

# UNIVERSITÀ DI PISA

Large-Scale and Multi-structured Databases - Project:  
Rotten Movies

Fabio Piras, Giacomo Volpi, Guillaume Quint

Academic Year: 2022/2023

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Feasibility Study</b>	<b>3</b>
1	Dataset analysis and creation . . . . .	3
2	Analysis result . . . . .	4
<b>3</b>	<b>Design</b>	<b>5</b>
1	Main actors . . . . .	5
2	Functional requirements . . . . .	5
3	Non functional requirements . . . . .	6
4	Implementation regarding the CAP theorem . . . . .	7
5	Use cases . . . . .	8
6	Class analysis . . . . .	9
<b>4</b>	<b>Document database</b>	<b>10</b>
1	Collection composition . . . . .	10
1 . 1	Movie collection . . . . .	10
1 . 2	User collection . . . . .	11
2	Indexes . . . . .	12
2 . 1	Movie collection . . . . .	13
2 . 2	User collection . . . . .	13
3	Partition and replicas . . . . .	13
4	Aggregations . . . . .	14
4 . 1	Return the best years by top critic and user ratings . . . . .	14
4 . 2	Return the best genres by top critic and user ratings . . . . .	14
4 . 3	Return the best production houses by top critic and user ratings . . . . .	15
4 . 4	Given a movie count the review it has received by each month . . . . .	16
4 . 5	Given a user count the review he/she has made divided by genres . . . . .	17
4 . 6	Return the number of user divided by an age bucket . . . . .	17
5	Sharding considerations . . . . .	18
<b>5</b>	<b>Software Architecture</b>	<b>19</b>
<b>6</b>	<b>Repository Structure</b>	<b>20</b>
<b>7</b>	<b>Application Structure</b>	<b>21</b>
1	Modules and code organization . . . . .	21
2	Managing consistency between MongoDB and Neo4j . . . . .	24
2 . 1	Insert movie . . . . .	24

# Chapter 1

## Introduction

Rotten Movies is a web service where users can keep track of the general liking for movies, they can also share their thoughts with the world after signing up. In addition users can follow renowned top critic to be constantly updated with their latest reviews.

Guest users, as well as all others, can browse or search movies based on filters like: title, year of release and personnel who worked in it. They can also view a Hall of Fame for the most positive review genres, production houses and years in which the best movies were released.

For this project, the application has been developed in Java with the Spring framework and Thymeleaf as templating engine to implement a web GUI. The application uses a document database to store the main information about movies, users, reviews and personnel; a graph database is instead used to keep track of the relationship between normal users and top critics in terms of who follows who and between the movies and the users who reviewed it.

# Chapter 2

## Feasibility Study

The very first step was to perform a look up for what type of data we needed to create the application and how to handle it by performing a feasibility study.

### 1 Dataset analysis and creation

Initially we searched online for available dataset, we ended up with six different files coming from Kaggle and imdb with a combined storage of 4.18 GB:

- title\_principal.tsv
- name\_basic.tsv
- title\_basic.tsv
- title\_crew.tsv
- poster\_URL.csv
- rotten\_movies.csv
- rotten\_reviews.csv

The first three were found on the imdb site containing general data about the title of the entries, type (not all were movies), cast, crew and other useful information. The fourth one came from Kaggle, and contains movies posters urls. The last two also came from Kaggle and they contain data scraped from the rotten tomatoes site regarding movie rating and their reviews. All of these dataset were organized in a relational manner.

**Initial steps and creation of the movie collection** After a general look it was clear that we needed to process the data to obtain a lighter dataset by deciding what to keep based on our needs and specifications; we decided to use Python as programming language to trim the original files, this was also achieved through the use of Google Colab.

We started by taking all the tsv files and transforming them into a single file, we performed a join operation on all of them based on their id and then discarded the useless information (Appendix ??). After that we performed the same operation on the csv files coming from Rotten Tomatoes (Appendix ??). We then proceeded to join the two files based on the title of the movies (Appendix ??).

Unfortunately we noticed that after the join there were more rows for the same movie, in fact, due to the relational nature of the first dataset, we had a different entry for personnel on

each row, so we rollback to before the join, but this time we collapsed all the information in a single row (Appendix ??).

Due to the variability of the data in the string fields, like the content of the reviews, we decided to perform an escape on various elements to avoid crashes and failures during the import into the DBSM (Appendix ??). We also had to do a similar process of normalization for the date fields (Appendix ??)

Finally we achieved our goal of having a JSON file for the movie collection, the file occupied a space of 270MB.

**Creation of the user collection** The following step was to generate a collection for the user. This was achieved by starting from the movie collection: for each different review author we generated a document, later to be placed in a single JSON file of which the total storage size was 86,6 MB. This step was performed directly on the Mongo shell (Appendix ??).

## 2 Analysis result

With the dataset for the document database ready we finally had a better understanding on how to shape the models and the relative functionality in the application code. We will discuss later of the design of the collections and the methods used to interact with them. The same goes for the graph database of which the structure was heavily influenced by the one in the MongoDB

# Chapter 3

## Design

### 1 Main actors

As already mentioned in the introduction we have mainly three actors: guest users and registered users. The latter can be divided between normal users and top critics. In addition there is an admin actor whose main role is to oversee the entire service.

### 2 Functional requirements

This section describes the functional requirements that need to be provided by the application in regards of the actors:

- Guest (Unregistered) Users can:

- login/register into the service
- search movies via the search bar and other filters
- view movies, their details and relative reviews
- view the personal page of the author of a selected *top critic* review
- view the different halls of fame

- Normal users can:

- logout from the service
- search movies via the search bar and other filters
- view movies, their details and relative reviews
- write a review for a selected film
- view the personal page of the author of a selected *top critic* review
- view the different halls of fame
- follow/unfollow a *top critic* user
- view the feed of latest reviews from the followed *top critics*
- view a suggestion feed for *top critics* to follow
- view the history of its own reviews
- modify its own reviews
- delete its own reviews

- change its account information
- Top Critics can:
  - logout from the service
  - search movies via the search bar and other filters
  - view movies, their details and relative reviews
  - write a top critic review for a selected film
  - view the different halls of fame
  - view the history of its own top critic reviews
  - modify its own top critic reviews
  - delete its own top critic reviews
  - change its account information
  - see the number of its followers

- Admin users can:

- logout from the service
- search movies via the search bar and other filters
- view movies, their details and relative reviews
- view the different halls of fame
- modify film details
- add/remove films
- browse *users* and *top critics*
- ban *users* and *top critics*
- register new *top critics*
- perform analytics on the user base

### 3 Non functional requirements

The following section lists non functional requirements for the application.

- The system must encrypt users' passwords
- The service must be built with the OOP paradigm
- The service must be implemented through a responsive website
- Avoidance of single point of failure in data storage
- High availability, accepting data displayed temporarily in an older version

## 4 Implementation regarding the CAP theorem

We will now discuss on how we decided to tackle the CAP theorem issue. We determined that the application is a read-heavy one where we expect that the number of read transactions are by far more frequent than write operations, so we decided that our application should prioritize high availability, low latency and should be capable of withstanding network partitions. In reference of figure 3.1 it is clear that we moved towards a AP approach in spite of temporarily data inconsistency.

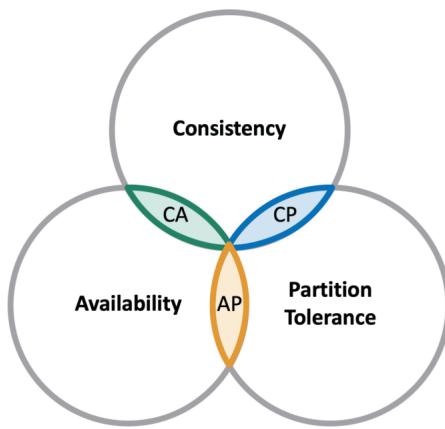
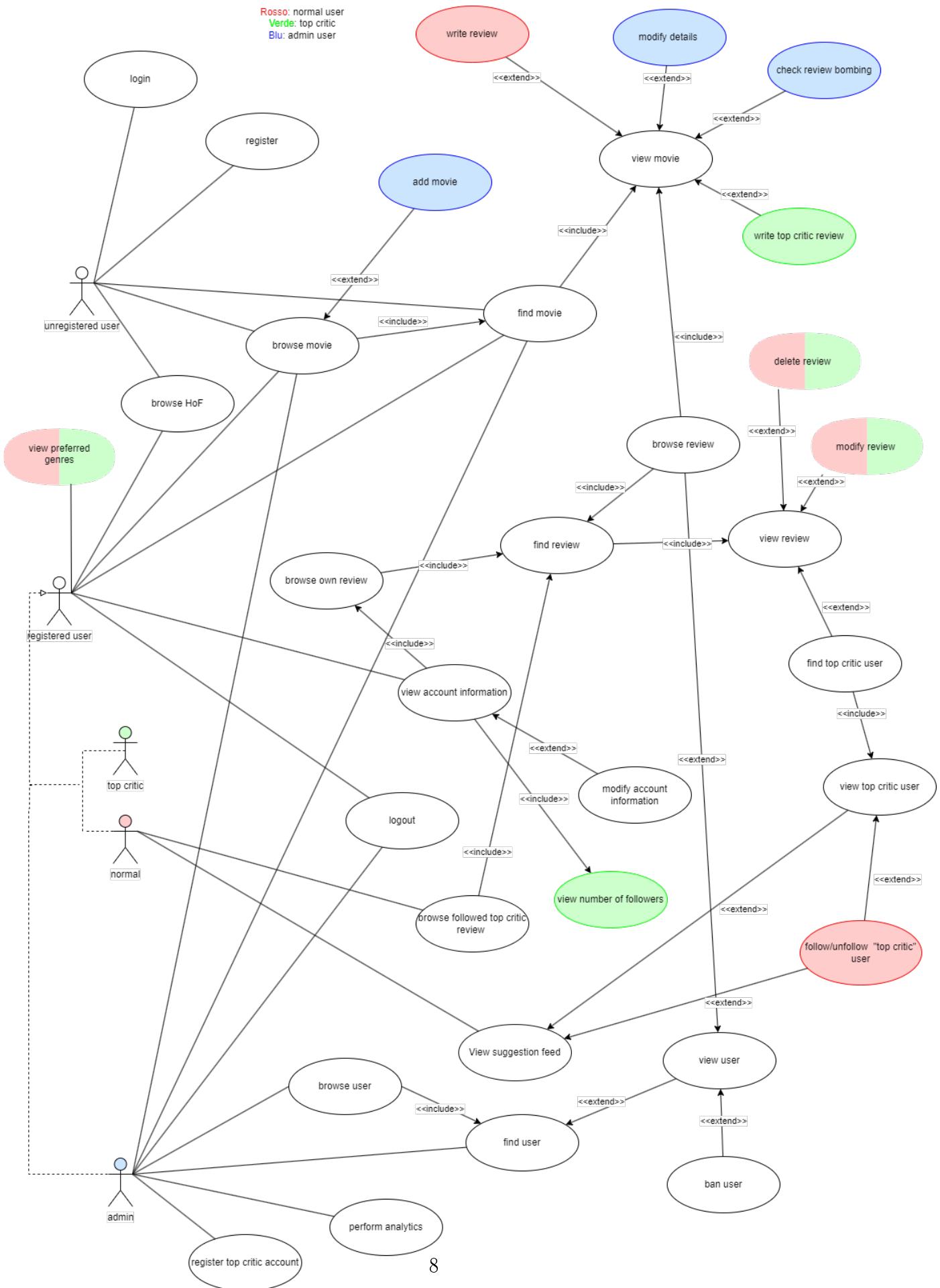


Figure 3.1: CAP theorem diagram

In order to guarantee the requirements of high availability, we decided to accept the cases in which data shown to the user could be not up to date to the latest version in the database.

## 5 Use cases



The *perform analytics* use case groups all of the following direct use cases:

- Group user base by age
- Most active users and top critics
- Most followed top critics

## 6 Class analysis

In this section we shall discuss the design of the various classes and how they relate to each other

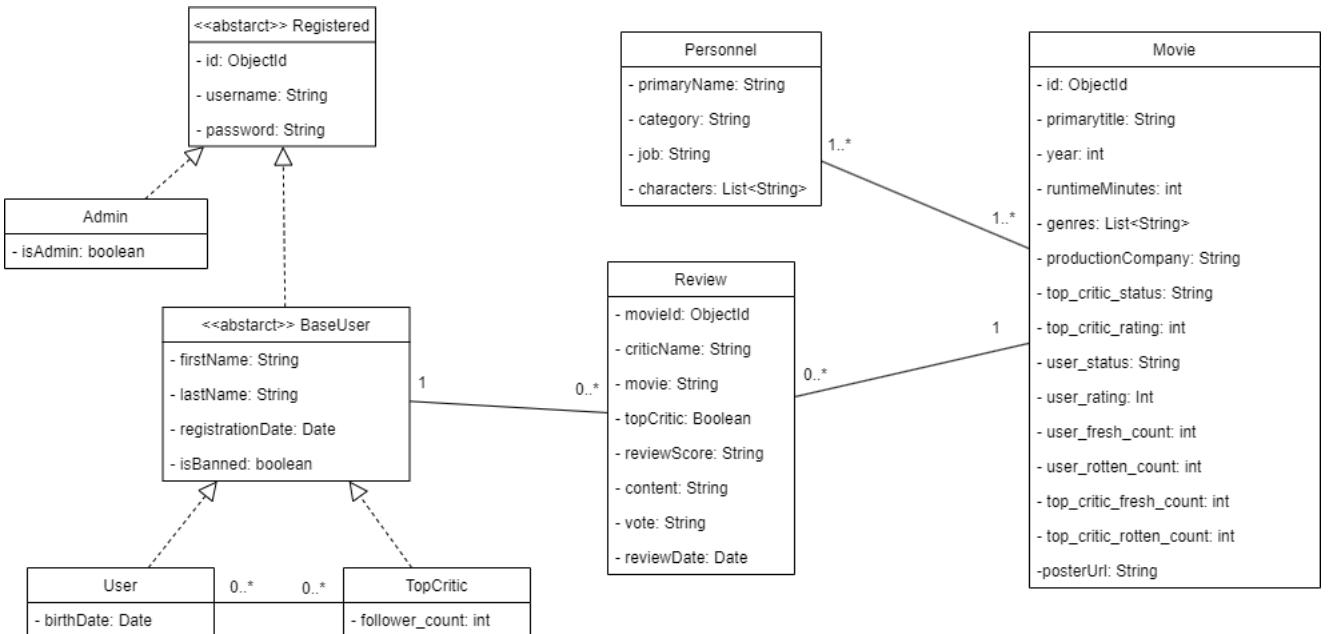


Figure 3.3: Class Diagram

The diagram expresses the following relationships between the entities.

- a `BaseUser` can write from zero to many reviews
- a `Review` can be written by a single `BaseUser` for a single `Movie`
- a `Movie` can have zero to many reviews and can have from 1 to many `Personnel` working in it
- a `Personnel` can work in 1 to many movies

# Chapter 4

## Document database

In this chapter we will discuss the organization of the document database and how we handled the replicas. We decided to use MongoDB as DBMS for the document database for the purpose of storing the main information for movies, users, reviews and personnel. In MongoDB we created the following two collections:

- movie
- user

Inside the movie collection are embedded the documents for the reviews and personnel

### 1 Collection composition

#### 1 .1 Movie collection

The following is the composition of a movie document in the collection.

```
1 {
2     "_id" : ObjectId(<<id_field>>),
3     "primaryTitle": "The first ever movie",
4     "year": 1989,
5     "runtimeMinutes": 70,
6     "genres": ["Crime", "Drama", "Romantic"],
7     "productionCompany": "Dingo Picture Production" ,
8     "personnel": [
9         {
10            "name": "John Doe",
11            "category": "producer",
12            "job": "writer"
13        },
14        {
15            "name": "Christopher Lee",
16            "category": "actor",
17            "character": ["The one"]
18        },
19        ...
20    ],
21    "top_critic_fresh_count": "4",
22    "top_critic_rotten_count": 0,
23    "user_fresh_count": 14,
```

```

24     "user_rotten_count": 1,
25     "top_critic_rating": 100,
26     "top_critic_status": "Fresh",
27     "user_rating": 93,
28     "user_status": "Fresh",
29     "review":
30     [
31         {
32             "critic_name": "AntonyE",
33             "review_date": "2018-12-07",
34             "review_type": "Rotten",
35             "top_critic": false,
36             "review_content": "I really didn't like it!",
37             "review_score": "1/10"
38         },
39         {
40             "critic_name": "AntonyE",
41             "review_date": "2015-12-07",
42             "review_type": "Fresh",
43             "top_critic": true,
44             "review_content": "I really liked it!",
45             "review_score": "A-"
46         },
47         ...
48     ]
49 }
```

Listing 4.1: Movie document

As previously stated the field personnel and review are arrays of embedded documents.

