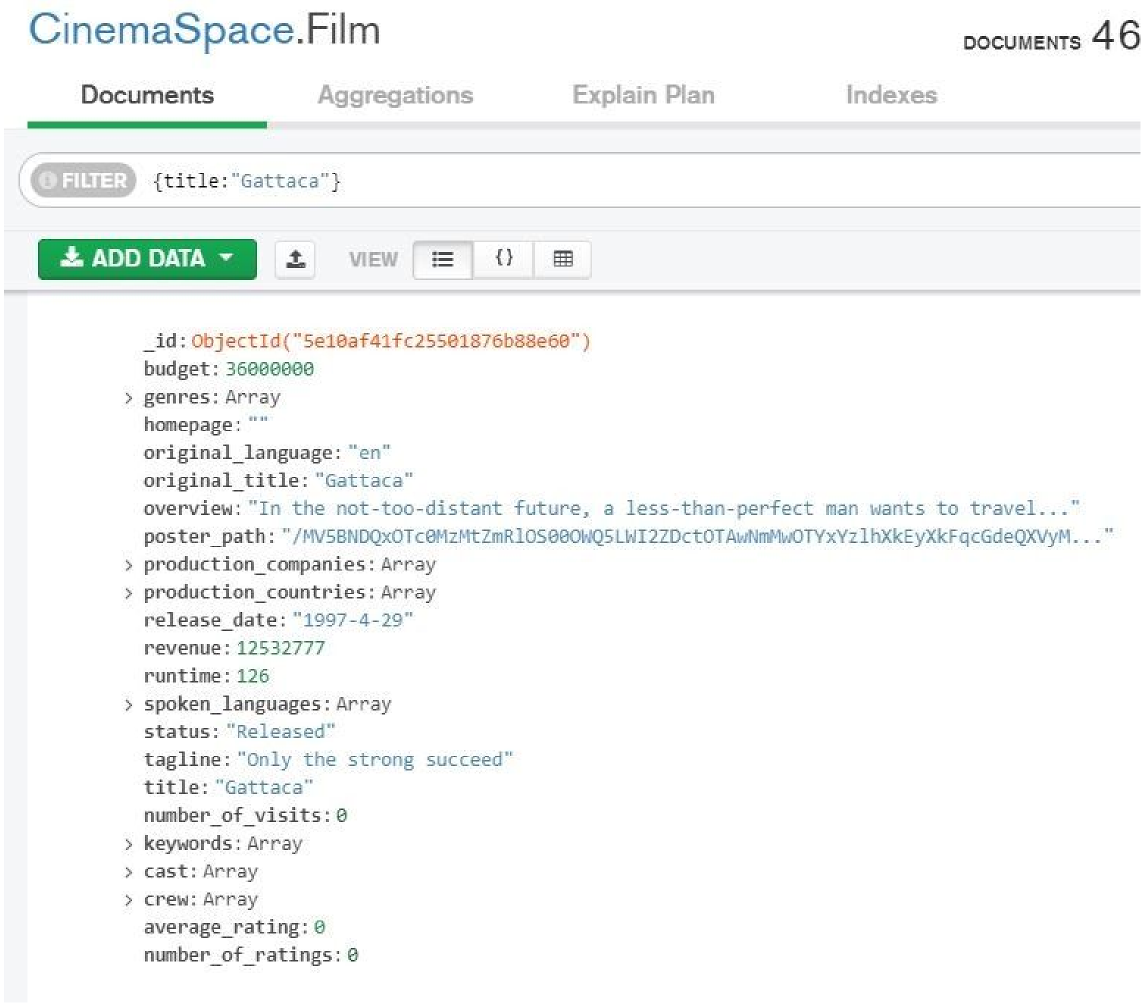
C:\Users\marie\Desktop\Gattaca.json

5. An Information Dialogue informs that the film or the collection of films were added.



Your have uploaded 1 film in the database.

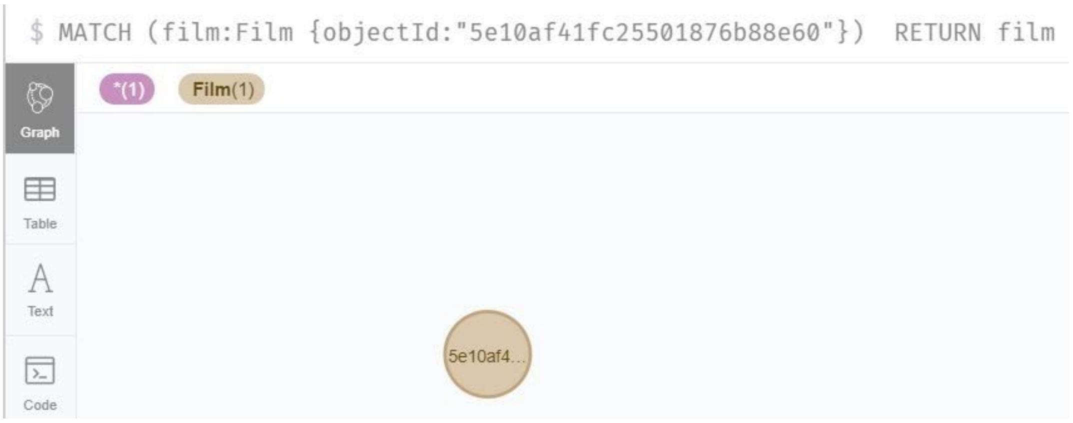
Database status after the insertion - MongoDB



```
{title:"Gattaca"}
```

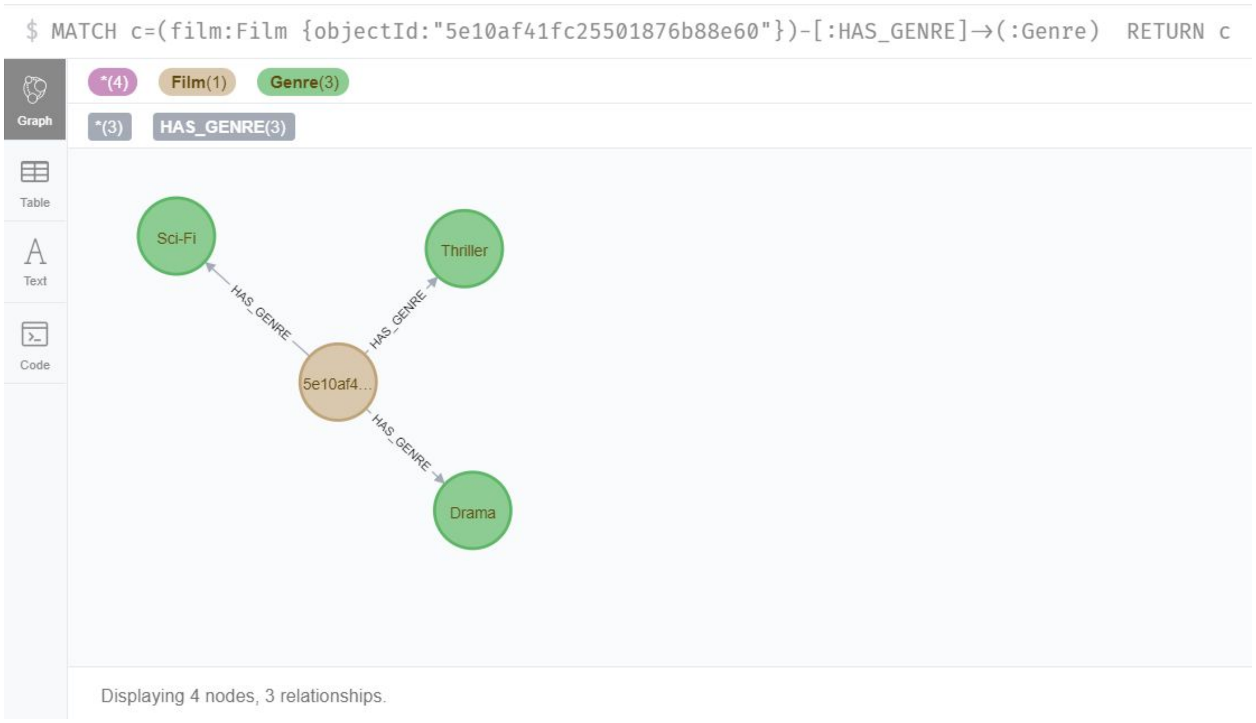
```
  _id: ObjectId("5e10af41fc25501876b88e60")
  budget: 36000000
> genres: Array
  homepage: ""
  original_language: "en"
  original_title: "Gattaca"
  overview: "In the not-too-distant future, a less-than-perfect man wants to travel..."
  poster_path: "/MV5BNDQxOTc0MzMtZmRlOS00OWQ5LWI2ZDctOTAwNmMwOTYxYzlhXkEyXkFqcGdeQXVyM..."
> production_companies: Array
> production_countries: Array
  release_date: "1997-4-29"
  revenue: 12532777
  runtime: 126
> spoken_languages: Array
  status: "Released"
  tagline: "Only the strong succeed"
  title: "Gattaca"
  number_of_visits: 0
> keywords: Array
> cast: Array
> crew: Array
  average_rating: 0
  number_of_ratings: 0
```

Database status after the insertion - Neo4j



```
$ MATCH (film:Film {objectId:"5e10af41fc25501876b88e60"})  RETURN film
```

With the genres associated to the inserted film:



```
$ MATCH c=(film:Film {objectId:"5e10af41fc25501876b88e60"})-[:HAS_GENRE]→(:Genre)  RETURN c
```

Displaying 4 nodes, 3 relationships.