## 1 .2 User collection

The following is the composition of a user document in the collection.

```

1 {
2     "_id"                  : ObjectId(<<id_field>>),
3     "username"              : "AntonyE",
4     "password"              : "hashed_password",
5     "firstName"             : "Anton",
6     "lastName"              : "Ego",
7     "registration_date"    : "2019-06-29",
8     "date_of_birth"         : "2002-07-16",
9     "last3Reviews":
10    [
11        {
12            "_id"                  : ObjectId(<<id_field>>),
13            "primaryTitle"        : "Star Wars: A new Hope",
14            "review_date"          : "2018-12-07",
15            "review_type"          : "fresh",
16            "top_critic"           : false,
17            "review_content"       : "I really liked it!",
18            "vote"                 : "8/10"
```

```

19     },
20     {
21         "_id" : ObjectId(<<id_field>>),
22         "primaryTitle" : "Ratatouille",
23         "review_date" : "2019-02-17",
24         "review_type" : "rotten",
25         "top_critic" : false,
26         "review_content" : "The director was also
27             ↵ controlled by a rat",
28         "vote" : "D+"
29     },
30     {
31         "_id" : ObjectId(<<id_field>>),
32         "primaryTitle" : "300",
33         "review_date" : "2020-02-15",
34         "review_type" : "fresh",
35         "top_critic" : false,
36         "review_content" : "Too much slow motion",
37         "vote" : "3.5/5"
38     },
39     "reviews": [
40     {
41         "movie_id" : ObjectId(<<id_field>>),
42         "primaryTitle" : "Evidence",
43         "review_index": 11
44     },
45     ...
46 ]
47
48 }

```

Listing 4.2: User document

It is important to notice that in this collection we have two fields for the reviews, *last3Reviews* and *reviews* which present different ways of storing data for the same underling entity. As suggested by the name itself *last3Reviews* contains the last three review in the form of an embedded document meanwhile *reviews* contains all the reviews made by a user in the form of document linking towards the movie for which they were written; in particular, the link is formed by the id of the movie, the title and the position in which the review can be found in the array *review* of the movie document. This structure was chosen so that we could avoid a full replica of the reviews in both collections and, at the same time, avoiding to perform join operation for those reviews that are more frequently checked, which are the most recent ones. The idea of creating a separated collection for reviews was immediately discarded because it would have implied a design for the document database that resembles a third normal form.

## 2 Indexes

In order to provide the best execution speed in search queries we use indexes. In particular we focus on the application part that is available also without registering to the site, like Hall of Fame and search movies functionalities. Without indexes we need a collection scan in user and movie collections to find the right document.

## 2 .1 Movie collection

- primaryTitle
- year
- genres
- top\_critic \_rating
- user\_rating

```
1 // Dimension is expressed in Kb
2 {
3     _id_: 228,
4     primaryTitle_1: 292,
5     year_1: 92,
6     genres_1: 180,
7     'personnel.primaryName_1': 1680,
8     top_critic_rating_1: 80,
9     user_rating_1: 80
10 }
```

Listing 4.3: Movie collection indexes

## 2 .2 User collection

- username
- date\_of\_birth

```
1 // Dimension is expressed in Kb
2
3 { _id_: 180, username_1: 168, date_of_birth_1: 100 }
```

Listing 4.4: User collection indexes

We decided to use these indexes because they provide a jump in term of speed in search queries without occupying much space. We accept the fact that writes are slower because our application is read-intesive. We consider the idea of using an index on personnel.primaryName but the cost in term of space was higher than the potential benefit, so we discarded it. For a full analytic report for the indexes, see appendix. (Appendix ??)

## 3 Partition and replicas

To ensure high availability we deploy a cluster of replicas (3). We install and configure MongoDB on all machines, with these priorities

1. 172.16.5.26 (primary)
2. 172.16.5.27 (secondary)
3. 172.16.5.28 (secondary)

We decided to use **nearest read preference** and **W2 write preference**. In fact we can tolerate that users see temporarily an old version of data with a 33% chance.

## 4 Aggregations

In this section we shall discuss the different aggregations that the application will implement, the values between «» represent a value passed by an above level.

### 4 .1 Return the best years by top critic and user ratings

Mongo shell

```
1 db.movie.aggregate([
2     {$group:
3         {
4             _id: "$year",
5             topCriticRating:{ $avg: "$top_critic_rating" },
6             userRating:{ $avg: "$user_rating" },
7             count:{ $sum:1 }
8         }
9     },
10    {$match: { count: { $gte:<<i>>} } },
11    { $sort :{ [ topCriticRating / userRating ]:-1 } },
12    { $limit : <<j>> }
13 ])
```

Listing 4.5: bestYear.js

### Java implementation

```
1 AggregateIterable<Document> aggregateResult = collection.aggregate(
2         Arrays.asList(
3             Aggregates.group("$year",
4                 avg("top_critic_rating", "$top_critic_rating"
5                     ),
6                     avg("user_rating", "$user_rating"),
7                     sum("count",1)),
8                     Aggregates.match(gte("count",numberOfMovies)),
9                     Aggregates.sort(opt.getBsonAggregationSort()),
10                    Aggregates.limit(Constants.
11 HALL_OF_FAME_ELEMENT_NUMBERS)
12                )
13        );
```

Listing 4.6: MovieMongoDBDAO.java

### 4 .2 Return the best genres by top critic and user ratings

Mongo shell

```
1 db.movie.aggregate([
2     {
3         $unwind: "$genres"
4     },
5     {$group:
6         {
7             _id: "$genres",
8             topCriticRating:{ $avg: "$top_critic_rating" },
9             userRating:{ $avg: "$user_rating" },
10            }
11        }
12    ]);
```

```

10             count:{ $sum:1 }
11         }
12     },
13     {$match:{ count:{ $gte:<<i>>} } },
14     { $sort :{[ topCriticRating / userRating ]:-1} } ,
15     { $limit:<<j>> }
16   ]
)

```

Listing 4.7: bestGenres.js

## Java implementation

```

1 AggregateIterable<Document> aggregateResult = collection.aggregate(
2         Arrays.asList(
3             Aggregates.unwind("$genres"),
4             Aggregates.group("$genres",
5                 avg("top_critic_rating", "$top_critic_rating",
6                     avg("user_rating", "$user_rating"),
7                     sum("count", 1)),
8                 Aggregates.match(gte("count", numberOfMovies)),
9                 Aggregates.sort(opt.getBsonAggregationSort()),
10                Aggregates.limit(Constants.
11                    HALL_OF_FAME_ELEMENT_NUMBERS)
12            )
);

```

Listing 4.8: MovieMongoDBDAO.java

## 4 .3 Return the best production houses by top critic and user ratings

### Mongo shell

```

1 db.movie.aggregate([
2     {$group:
3         {
4             _id: "$production_company",
5             topCriticRating:{ $avg:"$top_critic_rating" },
6             userRating:{ $avg:"$user_rating" },
7             count:{ $sum:1 }
8         }
9     },
10    {$match:{ count:{ $gte:<<i>>} } },
11    { $sort :{[ topCriticRating / userRating ]:-1} } ,
12    { $limit:<<j>> }
13 ]
)

```

Listing 4.9: bestProduction Houses.js

## Java implementation

```

1 AggregateIterable<Document> aggregateResult = collection.aggregate(
2         Arrays.asList(
3             Aggregates.group("$production_company",
4                 avg("top_critic_rating", "$top_critic_rating"
5
),
6
)
);

```

```

5                               avg("user_rating", "$user_rating") ,
6                               sum("count",1)) ,
7                               Aggregates.match(gte("count",numberOfMovies)) ,
8                               Aggregates.sort(opt.getBsonAggregationSort()) ,
9                               Aggregates.limit(Constants.
10                             HALL_OF_FAME_ELEMENT_NUMBERS)
11                           )

```

Listing 4.10: MovieMongoDBDAO.java

## 4 .4 Given a movie count the review it has received by each month

### Mongo shell

```

1 db.movie.aggregate([
2   {
3     $match:{ id:<<"id">>}
4   },
5   {$unwind:"$review"} ,
6   {
7     $group:
8     {
9       _id:{ year:{ $year:"$review.review_date"} , month:{ $month:"$review.
10      review_date" } } ,
11       count:{ $sum:1 }
12     }
13   },
14   {$sort:{ _id:1 }}
15 ])

```

Listing 4.11: movieCount.js

### Java implementation

```

1 Document yearDoc = new Document("year",new Document("$year","$review.
2   review_date"));
3   Document monthDoc = new Document("month",new Document("$month",
4   "$review.review_date"));
5   ArrayList<Document> test=new ArrayList<>();
6   test.add(yearDoc);
7   test.add(monthDoc);
8   AggregateIterable<Document> aggregateResult = collection.aggregate(
9     Arrays.asList(
10       Aggregates.match(eq("_id",id)),
11       Aggregates.unwind("$review"),
12       Aggregates.group(test,
13         sum("count",1)),
14       Aggregates.sort(Sortsascending("_id"))
15     )
16   );

```

Listing 4.12: MovieMongoDBDAO.java

## 4 .5 Given a user count the review he/she has made divided by genres

### Mongo shell

```
1 db.movie.aggregate([
2     {$match:
3         {"review.critic_name":{ $eq:<<"name">>}}
4     },
5     {$unwind: "$genres"},
6     {$group:{_id:"$genres", count:{$sum:1}}},
7     {$sort:{count:-1}},
8     {$limit: <<n>>}
9 ])
])
```

Listing 4.13: genreCount.js

### Java implementation

```
1 AggregateIterable<Document> aggregateResult = collectionMovie.aggregate(
2         Arrays.asList(
3             Aggregates.match(eq("review.critic_name",username)),
4             Aggregates.unwind("$genres"),
5             Aggregates.group("$genres",
6                 sum("count",1)),
7             Aggregates.sort(Sorts.descending("count")),
8             Aggregates.limit(Constants.
9                 HALL_OF_FAME_ELEMENT_NUMBERS)
10            )
11        );
12    );
```

Listing 4.14: BaseUserMongoDBDAO.java

## 4 .6 Return the number of user divided by an age bucket

### Mongo shell

```
1 db.user.aggregate([
2     {
3         $match:{ "date_of_birth":{ $exists:true}}
4     },
5     {
6         $bucket:
7             {
8                 groupBy: { $year:"$date_of_birth"}, 
9                 boundaries: [<<values>>],
10                 output:
11                     {
12                         "population": { $sum:1}
13                     }
14             }
15     }
16 ])
])
```

Listing 4.15: userPopulation.js

## Java implementation

```
1 BucketOptions opt = new BucketOptions() ;  
2     ArrayList<Integer> buck=new ArrayList<>();  
3     opt.output(new BsonField("population",new Document("$.sum",1))) ;  
4     int bucketYear=1970;  
5     buck.add(bucketYear);  
6     while(bucketYear<=2010){  
7         bucketYear=(bucketYear+offset);  
8         buck.add(bucketYear);  
9     }  
10    AggregateIterable<Document> aggregateResult = collectionUser.  
aggregate(  
11        Arrays.asList(  
12            Aggregates.match(exists("date_of_birth")),  
13            Aggregates.bucket(new Document("$.year", "  
$date_of_birth"),buck,opt)  
14        )  
15    );
```

Listing 4.16: AdminMongoDBDAO.java

## 5 Sharding considerations

We consider the possibility of sharding the movie collection, with the benefit of much less space occupation, using id as a possible sharding key. However we realized that we search movies not only by id but also with their primary titles, their years, etc. With these type of queries we end up in a query flooding scenario, where every replica must be consulted in order to find the appropriate document.

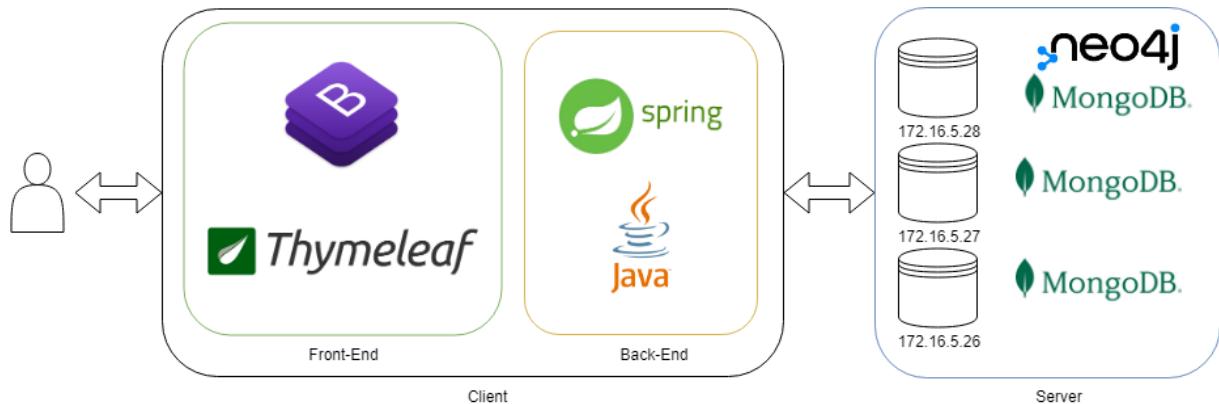
# Chapter 5

## Software Architecture

*Rotten Movies* is a web application developed in *Java* that implements the Model-View-Controller (MVC) paradigm through the use of the *Spring Framework*.

Specifically, the *view* layer is handled by the templating engine *Thymeleaf*, which enables the creation of custom web pages by interpolating data provided by the middle-ware with predefined html templates. Interface styling has been eased by the use of some predefined CSS classes defined in *Bootstrap 5.0*<sup>1</sup>

The back-end side of the application is supported by the document database *MongoDB* and the graph database *Neo4J*, which are accessed via the official mongo driver<sup>2</sup> and the neo4j driver<sup>3</sup> for Java, respectively



<sup>1</sup><https://getbootstrap.com/>

<sup>2</sup><https://www.mongodb.com/docs/drivers/java/sync/current/>

<sup>3</sup><https://neo4j.com/developer/java/>

# Chapter 6

## Repository Structure

# Chapter 7

## Application Structure

### 1 Modules and code organization

The picture below represent the packages in which the application is organized.

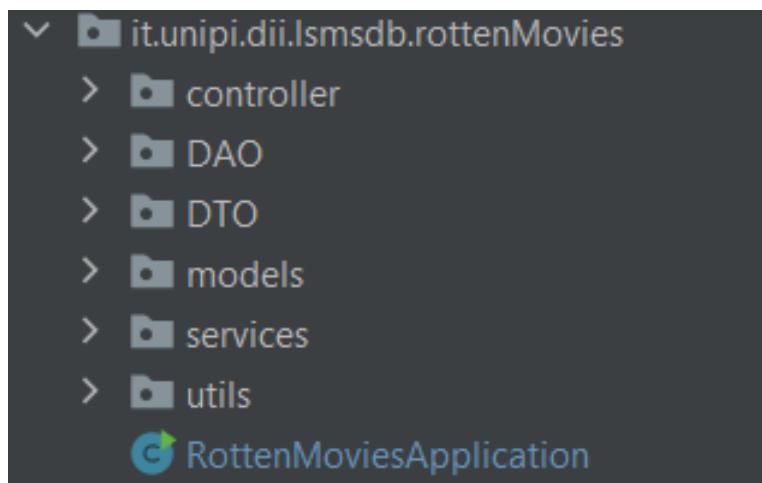


Figure 7.1: module organization

First of all we follow the reverse-domain convention for the root package, before passing to analyzing the source code, we put in the Appendix the pom.xml file for the dependency (Appendix ??).

- *controller* is responsible for handling the request to the various endpoint with the use of Spring

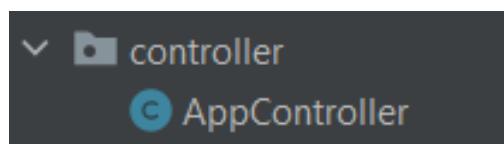


Figure 7.2: controller

- *DAO* (Data Access Object) is responsible for accessing the databases and retrieving the necessary object

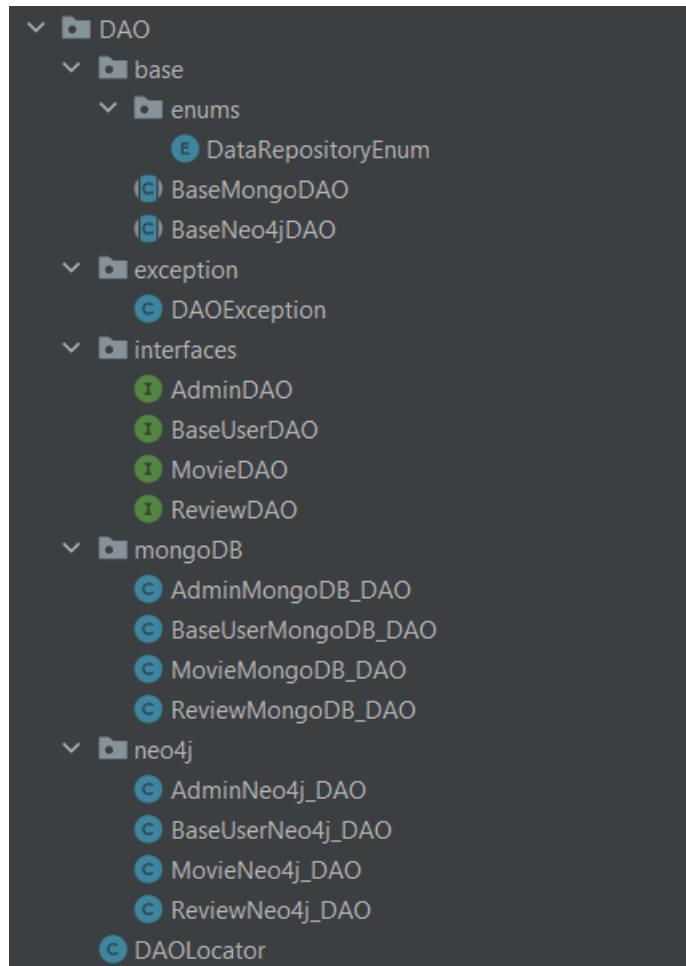


Figure 7.3: DAO

- *base* contains the classes responsible for handling the base connection to the DBs, *enums* is used as packet to differentiate the connection type in base of an enum for higher calls
- *exception* is responsible for generating and handling a custom exception invoked when trying to access the wrong database
- *interfaces* contains the various interfaces that map all the methods for accessing the databases differentiated in base of the general field for the operation, they extend the *AutoClosable* interface
- *mongoDB* handles the operations on the MongoDB for the different entities
- *neo4j* is the same as *mongoDB* but for the Neo4j database
- *DTO* (Data Transfer Object) presents all the classes that are used as container of the data passed between the service layer and the presentation layer

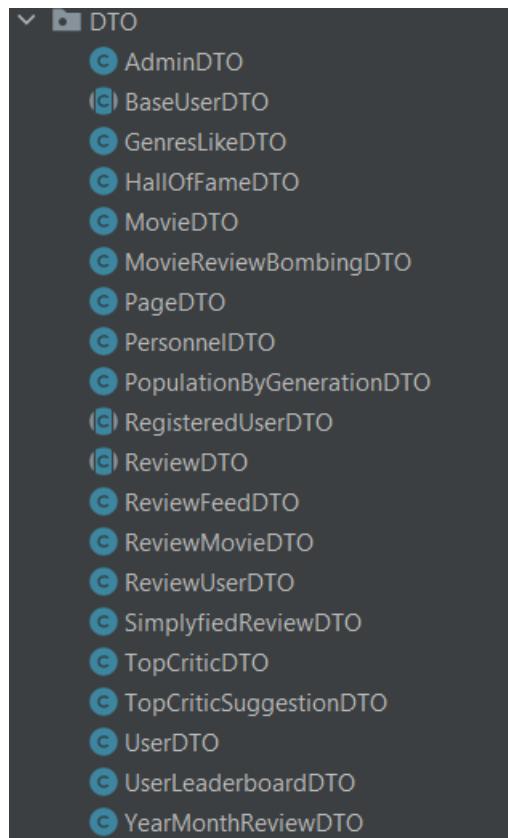


Figure 7.4: DTO

- *Models* presents all the classes that are mapped with the database organization. They all contains private field and have getters/setters.

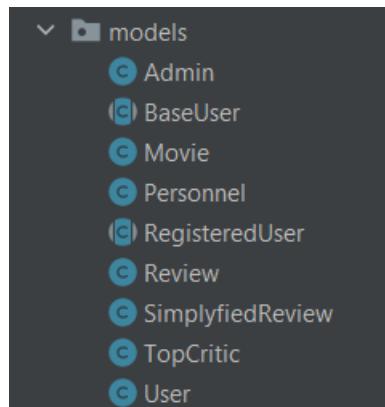


Figure 7.5: Models

- *Services* contains the middleware of our application. It is called in AppController function and interfaces with DAO classes.

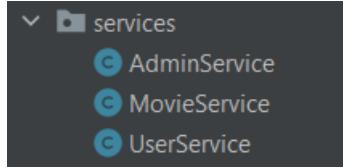


Figure 7.6: Services

- *Utils* provides utility methods to all packages. It includes password hashing, different possibility to sort/project results in Neo4j and MongoDB

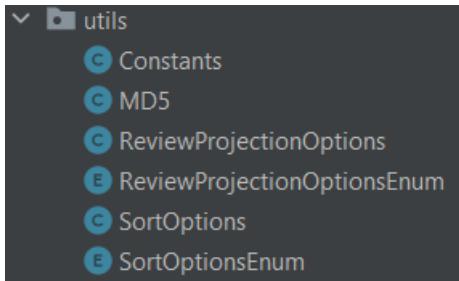


Figure 7.7: Utils

## 2 Managing consistency between MongoDB and Neo4j

Because we use two databases we need to manage consistency among them. An example on how it is managed is the addMovie method in MovieService. Here first we try to add a movie in MongoDB, if the Mongo operation is successfull we try to add the movie in Neo4j. If Neo4j fails we decided to roll-back the insert on MongoDB, deleting the movie added previously. This strategy is also adopted in add/update/delete operations.

### 2 .1 Insert movie

```

1  public ObjectId addMovie(String title) {
2      if (title == null || title.isEmpty()) {
3          return null;
4      }
5      Movie newMovie = new Movie();
6      newMovie.setPrimaryTitle(title);
7      ObjectId id = null;
8      try (MovieDAO moviedao = DAOLocator.getMovieDAO(DataRepositoryEnum.
MONGO)) {
9          id = moviedao.insert(newMovie);
10     } catch (Exception e) {
11         System.out.println(e);
12     }
13     if (id == null) {
14         return null;
15     }
16     newMovie.setId(id);
17     try (MovieDAO moviedao = DAOLocator.getMovieDAO(DataRepositoryEnum.
NEO4J)) {
18         id = moviedao.insert(newMovie);

```

```
19         } catch (Exception e) {
20             System.out.println(e);
21         }
22         if (id == null){ // roll back di mongo
23             try (MovieDAO moviedao = DAOLocator.getMovieDAO(
24                 DataRepositoryEnum.MONGO)) {
25                 moviedao.delete(newMovie);
26             } catch (Exception e) {
27                 System.out.println(e);
28             }
29         }
30     }
31 }
```

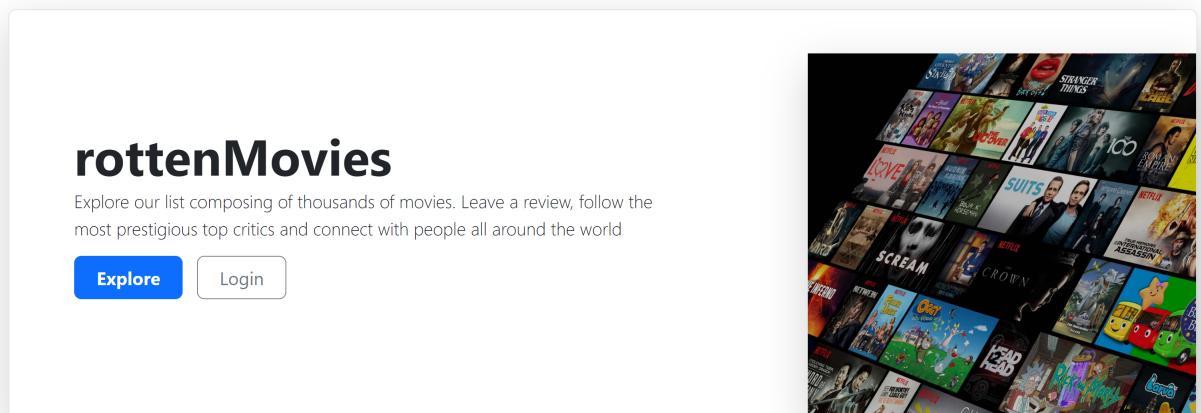
Listing 7.1: Test

# Chapter 8

## Instruction Manual

The application is accessible through a web browser and doesn't require an authentication to perform some basic navigation and scrolling of movies and reviews. To leave a review under a movie, follow top critics and receive personalized information, a user must log-in or sign-up to the service. Some notable and certified users may be registered by an administrator as a *Top Critic* which can be followed by other users and whose reviews are highlighted under a movie

### 1 Greeting Page



The landing page presents the option to jump directly to the main page for exploring movies and incites the new user to login or sign-up to the application.

 rotten Movies ≡

Title Search  
 Start Year Range  End Year Range   
 Personnel   
 Must include all names (separate by comma ',')  
 Genres   
 Must include all genres (separate by comma ',')

Sort By ▼  
 top critic rating  
 Ascending order

[Previous](#) 0 [Next](#)



**Image Not Found**

**On Dangerous Ground**  
1917 (WARNER BROTHERS PICTURES)

[Drama](#) [War](#)

[William A. Brady](#) [Lucien N. Andriot](#) [Gail Kane](#)  
[William Bailey](#) [Carlyle Blackwell](#) [Frances Marion](#)



**Image Not Found**

**Way Down East**  
1920 (Kino Lorber)

[Drama](#) [Romance](#)

[William A. Brady](#) [D.W. Griffith](#) [Lillian Gish](#)  
[Lowell Sherman](#) [Anthony Paul Kelly](#) [Lottie Blair Parker](#)

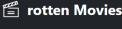


**Romeo and Juliet**  
1916 (Paramount Home Video)

[Drama](#)

[William Shakespeare](#) [John Webb Dillon](#) [Glen White](#)  
[J. Gordon Edwards](#) [Phil Rosen](#) [William Fox](#)

The exploring page offers multiple options to filter between all movies inside the database

 rotten Movies ≡

Avengers  
 Start Year Range  End Year Range   
 Personnel   
 Must include all names (separate by comma ',')  
 Genres   
 Must include all genres (separate by comma ',')

Sort By ▼  
 top critic rating  
 Ascending order

[Previous](#) 0 [Next](#)



**Avengers: Age of Ultron**  
2015 (Walt Disney Pictures)

[Action](#) [Adventure](#) [Sci-Fi](#)

[Robert Downey Jr.](#) [Joss Whedon](#) [Joe Simon](#)  
[Jack Kirby](#) [Stan Lee](#) [Mark Ruffalo](#) [Chris Evans](#)  
[Kevin Feige](#) [Chris Hemsworth](#) [Jim Starlin](#)

Upright (76%) Certified Fresh (75%)

[Login to Review](#) 141 mins



**Avengers: Infinity War**  
2018 (Walt Disney Pictures)

[Action](#) [Adventure](#) [Sci-Fi](#)

[Robert Downey Jr.](#) [Jack Kirby](#) [Stan Lee](#) [Mark Ruffalo](#)  
[Chris Evans](#) [Joe Russo](#) [Anthony Russo](#)  
[Christopher Markus](#) [Stephen McFeely](#) [Chris Hemsworth](#)

Upright (86%) Fresh (74%)

[Login to Review](#) 149 mins

It is possible to search by

- a string contained in the title
- a range of year in which the movie has been produced
- people that worked on the film. It is possible to build a list of names by employing a

comma (,) and stating if the movie must include all names or at least one of them

- genres. It is possible to build a list of genres by employing a comma (,) and stating if the movie must include all genres or at least one of them

The available sorting options are

- by top critic rating (the default)
- by user critic rating
- alphabetical order
- by date of production

and the user can opt for an ascending or descending sort.

At any point the user can click on the pills of genres or crew members to perform a search with that value.

If a user isn't logged-in, the button under a movie says 'Login to Review' and brings to the login page. Otherwise it becomes 'Review' and clicking on it allows the quick creation or editing of the review for that movie.

The screenshot shows the movie page for 'Avengers: Infinity War' on Rotten Tomatoes. At the top, there's a navigation bar with a magnifying glass icon and the text 'rotten Movies'. Below the title 'Avengers: Infinity War' are the release year '2018', runtime '149 min.', and studio 'Walt Disney Pictures'. A genre filter bar shows 'Action' (highlighted in blue), 'Adventure', and 'Sci-Fi'. The movie's plot summary is listed with bullet points: Robert Downey Jr. as Tony Stark/Iron Man, based on the Marvel comics by Stan Lee; Chris Evans as Steve Rogers/Captain America, director: Anthony Russo; and screenplays by Stephen McFeely, Mark Ruffalo as Bruce Banner/Hulk, director: Joe Russo, screenplay by Christopher Markus; and Chris Hemsworth as Thor, based on the Marvel comics by Jack Kirby. To the right of the summary is a large, vibrant movie poster for 'Avengers: Infinity War'. Below the poster, the release date 'APRIL 27' is visible. At the bottom of the page, there's a 'login to review this movie' button. The 'Reviews:' section below the poster displays three reviews from users: Josh Larsen (2018-04-25), Scott Menzel (2018-04-25), and Alan Zilberman (2018-04-25). Each review includes a snippet of the text, the user's name, the date, and the review score.

User	Date	Review	Score
Josh Larsen	(2018-04-25)	"...a business play forcing us to make yet another down payment on our collective Marvel mortgage."	Review Score: "2/4"
Scott Menzel	(2018-04-25)	"Be prepared to sweat, cry, and clench your armrest because Avengers: Infinity War is one hell of a ride."	Review Score: "9.5/10"
Alan Zilberman	(2018-04-25)	"If the Marvel Cinematic Universe aims to reshape popular entertainment, it needs to move beyond climax after climax with unremarkable, repetitive violence."	Review Score: ""

Selecting a movie allows to see a full description of it

**Reviews:**

Previous 2 Next

**Michael Sangiacomo (2018-04-25)**

"Avengers: Infinity War" is almost three hours of non-stop action on multiple battlefields leading to a conclusion that will forever change the Marvel cinematic universe."

Review  
Score: "A"

**Matthew Razak (2018-04-25)**

"This movie isn't actually about the Avengers. It's about Thanos."

Review  
Score: "8/10"

**Clay Cane (2018-04-25)**

"There is a fissure between critics and audiences, which I can admit. The movie is critic-proof. 'Avengers: Infinity War' is superhero porn, strictly for the fans, while 'Black Panther' was undeniably excellent, transcending the comic book fan base."

Review  
Score: "B-"

**Julian Roman (2018-04-25)**

"Infinity War is so gargantuan in scope, it almost becomes too much to digest. The smaller moments that have made the Marvel Cinematic Universe so enjoyable are lost in the spectacle."

Review  
Score: "3.5/5"

**Hannah Woodhead (2018-04-25)**

[View Profile](#)      Top Critic ★

"The high doesn't last, but it's dizzying all the same."

Review  
Score: "4/5"

**Diva Velez (2018-04-25)**

"AVENGERS: INFINITY WAR is nearly spread too thin by its massive amount of plot and characters. It redeems itself by being full of surprises and foreboding; focusing on the heroes' relationships and real peril in fierce, unrestrained action."

Review  
Score: "3.5/5"

**Fico Cangiano (2018-04-25)**

"Expectations surpassed. An ambitious, exhilarating and

Review

A selected movie allows also to read a paginated list of all its reviews

If a user isn't logged-in, the button under the movie says 'Login to Review this movie' and brings to the login page. Otherwise it becomes 'Review' and clicking on it allows the creation or editing of the review for that movie.



Please sign in

First Name	John
Last Name	Doe
Username	JDoe99
Password	*****
Birthday	20/07/1999

[Register](#)

[Already a User? Login](#)

User registration requires some basic information about the new user



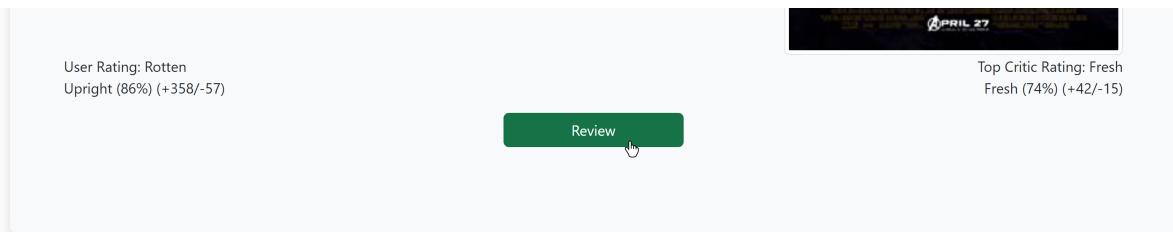
Please login

Username	JDoe99
Password	*****

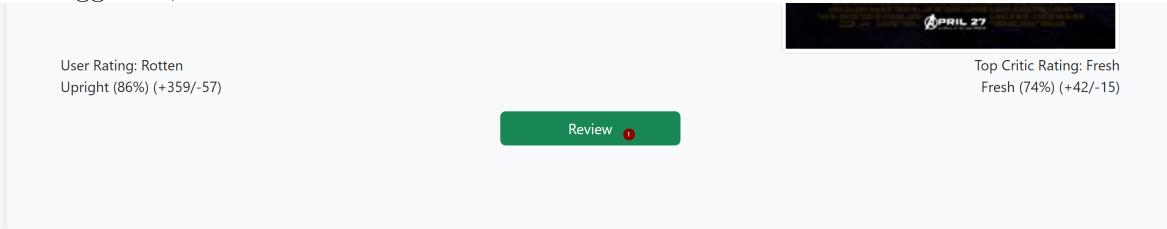
[Login](#)

[Don't have an account? Register](#)

If a user has already registered to the application, he/she can login with its credentials



After loggin-in, a user can leave a review under a movie



### Reviews:

[Previous](#) 0 [Next](#)

JDoe99 (2023-01-10)

Review  
Score: ""

Edit

If a user has never reviewed a movie, a new one is created and can be immediately modified

### Reviews:

[Previous](#) 0 [Next](#)

JDoe99 (2023-01-10)

Overcomes its artificial contrivances to become a  
touching psychological drama about despair and

9/10

Fresh

Apply

[Cancel](#)

[Delete](#)

Writing a review, a user can write its content, leave a summarizing score and determine if it's a *Fresh* (positive) or *Rotten* (negative) review. Here the user can also choose to delete its review



Top Critic reviews are highlighted by a little label with a star and a link that brings to its user page

## Michael Phillips

First Name: Michael  
Last Name: Phillips  
Registration Date: 2000-01-01  
Top Critic \*  
Number of followers: 28  
[Follow](#) [Unfollow](#)

### Most recent Reviews:

[Rebuilding Paradise \(2020-07-31\)](#)

"It's frequently gripping and finally very moving."

Review Score: "3.5/4"

[Sputnik \(2020-08-12\)](#)

"The clever and nicely gory "Sputnik" comes from Russia with love, slime, and an impressive lesson in efficient, low-cost pulp filmmaking."

Review Score: "3/4"

[Bill & Ted Face the Music \(2020-08-28\)](#)

"Cheesy, yes, hit-and-miss, maybe, but the bits that work really do work."

Review Score: "3/4"

### All reviews:

[The Boss](#) [Were the World Mine](#) [Oz the Great and Powerful](#) [Oliver Twist](#) [Monte Carlo](#) [Atonement](#)  
[Chimes at Midnight](#) [Doubt](#) [In Secret](#) [Possession](#) [Paris](#) [The American](#)

[The Tempest](#) [Coriolanus](#) [Alexandra](#) [Les Misérables](#) [Prometheus](#) [The Walk](#)

Viewing the user page of a Top Critic, a user can choose to follow or, if its already a follower, unfollow that critic

The screenshot shows the Rotten Tomatoes homepage. At the top right, there is a "Hall of Fame" dropdown menu. A mouse cursor is hovering over the "Genres" option, which is highlighted with a red dot. Other options in the dropdown are "Production Houses" and "Years". The main content area displays a movie card for "Avengers: Infinity War". The card includes the movie title, release year (2018), runtime (149 min), studio (Walt Disney Pictures), and genres (Action, Adventure, Sci-Fi). Below the movie card is a navigation bar for users who are not logged in.

The navbar for a user that hasn't logged-in has some basic features

The screenshot shows the Rotten Tomatoes homepage for a user who is not logged in. The navigation bar includes a search bar and a "Sort By" dropdown set to "Top Critic rating". To the right of the search bar is a "Minimum number of movies per genre" input field set to "5". A "Search" button is located next to the input field. Below the search bar is a table listing the top 10 genres by top critic rating. The table has columns for rank (#), genre, top critic rating, and movie count.

#	Genre	Top Critic Rating	Movie count
1	Documentary	75,09%	1430
2	News	72,50%	22
3	Biography	63,02%	968
4	History	62,59%	532
5	Music	60,23%	551
6	Film-Noir	59,42%	121
7	Sport	55,80%	328
8	War	55,78%	364
9	Animation	54,20%	380
10	Drama	53,32%	7565

This page shows the best genres across all movies, sorted by top critic rating or user rating. It is also possible to specify the minimum number of movies that a genre must have to appear in the list

The screenshot shows the Rotten Tomatoes homepage for a user who is not logged in. The navigation bar includes a search bar and a "Sort By" dropdown set to "Top Critic rating". To the right of the search bar is a "Minimum number of produced movies" input field set to "5". A "Search" button is located next to the input field. Below the search bar is a table listing the top 10 production houses based on top critic rating. The table has columns for rank (#), name, top critic rating, and movie count.

#	Name	Top Critic Rating	Movie count
1	HBO Documentary Films	95,50%	10
2	Rialto Pictures	94,07%	15
3	The Cinema Guild	92,78%	9
4	HBO	92,40%	5
5	Cinema Guild	91,00%	20
6	Disney/Pixar	89,88%	8
7	Docurama	89,00%	6
8	Gravitas	88,83%	6
9	Janus Films	88,46%	13
10	Area 23a	87,67%	6

This page shows the best production houses based on the reviews left by users on the platform. They can be sorted by top critic rating or user rating and the user can specify the minimum number of produced movies to appear on the list

**rotten Movies**

Sort By  Minimum number of movies released

#	Year	Top Critic Rating	Movie count
1	1912	76,14%	7
2	1950	68,63%	38
3	1930	66,31%	26
4	1951	65,63%	46
5	1924	62,50%	22
6	1933	62,50%	34
7	1918	62,35%	26
8	1953	61,55%	42
9	1952	59,50%	40
10	1955	58,75%	59

This page shows the best years in terms of produced movies with positive reviews left by users on the platform. They can be sorted by top critic rating or user rating and the user can specify the minimum number of released movies in that year to appear on the list

**Explore**  
Join our community, post your thoughts and follow like-minded top critics

**Links**  
[Index](#)  
[Logout](#)

**Personal Area** 2

- [Your profile](#) 1
- [Suggestions](#)
- [Your preferred genres](#)
- [Your personal feed](#)

**rotten Movies**

## Avengers: Infinity War

2018, 149 min, Walt Disney Pictures

Action Adventure Sci-Fi



The navbar of an authenticated user shows some more personalized pages

**JDoe99**

First Name: John

Last Name: Doe

Password

password

first name

John

last name

Doe

Birthday

19/07/1999

**Update**

Registration Date: 2023-01-10

Normal User

Most recent Reviews:

[Avengers: Infinity War \(2023-01-10\)](#)

"Overcomes its artificial contrivances to become a touching psychological drama about despair and loneliness."

Review Score: "9/10"

[Romeo and Juliet \(2023-01-10\)](#)

"Romeo and Juliet (1968) is Florentine director Franco Zeffirelli's beautiful modern interpretation of Shakespeare's enduring, classic yet tragic love story of"

Review Score: "2/5"

[The Three Musketeers \(2023-01-10\)](#)

"It might look good but it doesn't taste that good after a few bites."

Review Score: "C-"

All reviews:

[Avengers: Infinity War](#)

[Romeo and Juliet](#)

[The Three Musketeers](#)

In the personal page, a user can change its credentials, see its three most recent reviews and

the list of all its reviewed movies.

Leaving the password field blank won't change it.

#### rotten Movies

Here you can find your top critic suggestions, JDoe99

#	Username	Rate	
1	<a href="#">Edward Douglas</a>	80	<a href="#">Follow</a>
2	<a href="#">Robert Denerstein</a>	80	<a href="#">Follow</a>
3	<a href="#">Tim Appelo</a>	80	<a href="#">Follow</a>
4	<a href="#">Peter Rainer</a>	75	<a href="#">Follow</a>
5	<a href="#">Roxana Hadadi</a>	75	<a href="#">Follow</a>
6	<a href="#">Richard Roeper</a>	75	<a href="#">Follow</a>
7	<a href="#">Joe Morgenstern</a>	75	<a href="#">Follow</a>
8	<a href="#">Matt Zoller Seitz</a>	75	<a href="#">Follow</a>
9	<a href="#">Owen Gleiberman</a>	75	<a href="#">Follow</a>
10	<a href="#">Sean P. Means</a>	75	<a href="#">Follow</a>

The suggestion page shows a list of possible top critics that the user could follow. These top critics are chosen based on common reviewed movies (both positively or both negatively) between the two. The Rate index shows the estimated probability that the user might align with the suggested top critic views.

#### rotten Movies



#	Genre	Review count
1	Sci-Fi	2
2	Adventure	2
3	Action	1
4	Drama	1

The preferred genres page shows an aggregated overview of genres reviewed by the user. This highlights possible preferences towards certain genres more than others.

#### Here is your review feed, JDoe99

<a href="#">Bill &amp; Ted Face the Music</a>	<a href="#">Michael Phillips</a>	Date: "2020-08-28"
"Cheesy, yes, hi..."	<a href="#">Read full review</a>	
<a href="#">Sputnik</a>	<a href="#">Michael Phillips</a>	Date: "2020-08-12"
"The clever and ..."	<a href="#">Read full review</a>	
<a href="#">Rebuilding Paradise</a>	<a href="#">Michael Phillips</a>	Date: "2020-07-31"
"It's frequently..."	<a href="#">Read full review</a>	
<a href="#">I Used to Go Here</a>	<a href="#">Michael Phillips</a>	Date: "2020-07-28"
"The lightly car..."	<a href="#">Read full review</a>	
<a href="#">The Rental</a>	<a href="#">Michael Phillips</a>	Date: "2020-07-23"
"It starts out g..."	<a href="#">Read full review</a>	
<a href="#">Yes, God, Yes</a>	<a href="#">Michael Phillips</a>	Date: "2020-07-22"
"One of this sum..."	<a href="#">Read full review</a>	
<a href="#">Married to the Mob</a>	<a href="#">Roger Ebert</a>	Date: "2020-07-18"
"The results are..."	<a href="#">Read full review</a>	
<a href="#">Bloody Nose, Empty Pockets</a>	<a href="#">Michael Phillips</a>	Date: "2020-07-10"
"This movie crea..."	<a href="#">Read full review</a>	

The feed shows a list of recent reviews written by followed top critics. The review contains an excerpt of the full content, which can be seen by clicking on the button *Read full review*.



Please login

Username	admin
Password	*****

**Login**

[Don't have an account? Register](#)

A special kind of user can log-in to perform administration task on the application

**Explore**  
Join our community, post your thoughts and follow like-minded top critics

**Links**  
[Index](#)  
[Admin panel](#) ?  
[Logout](#) 1

[Hall of Fame](#) ▼

**rotten Movies** ≡

The admin navbar allows to reach the administrator panel

Username	Registration Date	Type	Birthday
Bob Bloom	2014-06-07	Normal User	1998-08-10
Bob Thomas	2001-01-13	Top Critic ★	----
Bob Mondello	1990-06-11	Top Critic ★	----
Bob Strauss	1997-04-22	Top Critic ★	----
Bob Grimm	2014-08-23	Normal User	1989-07-06

In the administrator panel the admin can search for a specific user. The side panel links to different analytic pages about the status of the application

**Bob Bloom**

First Name: Bob  
Last Name: Bloom  
Registration Date: 2014-06-07  
Normal User

**BAN** **UNBAN**

Most recent Reviews:

**Love & Debt (2020-08-19)**  
"Love & Debt" ends on a note of hopeful uncertainty. It is a movie that offers lessons that hundreds of families who are struggling today can learn from and embrace.

Review Score: "3/4"

**Tesla (2020-08-19)**  
"The crucial flaw with "Tesla" is that it, ironically, lacks spark. Almerryda's depiction of Tesla may be accurate, but cinematically it filters the spotlight that this nearly

Review Score: "2/4"

After selecting a user, the admin can ban or unban it

**rotten Movies**

Insert start year, the offset and the index

5  

Year	Count
1970-1975	815
1975-1980	803
1980-1985	834
1985-1990	817
1990-1995	808
1995-2000	806
2000-2005	757
2005-2010	327

This analytic shows the number of registered users for each year range, passed as input

**rotten Movies**

#	Username	Type	Number of Reviews
1	<a href="#">Emanuel Levy</a>	TopCritic	5850
2	<a href="#">Roger Ebert</a>	TopCritic	4850
3	<a href="#">Dennis Schwartz</a>	User	4813
4	<a href="#">Brian Omdorf</a>	User	4546
5	<a href="#">Frank Swietek</a>	User	4528
6	<a href="#">Jeffrey M. Anderson</a>	User	4153
7	<a href="#">Roger Moore</a>	TopCritic	4143
8	<a href="#">James Berardinelli</a>	TopCritic	3995
9	<a href="#">David Nusair</a>	User	3908
10	<a href="#">Frederic and Mary Ann Brussat</a>	User	3677

This page shows a ranking of users based on their number of reviews. This can be useful to verify a possible automated user, or promote a certified active user as a top critic

**rotten Movies**

#	Username	Type	Number of Reviews
1	<a href="#">Elissa Suh</a>		44
2	<a href="#">Gene Stout</a>		41
3	<a href="#">Harry Carr</a>		41
4	<a href="#">Lynne Heffley</a>		41
5	<a href="#">Gemma Files</a>		41
6	<a href="#">Chris Harvey</a>		40
7	<a href="#">Suzanne S. Brown</a>		40
8	<a href="#">Julia Irion Martins</a>		40
9	<a href="#">Manohla Dargis</a>		40
10	<a href="#">Drew Toal</a>		40

With this page, the administrator can see a ranking of most active top critics, based on the number of reviews



Please sign in

First Name	Alice
Last Name	Johnson
Username	AJohnson87
Password	*****
Birthday	12/03/1987

Register



The top critic creation page isn't accessible to the public, only to the registered administrator. It is very similar to the user registration page, but it doesn't redirect to the explore movie page after a successful sign-up

The screenshot shows the Rotten Tomatoes search interface. At the top, there's a search bar with the placeholder "Avatar". Below it are two input fields for "Start Year Range" (1980) and "End Year Range" (2020). There are also dropdown menus for "Personnel" (listing Leonardo DiCaprio and Johnny Depp) and "Genres" (Adventure, Sci-Fi). To the right of these, there's a "Sort By" dropdown set to "top critic rating" and a "Descending order" checkbox. At the bottom of the search bar area, there are "Previous" and "Next" buttons. To the right of the search bar, there's a green "Add Movie" button with a hand cursor icon over it. Below the search bar, there are three movie cards. The first two cards are partially visible, each showing an orange movie icon. The third card is fully visible and features a poster for the movie "ROMEO + JULIET" with the lead actors, Romeo and Juliet, looking intensely at each other.

Adding a new movie is done through the explore movie page. A new button labeled *Add Movie* appears near the *Search* one and will create a new movie, temporarily named as the string currently present in the search bar

title  
Avatar 2

year  
2022

runtimeMinutes  
192

productionCompany  
AMC

genres (use a comma ',' separated list of genres)  
Action, Adventure

posterUrl  
posterUrl

Personnel

primaryName	James Cameron
category	Director
job or interpreted characters	job_characters
primaryName	Sam Worthington
category	Actor
job or interpreted characters	Jake Sully

After adding a movie, or when selecting an existing one, the administrator can change its descriptive information

primaryName  
Balwant Bhatt

category  
director

job or interpreted characters  
job\_characters

primaryName  
Master Bhagwan

category  
actor

job or interpreted characters  
job\_characters

[Update](#) [Delete](#) [Check Review Bombing](#)

When viewing a movie, the administrator can check for possible review bombing for this movie, by clicking on the specific button

rotten Movies ≡

Insert the number of month  
36 Search

Movie	Storic Count	Storic Rate	Target Count	Target Rate
Joker	528	12.0	44	6.0

**Admin manual**

**Movie:** title of the selected movie

**Storic Count:** count of total number of review before the date: today - number of month passed in the input

**Storic Rate:** rate of the positive review before the date: today - number of month passed in the input

**Target Count:** count of the total number of review between today and the date: today - number of month passed in the input

**Target Rate:** rate of the positive review between today and the date: today - number of month passed in the input

The review bombing page shows some statistics about recent reviews made on a specific movie. An info box explains the meaning of each value

# A Python Code

## A .1 Creation of one single dataset from the tsv imdb file

```
1 import pandas as pd
2 from google.colab import drive
3 drive.mount('/content/drive')
4 numberofrows=None #100000
5 title_basics = pd.read_csv("/content/drive/MyDrive/Dataset/Original/
6     title_basics.tsv",sep='\t',nrows=numberofrows,header=0)
7 title_principals = pd.read_csv("/content/drive/MyDrive/Dataset/Original/
8     title_principals.tsv",nrows=numberofrows,sep='\t',header=0)
9
10 keep_col = ["tconst","titleType","primaryTitle","originalTitle","startYear",
11     "runtimeMinutes","genres"]
12 title_basics = title_basics[keep_col]
13 title_basics = title_basics[title_basics["titleType"].str.contains("movie")
14     == True]
15
16 print(title_basics.head(3))
17
18 merged1=pd.merge(title_basics,title_principals,how='inner',on='tconst')
19 del title_basics,title_principals
20 print(merged1)
21
22 name_basics=pd.read_csv("/content/drive/MyDrive/Dataset/Original/name_basics.
23     tsv",sep='\t',nrows=numberofrows,header=0)
24
25 merged2=pd.merge(merged1,name_basics,how='inner',on='nconst')
26 del merged1
27 merged2=merged2.drop(columns=["ordering","nconst","birthYear","deathYear",
28     "knownForTitles","primaryProfession"])
29 del name_basics
30 print(merged2)
31
32 category=merged2.groupby('tconst')[['category']].apply(list).reset_index(name=
33     'category')
34 job=merged2.groupby('tconst')[['job']].apply(list).reset_index(name='job')
35 characters=merged2.groupby('tconst')[['characters']].apply(list).reset_index(
36     name='characters')
37 primaryName=merged2.groupby('tconst')[['primaryName']].apply(list).reset_index(
38     name='primaryName')
39 result=merged2.drop_duplicates(subset=['tconst'])
40 result=result.drop(['category'], axis=1).drop(['job'], axis=1).drop(['
41     characters'], axis=1).drop(['primaryName'], axis=1)
42 result=result.merge(category, on='tconst').merge(job, on='tconst').merge(
43     characters, on='tconst').merge(primaryName, on='tconst')
44 print(result)
45
46 result.to_csv("/content/drive/MyDrive/Dataset/resultSetFinale.csv",index=
47     False)
```

Listing 8.1: Test

## A .2 Creation of one single dataset from the csv kaggle file

```

2 import pandas as pd
3 from google.colab import drive
4 drive.mount('/content/drive')
5 numberofrows=None #100000
6 movies = pd.read_csv("/content/drive/MyDrive/Dataset/Original/rotten_movies.csv",nrows=numberofrows, header=0)
7 reviews = pd.read_csv("/content/drive/MyDrive/Dataset/Original/rotten_reviews.csv",nrows=numberofrows, header=0)
8 to_keep = ["rotten_tomatoes_link", "movie_title", "production_company", "critics_consensus",
9             "tomatometer_status", "tomatometer_rating", "tomatometer_count",
10            "audience_status", "audience_rating", "audience_count",
11            "tomatometer_top_critics_count", "tomatometer_fresh_critics_count"
12            ,
13            "tomatometer_rotten_critics_count"]
14 movies = movies[to_keep]
15 to_drop = ["publisher_name"]
16 reviews=reviews.drop(columns=to_drop)
17
18 merged=pd.merge(movies, reviews, how='inner', on="rotten_tomatoes_link")
19 print(merged)
20
21 categories = {}
22 arr = ["critic_name", "top_critic", "review_type", "review_score", "review_date", "review_content"]
23 for x in arr:
24     categories[x]=merged.groupby('rotten_tomatoes_link')[x].apply(list).reset_index(name=x)
25
26 result=merged.drop_duplicates(subset=['rotten_tomatoes_link'])
27 for x in arr:
28     result=result.drop([x], axis=1)
29 for x in arr:
30     result=result.merge(categories[x], on='rotten_tomatoes_link')
31
32 print(result)
33
34 result.to_csv("/content/drive/MyDrive/Dataset/resultSetRotten.csv",index=False)

```

Listing 8.2: Test

### A .3 Merging of the file generated in the previous script

```

1
2 import pandas as pd
3 from google.colab import drive
4 drive.mount('/content/drive')
5 numberofrows=None
6 imdb = pd.read_csv("/content/drive/MyDrive/Dataset/resultSetFinale.csv",nrows=numberofrows, header=0)
7 rotten = pd.read_csv("/content/drive/MyDrive/Dataset/resultSetRotten.csv",nrows=numberofrows, header=0)
8
9 merged = {}
10 choose = ['primaryTitle', 'originalTitle']
11 rowHeadDataset = 20
12 for x in choose:

```

```

13 merged[x]=pd.merge(imdb, rotten ,how='inner' , left_on=x , right_on='
    movie_title')
14 print(x)
15 merged[x]=merged[x].drop_duplicates(subset=[x])
16 merged[x]=merged[x].drop(columns=['titleType','tomatometer_count','
    tomatometer_top_critics_count'])
17 merged[x]=merged[x].rename(columns={'startYear':'year'})
18 merged[x]=merged[x].drop(columns=['rotten_tomatoes_link','movie_title']+[
    for j in choose if j!=x])
19 print(len(pd.unique(merged[x][x])))
20 print(list(merged[x]))
21 print("====")
22 merged[x].to_csv(f"/content/drive/MyDrive/Dataset/ImdbJoinRotten{x}.csv",
    index=False)
23 merged[x]=merged[x].head(rowHeadDataset)
24 merged[x].to_csv(f"/content/drive/MyDrive/Dataset/headDataset{x}.csv",index
    =False)

```

Listing 8.3: Test

#### A .4 Collapsing different rows in a single one generating an array for personnel field

```

1
2 import pandas as pd
3 from ast import literal_eval
4 from google.colab import drive
5 drive.mount('/content/drive')
6
7 numberofrows=None
8 df = pd.read_csv("/content/drive/MyDrive/Dataset/ImdbJoinRottenprimaryTitle.csv",
    nrows=numberofrows, header=0)
9
10 #print([x.split(',') for x in df['genres']])
11 print(df)
12
13 col = ["primaryName","category","job","characters"]
14 coll1 = ["critic_name","top_critic","review_type","review_score","review_date",
    "review_content"]
15
16 df['personnel'] = ""
17 df['review'] = ""
18
19 for row in range(df[col[0]].size):
20     it = df['genres'][row]
21     df['genres'][row] = ["" + x + "" for x in it.split(',') if it != '\n'
        else []]
22     tmp = []
23     for c in col:
24         tmp.append({c:eval(df[c][row])})
25     res = []
26     for c in range(len(tmp[0][col[0]])):
27         res.append({})
28     for i, j in zip(col, tmp):
29         for idx, x in enumerate(j[i]):
30             #print(i, idx, x)
31             if x != '\n':
32                 if i == 'characters':
33                     x = eval(x)

```

```

34         res[idx][i + ""] = "" + str(x).replace("'", "#single-quote##"
35         "").replace('"', "#double-quote##") + ""
36 df['personnel'][row] = list(res)
37 #
38 tmp = []
39 for c in col1:
40     to_eval = df[c][row].replace('nan', 'None')
41     arr = eval(to_eval)
42     if c == "review_date":
43         for i, elem in enumerate(arr):
44             arr[i] = elem + "T00:00:00.000+00:00"
45     tmp.append({c: arr})
46     #print(tmp)
47 res = []
48 for c in range(len(tmp[0][col1[0]])):
49     res.append({})
50 for i, j in zip(col1, tmp):
51     for idx, x in enumerate(j[i]):
52         #print(i, idx, x)
53         if x != '\\N':
54             res[idx][i + ""] = "" + str(x).replace("True", "true").
55             replace("False", "false").replace("'", "#single-quote##").replace('"',
56             "#double-quote##") + ""
57 df['review'][row] = list(res)
58 #df['review'][row] = eval(str(res))
59 #print(res)
60 #print()
61
62 df=df.drop(columns=col)
63 df=df.drop(columns=col1)
64 df=df.drop(columns=['tconst'])
65
66 print(df["review"][0])
67
68 it = df['personnel'][0][4]['review_content']
69 print(type(it))
70 print(it)
71
72 df.to_csv("/content/drive/MyDrive/Dataset/
73             movieCollectionEmbeddedReviewPersonnel.csv", index=False)
74 df = df.head(20)
75 df.to_csv("/content/drive/MyDrive/Dataset/
76             headmovieCollectionEmbeddedReviewPersonnel.csv", index=False)

```

Listing 8.4: Test

## A .5 Generates a hashed password for all the users

```

1 import hashlib
2 #from pprint import pprint as print
3 from pymongo import MongoClient
4
5 def get_database():
6     CONNECTION_STRING = "mongodb://localhost:27017"
7     client = MongoClient(CONNECTION_STRING)
8     return client['rottenMovies']
9
10 if __name__ == "__main__":

```

```

11     dbname = get_database()
12     collection = dbname[ 'user' ]
13     total = collection . count_documents( {} )
14     for i , user in enumerate(collection . find ()):
15         all_reviews = user[ 'last_3_reviews' ]
16         sorted_list = sorted(all_reviews , key=lambda t: t[ 'review_date' ])
17         [-3:]
18
19         hashed = hashlib . md5(user[ "username" ]. encode () ) . hexdigest ()
20
21         newvalues = { "$set": { 'password': hashed , 'last_3_reviews': sorted_list } }
22         filter = { 'username': user[ 'username' ] }
23         collection . update_one(filter , newvalues)
24         print(f" {i}/{total} % \r" , end= ' ')
25     print()

```

Listing 8.5: Test

## A .6 Generates the graph database

```

1 from pymongo import MongoClient
2 from neo4j import GraphDatabase
3 from random import randint , shuffle
4
5 def get_database():
6     CONNECTION_STRING = "mongodb://localhost:27017"
7     client = MongoClient(CONNECTION_STRING)
8     return client[ 'rottenMovies' ]
9
10 class Neo4jGraph:
11
12     def __init__(self , uri , user , password):
13         self . driver = GraphDatabase . driver(uri , auth=(user , password) ,
14         database="rottenmoviesgraphdb")
15
16     def close(self):
17         self . driver . close()
18
19     def addUser(self , uid , name , isTop):
20         with self . driver . session() as session:
21             if isTop:
22                 result = session . execute_write(self . _addTopCritic , uid , name)
23             else:
24                 result = session . execute_write(self . _addUser , uid , name)
25
26     def addMovie(self , mid , title):
27         with self . driver . session() as session:
28             result = session . execute_write(self . _addMovie , mid , title)
29
30     def addReview(self , name , mid , freshness , content , date):
31         with self . driver . session() as session:
32             result = session . execute_write(self . _addReview , name , mid ,
33             freshness , content , date)
34
35     def addFollow(self , uid , cid):
36         with self . driver . session() as session:
37             result = session . execute_write(self . _addFollow , uid , cid)

```

```

37     @staticmethod
38     def _addUser(tx, uid, name):
39         query = "CREATE (n:User{id:'" + str(uid) + "'}, name:'" + name +
40         replace("'", "\\'') + "')"
41         #print(query)
42         result = tx.run(query)
43
44     @staticmethod
45     def _addTopCritic(tx, cid, name):
46         query = "CREATE(m: TopCritic{id:'" + str(cid) + "'}, name:'" + name +
47         replace("'", "\\'') + "')"
48         #print(query)
49         result = tx.run(query)
50
51     @staticmethod
52     def _addMovie(tx, mid, title):
53         query = "CREATE(o:Movie{id:'" + str(mid) + "'}, title:'" + title +
54         replace("'", "\\'') + "')"
55         #print(query)
56         result = tx.run(query)
57
58     @staticmethod
59     def _addReview(tx, name, mid, freshness, content, date): # date in format
60         YYYY-mm-dd, freshness in [TRUE, FALSE]
61         query = "MATCH(n{name:'" + str(name).replace("'", "\\'') + "'}), (m:
62         :Movie{id:'" + str(mid) + "'}) CREATE (n)-[r:REVIEWED{freshness:" +
63         freshness + ", date:date('" + date + "')}, content:'" + content.replace(
64         "'", "\\'') + "'}->(m)"
65         #print(query)
66         result = tx.run(query)
67
68     @staticmethod
69     def _addFollow(tx, uid, cid):
70         query = "MATCH(n:User{id:'" + str(uid) + "'}, (m: TopCritic{id:'" +
71         + str(cid) + "'}) CREATE (n)-[r:FOLLOWS]->(m)"
72         #print(query)
73         result = tx.run(query)
74
75 if __name__ == "__main__":
76     # dbs initialization
77     dbname = get_database()
78     graphDB = Neo4jGraph("bolt://localhost:7687", "neo4j", "password")
79
80     # user creation
81     collection = dbname['user']
82     total = collection.count_documents({})
83     print(f"user {total}")
84     for i, user in enumerate(list(collection.find({}), {"_id":1, "username":1,
85     "date_of_birth":1})):
86         graphDB.addUser(user['_id'], user['username'], 'date_of_birth' not in
87         user)
88         if not i%100:
89             print(f"{(i+1)/total}%\r", end=' ')
90
91     # movie creation and review linking
92     collection = dbname['movie']
93     total = collection.count_documents({})
94     print(f"\nmovie {total}")
95     for i, movie in enumerate(list(collection.find({}), {"_id":1, "primaryTitle":1,
96     "review":1})):
97         graphDB.addMovie(movie['_id'], movie['primaryTitle'])

```

```

87     movie[ 'review' ] = list({v[ 'critic_name' ]:v for v in movie[ 'review' ]}).
88     values() # make unique reviews per critic
89     for rev in movie[ 'review' ]:
90         graphDB.addReview(rev[ 'critic_name' ], movie[ 'id' ], {"Fresh": "
91         TRUE", "Rotten": "FALSE"}[rev[ 'review_type' ]], str(rev[ 'review_content' ])
92         [:15], str(rev[ 'review_date' ])[:10])
93         print(f"\{(i+1)/total:%}\r", end=' ')
94
95     # follow linking
96     collection = dbname[ 'user' ]
97     uids = [x[ '_id' ] for x in list(collection.find({"date_of_birth":{ "$exists
98         ":True}}, { "_id":1}))]
99     cids = [x[ '_id' ] for x in list(collection.find({"date_of_birth":{ "$exists
100        ":False}}, { "_id":1}))]
101    total = len(uids)
102    print(f"\nfollow {total = }")
103    for i, user in enumerate(uids):
104        shuffle(cids)
105        for j in range(randint(0, 20)):
106            graphDB.addFollow(user, cids[j])
107            print(f"\{i/total:%}\r", end=' ')
108
109    graphDB.close()

```

Listing 8.6: Test

## B Mongosh scripts

### B .1 Perform the escape on the string fields

```
1
2 db.movie.find().forEach(
3     x => {
4         print(x.primaryTitle);
5         x.review = JSON.parse(
6             x.review.replaceAll('\\', '\"')
7                 .replaceAll('\\', '\"')
8                 .replaceAll('false', 'false')
9                 .replaceAll('true', 'true')
10                .replaceAll('None', 'null')
11                .replaceAll(/\x\d{2}/g, '')
12                .replaceAll("##single-quote##", "\'")
13                .replaceAll("##double-quote##", '\"')
14                .replaceAll("\x", "x")
15        );
16        x.personnel = JSON.parse(
17            x.personnel.replaceAll('\\', '\"')
18                .replaceAll('\\', '\"')
19                .replaceAll('None', 'null')
20                .replaceAll("##single-quote##", "\'")
21                .replaceAll("##double-quote##", '\"')
22                .replaceAll('[\\', '[')
23                .replaceAll('[\\", [\"')
24                .replaceAll('\\]', ']')
25                .replaceAll('\\\"]', '\"')
26                .replaceAll(/(\[[^:]*)\\\", \\\"([:^]*\])/g, '$1', '$2')
27                .replaceAll(/(\[[^:]*)\\\", \\\"([:^]*\])/g, '$1', '$2')
28                .replaceAll(/(\[[^:]*)\\\", \\'([:^]*\])/g, '$1', '$2')
29        );
30        x.genres = JSON.parse(
31            x.genres.replaceAll('\\', '\"')
32                .replaceAll('\\', '\"')
33                .replaceAll('None', 'null')
34                .replaceAll("##single-quote##", "\'")
35                .replaceAll("##double-quote##", '\"')
36        );
37        db.movie.updateOne(
38            {"_id": x._id},
39            {$set:
40                {
41                    "review": x.review,
42                    "personnel": x.personnel,
43                    "genres": x.genres,
44                    "runtimeMinutes": parseInt(x.runtimeMinutes),
45                    "year": parseInt(x.year),
46                    "tomatometer_rating": parseFloat(x.tomatometer_rating),
47                    "audience_rating": parseFloat(x.audience_rating),
48                    "audience_count": parseFloat(x.audience_count),
49                    "tomatometer_fresh_critics_count": parseInt(x.
50                        tomatometer_fresh_critics_count),
51                    "tomatometer_rotten_critics_count": parseInt(x.
52                        tomatometer_rotten_critics_count)
53                }
54            }
55        );
56    }
57);
```

```
54     }
55 );
```

Listing 8.7: Test

## B .2 Normalize the date field in the DB

```
1 total = db.movie.find().count();
2 i = 0;
3 db.movie.find().forEach(
4     x => {
5         print(x.primaryTitle);
6         x.review.forEach(rev =>{
7             if(typeof (rev.review_date) === "string" ){
8                 db.movie.updateOne(
9                     {primaryTitle: x.primaryTitle },
10                    { $set: { "review.$[elem].review_date" : new Date(rev.
11 review_date) } },
12                     { arrayFilters: [ { "elem.critic_name": rev.critic_name } ]
13                 )
14             })
15             print(100*i++/total);
16 }) ;
```

Listing 8.8: Test

## B .3 Create a new collection for the user based on the data present in the movie collection

```
1 total = db.runCommand({ distinct: "movie", key: "review.critic_name", query:
2     {"review.critic_name":{$ne: null}}}).values.length
3 i = 0;
4 db.runCommand(
5 { distinct: "movie", key: "review.critic_name", query: {"review.critic_name"
6     :{$ne: null}}}).values.forEach(
7     (x) => {
8         review_arr = []
9         movie_arr = []
10        is_top = false
11        db.movie.aggregate(
12            [
13                { $project:
14                    {
15                        index: { $indexOfArray: [ "$review.critic_name", x ] },
16                        primaryTitle: 1
17                    }
18                },
19                {$match:{ index:{ $gt:-1 }}}
20            ]
21        ).forEach(
22            y => {
23                tmp = db.movie.aggregate([
24                    {
25                        $project:
26                            {
27                                top_critic: {
```

```

25                     $arrayElemAt: ["$review.top_critic", y.index]
26                 },
27                 primaryTitle: y.primaryTitle,
28                 review_type: {
29                     $arrayElemAt: ["$review.review_type", y.index
30                 ]
31             },
32             review_score: {
33                 $arrayElemAt: ["$review.review_score", y.
34             index]
35         },
36         review_date: {
37             $arrayElemAt: ["$review.review_date", y.index
38         ]
39     },
40 }
41 },
42 {
43     $match: {_id: {$eq: y._id}}
44 }
45 ]) . toArray () [0];
46 is_top |= tmp.top_critic;
47 review_arr.push(tmp)
48 //movie_arr.push(tmp._id)
49 movie_arr.push({ "movie_id": tmp._id, "primaryTitle": y.
50 primaryTitle, "review_index": y.index})
51 }
52
53 name_parts = x.split (/ \s /)
54 first_name = name_parts.splice (0, 1) [0]
55 last_name = name_parts.join (' ')
56
57 print(100*i++/total, x, is_top)
58 //print(first_name, ':', last_name)
59 //print(review_arr)
60 //print(movie_arr)
61 db.user.insertOne(
62     {
63         "username": x,
64         "password": "",
65         "first_name": first_name,
66         "last_name": last_name,
67         "registration_date": new Date("2000-01-01"),
68         "last_3_reviews": review_arr,
69         "reviews" : movie_arr
70     }
71 );
72 if (!is_top){
73     db.user.updateOne(
74         { "username": x },
75         { $set:
76             { "date_of_birth": new Date("1970-07-20") }
77         }
78     }
79     print("====")
80 }

```

Listing 8.9: Test

## C MongoDB indexes:Movie collection

### C .1 primaryTitle

Before the index

```

1  {
2      explainVersion: '1',
3      queryPlanner: {
4          namespace: 'rottenMovies.movie',
5          indexFilterSet: false,
6          parsedQuery: { primaryTitle: { '$eq': 'Evidence' } },
7          queryHash: '9839850C',
8          planCacheKey: '9839850C',
9          maxIndexedOrSolutionsReached: false,
10         maxIndexedAndSolutionsReached: false,
11         maxScansToExplodeReached: false,
12         winningPlan: {
13             stage: 'COLLSCAN',
14             filter: { primaryTitle: { '$eq': 'Evidence' } },
15             direction: 'forward'
16         },
17         rejectedPlans: []
18     },
19     executionStats: {
20         executionSuccess: true,
21         nReturned: 1,
22         executionTimeMillis: 275,
23         totalKeysExamined: 0,
24         totalDocsExamined: 14104,
25         executionStages: {
26             stage: 'COLLSCAN',
27             filter: { primaryTitle: { '$eq': 'Evidence' } },
28             nReturned: 1,
29             executionTimeMillisEstimate: 245,
30             works: 14106,
31             advanced: 1,
32             needTime: 14104,
33             needYield: 0,
34             saveState: 18,
35             restoreState: 18,
36             isEOF: 1,
37             direction: 'forward',
38             docsExamined: 14104
39         }
40     },
41     command: {
42         find: 'movie',
43         filter: { primaryTitle: 'Evidence' },
44         '$db': 'rottenMovies'
45     },
46     serverInfo: {
47         host: 'Profile2022LARGE10',
48         port: 27017,
49         version: '6.0.3',

```

```

50     gitVersion: 'f803681c3ae19817d31958965850193de067c516',
51 },
52 serverParameters: {
53     internalQueryFacetBufferSizeBytes: 104857600,
54     internalQueryFacetMaxOutputDocSizeBytes: 104857600,
55     internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
56     internalDocumentSourceGroupMaxMemoryBytes: 104857600,
57     internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
58     internalQueryProhibitBlockingMergeOnMongoS: 0,
59     internalQueryMaxAddToSetBytes: 104857600,
60     internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
61 },
62 ok: 1,
63 '$clusterTime': {
64     clusterTime: Timestamp({ t: 1673280853, i: 1 }),
65     signature: {
66         hash: Binary(Buffer.from("0000000000000000000000000000000000000000000000000000000000000000", "hex")),
67         keyId: Long("0")
68     }
69 },
70 operationTime: Timestamp({ t: 1673280853, i: 1 })
71 }

```

Listing 8.10: Test

After the index

```

1 {
2     explainVersion: '1',
3     queryPlanner: {
4         namespace: 'rottenMovies.movie',
5         indexFilterSet: false,
6         parsedQuery: { primaryTitle: { '$eq': 'Evidence' } },
7         queryHash: '9839850C',
8         planCacheKey: 'B734708E',
9         maxIndexedOrSolutionsReached: false,
10        maxIndexedAndSolutionsReached: false,
11        maxScansToExplodeReached: false,
12        winningPlan: {
13            stage: 'FETCH',
14            inputStage: {
15                stage: 'IXSCAN',
16                keyPattern: { primaryTitle: 1 },
17                indexName: 'primaryTitle_1',
18                isMultiKey: false,
19                multiKeyPaths: { primaryTitle: [] },
20                isUnique: false,
21                isSparse: false,
22                isPartial: false,
23                indexVersion: 2,
24                direction: 'forward',
25                indexBounds: { primaryTitle: [ "[Evidence", "Evidence"] ] }
26            }
27        },
28        rejectedPlans: []
29    },
30    executionStats: {
31        executionSuccess: true,
32        nReturned: 1,
33        executionTimeMillis: 1,
34        totalKeysExamined: 1,

```

```

35     totalDocsExamined: 1,
36     executionStages: {
37       stage: 'FETCH',
38       nReturned: 1,
39       executionTimeMillisEstimate: 0,
40       works: 2,
41       advanced: 1,
42       needTime: 0,
43       needYield: 0,
44       saveState: 0,
45       restoreState: 0,
46       isEOF: 1,
47       docsExamined: 1,
48       alreadyHasObj: 0,
49       inputStage: {
50         stage: 'IXSCAN',
51         nReturned: 1,
52         executionTimeMillisEstimate: 0,
53         works: 2,
54         advanced: 1,
55         needTime: 0,
56         needYield: 0,
57         saveState: 0,
58         restoreState: 0,
59         isEOF: 1,
60         keyPattern: { primaryTitle: 1 },
61         indexName: 'primaryTitle_1',
62         isMultiKey: false,
63         multiKeyPaths: { primaryTitle: [] },
64         isUnique: false,
65         isSparse: false,
66         isPartial: false,
67         indexVersion: 2,
68         direction: 'forward',
69         indexBounds: { primaryTitle: [ '[ "Evidence", "Evidence"]' ] },
70         keysExamined: 1,
71         seeks: 1,
72         dupsTested: 0,
73         dupsDropped: 0
74       }
75     }
76   },
77   command: {
78     find: 'movie',
79     filter: { primaryTitle: 'Evidence' },
80     '$db': 'rottenMovies'
81   },
82   serverInfo: {
83     host: 'Profile2022LARGE10',
84     port: 27017,
85     version: '6.0.3',
86     gitVersion: 'f803681c3ae19817d31958965850193de067c516',
87   },
88   serverParameters: {
89     internalQueryFacetBufferSizeBytes: 104857600,
90     internalQueryFacetMaxOutputDocSizeBytes: 104857600,
91     internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
92     internalDocumentSourceGroupMaxMemoryBytes: 104857600,
93     internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
94     internalQueryProhibitBlockingMergeOnMongoS: 0,
95     internalQueryMaxAddToSetBytes: 104857600,

```

```

96     internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
97   },
98   ok: 1,
99   '$clusterTime': {
100     clusterTime: Timestamp({ t: 1673285103, i: 1 }),
101     signature: {
102       hash: Binary(Buffer.from("0000000000000000000000000000000000000000000000000000000000000000", "hex"), 0),
103       keyId: Long("0")
104     }
105   },
106   operationTime: Timestamp({ t: 1673285103, i: 1 })
107 }

```

Listing 8.11: Test

## C .2 year

Before the index

```

1  {
2    explainVersion: '1',
3    queryPlanner: {
4      namespace: 'rottenMovies.movie',
5      indexFilterSet: false,
6      parsedQuery: { year: { '$eq': 2012 } },
7      queryHash: '412E8B51',
8      planCacheKey: '412E8B51',
9      maxIndexedOrSolutionsReached: false,
10     maxIndexedAndSolutionsReached: false,
11     maxScansToExplodeReached: false,
12     winningPlan: {
13       stage: 'COLLSCAN',
14       filter: { year: { '$eq': 2012 } },
15       direction: 'forward'
16     },
17     rejectedPlans: []
18   },
19   executionStats: {
20     executionSuccess: true,
21     nReturned: 480,
22     executionTimeMillis: 13,
23     totalKeysExamined: 0,
24     totalDocsExamined: 14104,
25     executionStages: {
26       stage: 'COLLSCAN',
27       filter: { year: { '$eq': 2012 } },
28       nReturned: 480,
29       executionTimeMillisEstimate: 1,
30       works: 14106,
31       advanced: 480,
32       needTime: 13625,
33       needYield: 0,
34       saveState: 14,
35       restoreState: 14,
36       isEOF: 1,
37       direction: 'forward',
38       docsExamined: 14104
39     }
40   },

```

```

41   command: { find: 'movie', filter: { year: 2012 }, '$db': 'rottenMovies' },
42   serverInfo: {
43     host: 'Profile2022LARGE10',
44     port: 27017,
45     version: '6.0.3',
46     gitVersion: 'f803681c3ae19817d31958965850193de067c516',
47   },
48   serverParameters: {
49     internalQueryFacetBufferSizeBytes: 104857600,
50     internalQueryFacetMaxOutputDocSizeBytes: 104857600,
51     internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
52     internalDocumentSourceGroupMaxMemoryBytes: 104857600,
53     internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
54     internalQueryProhibitBlockingMergeOnMongoS: 0,
55     internalQueryMaxAddToSetBytes: 104857600,
56     internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
57   },
58   ok: 1,
59   '$clusterTime': {
60     clusterTime: Timestamp({ t: 1673280923, i: 1 }),
61     signature: {
62       hash: Binary(Buffer.from("0000000000000000000000000000000000000000000000000000000000000000", "hex"), 0),
63       keyId: Long("0")
64     }
65   },
66   operationTime: Timestamp({ t: 1673280923, i: 1 })
67 }

```

Listing 8.12: Test

After the index

```

1  {
2   explainVersion: '1',
3   queryPlanner: {
4     namespace: 'rottenMovies.movie',
5     indexFilterSet: false,
6     parsedQuery: { year: { '$eq': 2012 } },
7     queryHash: '412E8B51',
8     planCacheKey: '62915BA3',
9     maxIndexedOrSolutionsReached: false,
10    maxIndexedAndSolutionsReached: false,
11    maxScansToExplodeReached: false,
12    winningPlan: {
13      stage: 'FETCH',
14      inputStage: {
15        stage: 'IXSCAN',
16        keyPattern: { year: 1 },
17        indexName: 'year_1',
18        isMultiKey: false,
19        multiKeyPaths: { year: [] },
20        isUnique: false,
21        isSparse: false,
22        isPartial: false,
23        indexVersion: 2,
24        direction: 'forward',
25        indexBounds: { year: [ '[2012, 2012]' ] }
26      }
27    },
28    rejectedPlans: []
29  }

```

```

30     executionStats: {
31         executionSuccess: true ,
32         nReturned: 480 ,
33         executionTimeMillis: 2 ,
34         totalKeysExamined: 480 ,
35         totalDocsExamined: 480 ,
36         executionStages: {
37             stage: 'FETCH' ,
38             nReturned: 480 ,
39             executionTimeMillisEstimate: 0 ,
40             works: 481 ,
41             advanced: 480 ,
42             needTime: 0 ,
43             needYield: 0 ,
44             saveState: 0 ,
45             restoreState: 0 ,
46             isEOF: 1 ,
47             docsExamined: 480 ,
48             alreadyHasObj: 0 ,
49             inputStage: {
50                 stage: 'IXSCAN' ,
51                 nReturned: 480 ,
52                 executionTimeMillisEstimate: 0 ,
53                 works: 481 ,
54                 advanced: 480 ,
55                 needTime: 0 ,
56                 needYield: 0 ,
57                 saveState: 0 ,
58                 restoreState: 0 ,
59                 isEOF: 1 ,
60                 keyPattern: { year: 1 } ,
61                 indexName: 'year_1' ,
62                 isMultiKey: false ,
63                 multiKeyPaths: { year: [] } ,
64                 isUnique: false ,
65                 isSparse: false ,
66                 isPartial: false ,
67                 indexVersion: 2 ,
68                 direction: 'forward' ,
69                 indexBounds: { year: [ '[2012, 2012]' ] } ,
70                 keysExamined: 480 ,
71                 seeks: 1 ,
72                 dupsTested: 0 ,
73                 dupsDropped: 0
74             }
75         }
76     },
77     command: { find: 'movie' , filter: { year: 2012 } , '$db': 'rottenMovies' } ,
78     serverInfo: {
79         host: 'Profile2022LARGE10' ,
80         port: 27017 ,
81         version: '6.0.3' ,
82         gitVersion: 'f803681c3ae19817d31958965850193de067c516' ,
83     } ,
84     serverParameters: {
85         internalQueryFacetBufferSizeBytes: 104857600 ,
86         internalQueryFacetMaxOutputDocSizeBytes: 104857600 ,
87         internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600 ,
88         internalDocumentSourceGroupMaxMemoryBytes: 104857600 ,
89         internalQueryMaxBlockingSortMemoryUsageBytes: 104857600 ,
90         internalQueryProhibitBlockingMergeOnMongoS: 0 ,

```

```

91     internalQueryMaxAddToSetBytes: 104857600,
92     internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
93   },
94   ok: 1,
95   '$clusterTime': {
96     clusterTime: Timestamp({ t: 1673285143, i: 1 }),
97     signature: {
98       hash: Binary(Buffer.from("0000000000000000000000000000000000000000000000000000000000000000", "hex"), 0),
99       keyId: Long("0")
100    }
101  },
102  operationTime: Timestamp({ t: 1673285143, i: 1 })
103 }

```

Listing 8.13: Test

### C .3 top critic rating

Before the index

```

1  {
2   explainVersion: '1',
3   queryPlanner: {
4     namespace: 'rottenMovies.movie',
5     indexFilterSet: false,
6     parsedQuery: {},
7     queryHash: '33018E32',
8     planCacheKey: '33018E32',
9     maxIndexedOrSolutionsReached: false,
10    maxIndexedAndSolutionsReached: false,
11    maxScansToExplodeReached: false,
12    winningPlan: {
13      stage: 'SORT',
14      sortPattern: { top_critic_rating: 1 },
15      memLimit: 104857600,
16      type: 'simple',
17      inputStage: { stage: 'COLLSCAN', direction: 'forward' }
18    },
19    rejectedPlans: []
20  },
21  executionStats: {
22    executionSuccess: true,
23    nReturned: 14104,
24    executionTimeMillis: 1818,
25    totalKeysExamined: 0,
26    totalDocsExamined: 14104,
27    executionStages: {
28      stage: 'SORT',
29      nReturned: 14104,
30      executionTimeMillisEstimate: 1740,
31      works: 28211,
32      advanced: 14104,
33      needTime: 14106,
34      needYield: 0,
35      saveState: 47,
36      restoreState: 47,
37      isEOF: 1,
38      sortPattern: { top_critic_rating: 1 },
39      memLimit: 104857600,

```

```

40      type: 'simple',
41      totalDataSizeSorted: 262640385,
42      usedDisk: true,
43      spills: 3,
44      inputStage: {
45          stage: 'COLLSCAN',
46          nReturned: 14104,
47          executionTimeMillisEstimate: 0,
48          works: 14106,
49          advanced: 14104,
50          needTime: 1,
51          needYield: 0,
52          saveState: 47,
53          restoreState: 47,
54          isEOF: 1,
55          direction: 'forward',
56          docsExamined: 14104
57      }
58  }
59 },
60 command: {
61     find: 'movie',
62     filter: {},
63     sort: { top_critic_rating: 1 },
64     '$db': 'rottenMovies'
65 },
66 serverInfo: {
67     host: 'Profile2022LARGE10',
68     port: 27017,
69     version: '6.0.3',
70     gitVersion: 'f803681c3ae19817d31958965850193de067c516'
71 },
72 serverParameters: {
73     internalQueryFacetBufferSizeBytes: 104857600,
74     internalQueryFacetMaxOutputDocSizeBytes: 104857600,
75     internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
76     internalDocumentSourceGroupMaxMemoryBytes: 104857600,
77     internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
78     internalQueryProhibitBlockingMergeOnMongoS: 0,
79     internalQueryMaxAddToSetBytes: 104857600,
80     internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
81 },
82 ok: 1,
83 '$clusterTime': {
84     clusterTime: Timestamp({ t: 1673287293, i: 1 }),
85     signature: {
86         hash: Binary(Buffer.from("0000000000000000000000000000000000000000000000000000000000000000", "hex"), 0),
87         keyId: Long("0")
88     }
89 },
90 operationTime: Timestamp({ t: 1673287293, i: 1 })
91 }

```

Listing 8.14: Test

After the index

```

1 {
2     explainVersion: '1',
3     queryPlanner: {
4         namespace: 'rottenMovies.movie',

```

```

5     indexFilterSet: false ,
6     parsedQuery: {},
7     queryHash: '33018E32',
8     planCacheKey: '33018E32',
9     maxIndexedOrSolutionsReached: false ,
10    maxIndexedAndSolutionsReached: false ,
11    maxScansToExplodeReached: false ,
12    winningPlan: {
13      stage: 'FETCH',
14      inputStage: {
15        stage: 'IXSCAN',
16        keyPattern: { top_critic_rating: 1 },
17        indexName: 'top_critic_rating_1',
18        isMultiKey: false ,
19        multiKeyPaths: { top_critic_rating: [] },
20        isUnique: false ,
21        isSparse: false ,
22        isPartial: false ,
23        indexVersion: 2,
24        direction: 'forward',
25        indexBounds: { top_critic_rating: [ '[MinKey, MaxKey]' ] }
26      }
27    },
28    rejectedPlans: []
29  },
30  executionStats: {
31    executionSuccess: true ,
32    nReturned: 14104,
33    executionTimeMillis: 24,
34    totalKeysExamined: 14104,
35    totalDocsExamined: 14104,
36    executionStages: {
37      stage: 'FETCH',
38      nReturned: 14104,
39      executionTimeMillisEstimate: 5,
40      works: 14105,
41      advanced: 14104,
42      needTime: 0,
43      needYield: 0,
44      saveState: 14,
45      restoreState: 14,
46      isEOF: 1,
47      docsExamined: 14104,
48      alreadyHasObj: 0,
49      inputStage: {
50        stage: 'IXSCAN',
51        nReturned: 14104,
52        executionTimeMillisEstimate: 1,
53        works: 14105,
54        advanced: 14104,
55        needTime: 0,
56        needYield: 0,
57        saveState: 14,
58        restoreState: 14,
59        isEOF: 1,
60        keyPattern: { top_critic_rating: 1 },
61        indexName: 'top_critic_rating_1',
62        isMultiKey: false ,
63        multiKeyPaths: { top_critic_rating: [] },
64        isUnique: false ,
65        isSparse: false ,

```

Listing 8.15: Test

#### C .4 user rating

Before the index

```
1 {  
2   explainVersion: '1',  
3   queryPlanner: {  
4     namespace: 'rottenMovies.movie',  
5     indexFilterSet: false,  
6     parsedQuery: {},  
7     queryHash: '3E9B1E6C',  
8     planCacheKey: '3E9B1E6C',  
9     maxIndexedOrSolutionsReached: false,
```

```

10     maxIndexedAndSolutionsReached: false ,
11     maxScansToExplodeReached: false ,
12     winningPlan: {
13       stage: 'SORT' ,
14       sortPattern: { user_rating: 1 } ,
15       memLimit: 104857600,
16       type: 'simple' ,
17       inputStage: { stage: 'COLLSCAN' , direction: 'forward' }
18     },
19     rejectedPlans: []
20   },
21   executionStats: {
22     executionSuccess: true ,
23     nReturned: 14104,
24     executionTimeMillis: 1779,
25     totalKeysExamined: 0,
26     totalDocsExamined: 14104,
27     executionStages: {
28       stage: 'SORT' ,
29       nReturned: 14104,
30       executionTimeMillisEstimate: 1698,
31       works: 28211,
32       advanced: 14104,
33       needTime: 14106,
34       needYield: 0,
35       saveState: 45,
36       restoreState: 45,
37       isEOF: 1,
38       sortPattern: { user_rating: 1 } ,
39       memLimit: 104857600,
40       type: 'simple' ,
41       totalDataSizeSorted: 262640385,
42       usedDisk: true ,
43       spills: 3,
44       inputStage: {
45         stage: 'COLLSCAN' ,
46         nReturned: 14104,
47         executionTimeMillisEstimate: 0,
48         works: 14106,
49         advanced: 14104,
50         needTime: 1,
51         needYield: 0,
52         saveState: 45,
53         restoreState: 45,
54         isEOF: 1,
55         direction: 'forward' ,
56         docsExamined: 14104
57       }
58     }
59   },
60   command: {
61     find: 'movie' ,
62     filter: {},
63     sort: { user_rating: 1 } ,
64     '$db': 'rottenMovies' ,
65   },
66   serverInfo: {
67     host: 'Profile2022LARGE10' ,
68     port: 27017,
69     version: '6.0.3' ,
70     gitVersion: 'f803681c3ae19817d31958965850193de067c516' ,

```

```

71 },
72 serverParameters: {
73     internalQueryFacetBufferSizeBytes: 104857600,
74     internalQueryFacetMaxOutputDocSizeBytes: 104857600,
75     internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
76     internalDocumentSourceGroupMaxMemoryBytes: 104857600,
77     internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
78     internalQueryProhibitBlockingMergeOnMongoS: 0,
79     internalQueryMaxAddToSetBytes: 104857600,
80     internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
81 },
82 ok: 1,
83 '$clusterTime': {
84     clusterTime: Timestamp({ t: 1673287353, i: 1 }),
85     signature: {
86         hash: Binary(Buffer.from("0000000000000000000000000000000000000000000000000000000000000000", "hex"), 0),
87         keyId: Long("0")
88     }
89 },
90 operationTime: Timestamp({ t: 1673287353, i: 1 })
91 }

```

Listing 8.16: Test

After the index

```

1 {
2     explainVersion: '1',
3     queryPlanner: {
4         namespace: 'rottenMovies.movie',
5         indexFilterSet: false,
6         parsedQuery: {},
7         queryHash: '3E9B1E6C',
8         planCacheKey: '3E9B1E6C',
9         maxIndexedOrSolutionsReached: false,
10        maxIndexedAndSolutionsReached: false,
11        maxScansToExplodeReached: false,
12        winningPlan: {
13            stage: 'FETCH',
14            inputStage: {
15                stage: 'IXSCAN',
16                keyPattern: { user_rating: 1 },
17                indexName: 'user_rating_1',
18                isMultiKey: false,
19                multiKeyPaths: { user_rating: [] },
20                isUnique: false,
21                isSparse: false,
22                isPartial: false,
23                indexVersion: 2,
24                direction: 'forward',
25                indexBounds: { user_rating: [ '[MinKey, MaxKey]' ] }
26            }
27        },
28        rejectedPlans: []
29    },
30    executionStats: {
31        executionSuccess: true,
32        nReturned: 14104,
33        executionTimeMillis: 25,
34        totalKeysExamined: 14104,
35        totalDocsExamined: 14104,

```

```

36     executionStages: {
37         stage: 'FETCH',
38         nReturned: 14104,
39         executionTimeMillisEstimate: 5,
40         works: 14105,
41         advanced: 14104,
42         needTime: 0,
43         needYield: 0,
44         saveState: 14,
45         restoreState: 14,
46         isEOF: 1,
47         docsExamined: 14104,
48         alreadyHasObj: 0,
49         inputStage: {
50             stage: 'IXSCAN',
51             nReturned: 14104,
52             executionTimeMillisEstimate: 2,
53             works: 14105,
54             advanced: 14104,
55             needTime: 0,
56             needYield: 0,
57             saveState: 14,
58             restoreState: 14,
59             isEOF: 1,
60             keyPattern: { user_rating: 1 },
61             indexName: 'user_rating_1',
62             isMultiKey: false,
63             multiKeyPaths: { user_rating: [] },
64             isUnique: false,
65             isSparse: false,
66             isPartial: false,
67             indexVersion: 2,
68             direction: 'forward',
69             indexBounds: { user_rating: [ '[MinKey, MaxKey]' ] },
70             keysExamined: 14104,
71             seeks: 1,
72             dupsTested: 0,
73             dupsDropped: 0
74         }
75     }
76 },
77 command: {
78     find: 'movie',
79     filter: {},
80     sort: { user_rating: 1 },
81     '$db': 'rottenMovies'
82 },
83 serverInfo: {
84     host: 'Profile2022LARGE10',
85     port: 27017,
86     version: '6.0.3',
87     gitVersion: 'f803681c3ae19817d31958965850193de067c516',
88 },
89 serverParameters: {
90     internalQueryFacetBufferSizeBytes: 104857600,
91     internalQueryFacetMaxOutputDocSizeBytes: 104857600,
92     internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
93     internalDocumentSourceGroupMaxMemoryBytes: 104857600,
94     internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
95     internalQueryProhibitBlockingMergeOnMongoS: 0,
96     internalQueryMaxAddToSetBytes: 104857600,

```

```

97     internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
98   },
99   ok: 1,
100  '$clusterTime': {
101    clusterTime: Timestamp({ t: 1673285383, i: 1 }),
102    signature: {
103      hash: Binary(Buffer.from("0000000000000000000000000000000000000000000000000000000000000000", "hex"), 0),
104      keyId: Long("0")
105    }
106  },
107  operationTime: Timestamp({ t: 1673285383, i: 1 })
108 }

```

Listing 8.17: Test

## C .5 personnel.primaryName

Before the index

```

1  {
2   explainVersion: '1',
3   queryPlanner: {
4     namespace: 'rottenMovies.movie',
5     indexFilterSet: false,
6     parsedQuery: { 'personnel.primaryName': { '$eq': '' } },
7     queryHash: 'E212F03B',
8     planCacheKey: 'E212F03B',
9     maxIndexedOrSolutionsReached: false,
10    maxIndexedAndSolutionsReached: false,
11    maxScansToExplodeReached: false,
12    winningPlan: {
13      stage: 'COLLSCAN',
14      filter: { 'personnel.primaryName': { '$eq': '' } },
15      direction: 'forward'
16    },
17    rejectedPlans: []
18  },
19  executionStats: {
20    executionSuccess: true,
21    nReturned: 0,
22    executionTimeMillis: 47,
23    totalKeysExamined: 0,
24    totalDocsExamined: 14104,
25    executionStages: {
26      stage: 'COLLSCAN',
27      filter: { 'personnel.primaryName': { '$eq': '' } },
28      nReturned: 0,
29      executionTimeMillisEstimate: 9,
30      works: 14106,
31      advanced: 0,
32      needTime: 14105,
33      needYield: 0,
34      saveState: 14,
35      restoreState: 14,
36      isEOF: 1,
37      direction: 'forward',
38      docsExamined: 14104
39    }
40  },

```

```

41   command: {
42     find: 'movie',
43     filter: { 'personnel.primaryName': '' },
44     '$db': 'rottenMovies'
45   },
46   serverInfo: {
47     host: 'Profile2022LARGE10',
48     port: 27017,
49     version: '6.0.3',
50     gitVersion: 'f803681c3ae19817d31958965850193de067c516',
51   },
52   serverParameters: {
53     internalQueryFacetBufferSizeBytes: 104857600,
54     internalQueryFacetMaxOutputDocSizeBytes: 104857600,
55     internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
56     internalDocumentSourceGroupMaxMemoryBytes: 104857600,
57     internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
58     internalQueryProhibitBlockingMergeOnMongoS: 0,
59     internalQueryMaxAddToSetBytes: 104857600,
60     internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
61   },
62   ok: 1,
63   '$clusterTime': {
64     clusterTime: Timestamp({ t: 1673287593, i: 1 }),
65     signature: {
66       hash: Binary(Buffer.from("0000000000000000000000000000000000000000000000000000000000000000", "hex")),
67       keyId: Long("0")
68     }
69   },
70   operationTime: Timestamp({ t: 1673287593, i: 1 })
71 }

```

Listing 8.18: Test

After the index

```

1  {
2   explainVersion: '1',
3   queryPlanner: {
4     namespace: 'rottenMovies.movie',
5     indexFilterSet: false,
6     parsedQuery: { 'personnel.primaryName': { '$eq': '' } },
7     queryHash: 'E212F03B',
8     planCacheKey: '9D4A6814',
9     maxIndexedOrSolutionsReached: false,
10    maxIndexedAndSolutionsReached: false,
11    maxScansToExplodeReached: false,
12    winningPlan: {
13      stage: 'FETCH',
14      inputStage: {
15        stage: 'IXSCAN',
16        keyPattern: { 'personnel.primaryName': 1 },
17        indexName: 'personnel.primaryName_1',
18        isMultiKey: true,
19        multiKeyPaths: { 'personnel.primaryName': [ 'personnel' ] },
20        isUnique: false,
21        isSparse: false,
22        isPartial: false,
23        indexVersion: 2,
24        direction: 'forward',
25        indexBounds: { 'personnel.primaryName': [ "", "" ] } }

```

```

26     }
27   },
28   rejectedPlans: []
29 },
30 executionStats: {
31   executionSuccess: true,
32   nReturned: 0,
33   executionTimeMillis: 0,
34   totalKeysExamined: 0,
35   totalDocsExamined: 0,
36   executionStages: {
37     stage: 'FETCH',
38     nReturned: 0,
39     executionTimeMillisEstimate: 0,
40     works: 1,
41     advanced: 0,
42     needTime: 0,
43     needYield: 0,
44     saveState: 0,
45     restoreState: 0,
46     isEOF: 1,
47     docsExamined: 0,
48     alreadyHasObj: 0,
49     inputStage: {
50       stage: 'IXSCAN',
51       nReturned: 0,
52       executionTimeMillisEstimate: 0,
53       works: 1,
54       advanced: 0,
55       needTime: 0,
56       needYield: 0,
57       saveState: 0,
58       restoreState: 0,
59       isEOF: 1,
60       keyPattern: { 'personnel.primaryName': 1 },
61       indexName: 'personnel.primaryName_1',
62       isMultiKey: true,
63       multiKeyPaths: { 'personnel.primaryName': [ 'personnel' ] },
64       isUnique: false,
65       isSparse: false,
66       isPartial: false,
67       indexVersion: 2,
68       direction: 'forward',
69       indexBounds: { 'personnel.primaryName': [ ['', ''], ] },
70       keysExamined: 0,
71       seeks: 1,
72       dupsTested: 0,
73       dupsDropped: 0
74     }
75   }
76 },
77 command: {
78   find: 'movie',
79   filter: { 'personnel.primaryName': '' },
80   '$db': 'rottenMovies'
81 },
82 serverInfo: {
83   host: 'Profile2022LARGE10',
84   port: 27017,
85   version: '6.0.3',
86   gitVersion: 'f803681c3ae19817d31958965850193de067c516'

```

Listing 8.19: Test

## D MongoDB indexes:User collection

## D .1 username

Before the index

```
1 {
2     explainVersion: '1',
3     queryPlanner: {
4         namespace: 'rottenMovies.user',
5         indexFilterSet: false,
6         parsedQuery: { username: { '$eq': 'Abbie Bernstein' } },
7         queryHash: '7D9BB680',
8         planCacheKey: '7D9BB680',
9         maxIndexedOrSolutionsReached: false,
10        maxIndexedAndSolutionsReached: false,
11        maxScansToExplodeReached: false,
12        winningPlan: {
13            stage: 'COLLSCAN',
14            filter: { username: { '$eq': 'Abbie Bernstein' } },
15            direction: 'forward',
16        },
17        rejectedPlans: []
18    },
19    executionStats: {
20        executionSuccess: true,
21        nReturned: 1,
22        executionTimeMillis: 6,
23        totalKeysExamined: 0,
24        totalDocsExamined: 8339,
25        executionStages: {
26            stage: 'COLLSCAN',
27            filter: { username: { '$eq': 'Abbie Bernstein' } },
28            nReturned: 1,
```

Listing 8.20: Test

After the index

```
1  {
2      explainVersion: '1',
3      queryPlanner: {
4          namespace: 'rottenMovies.user',
5          indexFilterSet: false,
6          parsedQuery: { username: { '$eq': 'Abbie Bernstein' } },
7          queryHash: '7D9BB680',
8          planCacheKey: '24069050',
9          maxIndexedOrSolutionsReached: false,
10         maxIndexedAndSolutionsReached: false,
11         maxScansToExplodeReached: false,
12         winningPlan: {
13             stage: 'FETCH',
```

```

14     inputStage: {
15         stage: 'IXSCAN',
16         keyPattern: { username: 1 },
17         indexName: 'username_1',
18         isMultiKey: false,
19         multiKeyPaths: { username: [] },
20         isUnique: false,
21         isSparse: false,
22         isPartial: false,
23         indexVersion: 2,
24         direction: 'forward',
25         indexBounds: { username: [ '['"Abbie Bernstein", "Abbie Bernstein"]' ]
26             }
27         }
28     },
29     rejectedPlans: []
30 },
31 executionStats: {
32     executionSuccess: true,
33     nReturned: 1,
34     executionTimeMillis: 1,
35     totalKeysExamined: 1,
36     totalDocsExamined: 1,
37     executionStages: {
38         stage: 'FETCH',
39         nReturned: 1,
40         executionTimeMillisEstimate: 1,
41         works: 2,
42         advanced: 1,
43         needTime: 0,
44         needYield: 0,
45         saveState: 0,
46         restoreState: 0,
47         isEOF: 1,
48         docsExamined: 1,
49         alreadyHasObj: 0,
50         inputStage: {
51             stage: 'IXSCAN',
52             nReturned: 1,
53             executionTimeMillisEstimate: 1,
54             works: 2,
55             advanced: 1,
56             needTime: 0,
57             needYield: 0,
58             saveState: 0,
59             restoreState: 0,
60             isEOF: 1,
61             keyPattern: { username: 1 },
62             indexName: 'username_1',
63             isMultiKey: false,
64             multiKeyPaths: { username: [] },
65             isUnique: false,
66             isSparse: false,
67             isPartial: false,
68             indexVersion: 2,
69             direction: 'forward',
70             indexBounds: { username: [ '['"Abbie Bernstein", "Abbie Bernstein"]' ]
71         },
72         keysExamined: 1,
73         seeks: 1,
74         dupsTested: 0,

```

```

73         dupsDropped: 0
74     }
75   }
76 },
77 command: {
78   find: 'user',
79   filter: { username: 'Abbie Bernstein' },
80   '$db': 'rottenMovies'
81 },
82 serverInfo: {
83   host: 'Profile2022LARGE10',
84   port: 27017,
85   version: '6.0.3',
86   gitVersion: 'f803681c3ae19817d31958965850193de067c516'
87 },
88 serverParameters: {
89   internalQueryFacetBufferSizeBytes: 104857600,
90   internalQueryFacetMaxOutputDocSizeBytes: 104857600,
91   internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
92   internalDocumentSourceGroupMaxMemoryBytes: 104857600,
93   internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
94   internalQueryProhibitBlockingMergeOnMongoS: 0,
95   internalQueryMaxAddToSetBytes: 104857600,
96   internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
97 },
98 ok: 1,
99 '$clusterTime': {
100   clusterTime: Timestamp({ t: 1673285013, i: 1 }),
101   signature: {
102     hash: Binary(Buffer.from("0000000000000000000000000000000000000000000000000000000000000000", "hex"), 0),
103     keyId: Long("0")
104   }
105 },
106 operationTime: Timestamp({ t: 1673285013, i: 1 })
107 }

```

Listing 8.21: Test

## D .2 date of birth

Before the index

```

1 {
2   explainVersion: '1',
3   queryPlanner: {
4     namespace: 'rottenMovies.user',
5     indexFilterSet: false,
6     parsedQuery: {
7       date_of_birth: {
8         '$eq': 'Mon Jan 09 2023 17:05:07 GMT+0000 (Western European Standard
9         Time)'
10      }
11    },
12    queryHash: 'D7A0117C',
13    planCacheKey: 'D7A0117C',
14    maxIndexedOrSolutionsReached: false,
15    maxIndexedAndSolutionsReached: false,
16    maxScansToExplodeReached: false,
17    winningPlan: {

```

```

17         stage: 'COLLSCAN',
18         filter: {
19             date_of_birth: {
20                 '$eq': 'Mon Jan 09 2023 17:05:07 GMT+0000 (Western European
21                 Standard Time)'
22             }
23         },
24         direction: 'forward'
25     },
26     rejectedPlans: []
27 },
28 executionStats: {
29     executionSuccess: true,
30     nReturned: 0,
31     executionTimeMillis: 32,
32     totalKeysExamined: 0,
33     totalDocsExamined: 8339,
34     executionStages: {
35         stage: 'COLLSCAN',
36         filter: {
37             date_of_birth: {
38                 '$eq': 'Mon Jan 09 2023 17:05:07 GMT+0000 (Western European
39                 Standard Time)'
40             }
41         },
42         nReturned: 0,
43         executionTimeMillisEstimate: 23,
44         works: 8341,
45         advanced: 0,
46         needTime: 8340,
47         needYield: 0,
48         saveState: 8,
49         restoreState: 8,
50         isEOF: 1,
51         direction: 'forward',
52         docsExamined: 8339
53     }
54 },
55 command: {
56     find: 'user',
57     filter: {
58         date_of_birth: 'Mon Jan 09 2023 17:05:07 GMT+0000 (Western European
59         Standard Time)'
60     },
61     '$db': 'rottenMovies'
62 },
63 serverInfo: {
64     host: 'Profile2022LARGE10',
65     port: 27017,
66     version: '6.0.3',
67     gitVersion: 'f803681c3ae19817d31958965850193de067c516',
68 },
69 serverParameters: {
70     internalQueryFacetBufferSizeBytes: 104857600,
71     internalQueryFacetMaxOutputDocSizeBytes: 104857600,
72     internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
73     internalDocumentSourceGroupMaxMemoryBytes: 104857600,
74     internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
75     internalQueryProhibitBlockingMergeOnMongoS: 0,
76     internalQueryMaxAddToSetBytes: 104857600,
77     internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
78 }
```

Listing 8.22: Test

After the index

```
1 {
2     explainVersion: '1',
3     queryPlanner: {
4         namespace: 'rottenMovies.user',
5         indexFilterSet: false,
6         parsedQuery: {
7             date_of_birth: {
8                 '$eq': 'Mon Jan 09 2023 17:24:42 GMT+0000 (Western European Standard
9                 Time)'
10            }
11        },
12        queryHash: 'D7A0117C',
13        planCacheKey: '90F68BB6',
14        maxIndexedOrSolutionsReached: false,
15        maxIndexedAndSolutionsReached: false,
16        maxScansToExplodeReached: false,
17        winningPlan: {
18            stage: 'FETCH',
19            inputStage: {
20                stage: 'IXSCAN',
21                keyPattern: { date_of_birth: 1 },
22                indexName: 'date_of_birth_1',
23                isMultiKey: false,
24                multiKeyPaths: { date_of_birth: [] },
25                isUnique: false,
26                isSparse: false,
27                isPartial: false,
28                indexVersion: 2,
29                direction: 'forward',
30                indexBounds: {
31                    date_of_birth: [
32                        ["Mon Jan 09 2023 17:24:42 GMT+0000 (Western European Standard
33                        Time)", "Mon Jan 09 2023 17:24:42 GMT+0000 (Western European Standard
34                        Time)"]
35                    ]
36                }
37            }
38        },
39        rejectedPlans: []
40    },
41    executionStats: {
42        executionSuccess: true,
43        nReturned: 0,
44        executionTimeMillis: 1,
45        totalKeysExamined: 0,
```

```

43     totalDocsExamined: 0,
44     executionStages: {
45         stage: 'FETCH',
46         nReturned: 0,
47         executionTimeMillisEstimate: 0,
48         works: 1,
49         advanced: 0,
50         needTime: 0,
51         needYield: 0,
52         saveState: 0,
53         restoreState: 0,
54         isEOF: 1,
55         docsExamined: 0,
56         alreadyHasObj: 0,
57         inputStage: {
58             stage: 'IXSCAN',
59             nReturned: 0,
60             executionTimeMillisEstimate: 0,
61             works: 1,
62             advanced: 0,
63             needTime: 0,
64             needYield: 0,
65             saveState: 0,
66             restoreState: 0,
67             isEOF: 1,
68             keyPattern: { date_of_birth: 1 },
69             indexName: 'date_of_birth_1',
70             isMultiKey: false,
71             multiKeyPaths: { date_of_birth: [] },
72             isUnique: false,
73             isSparse: false,
74             isPartial: false,
75             indexVersion: 2,
76             direction: 'forward',
77             indexBounds: {
78                 date_of_birth: [
79                     '["Mon Jan 09 2023 17:24:42 GMT+0000 (Western European Standard Time)", "Mon Jan 09 2023 17:24:42 GMT+0000 (Western European Standard Time)"]',
80                     ]
81                 },
82                 keysExamined: 0,
83                 seeks: 1,
84                 dupsTested: 0,
85                 dupsDropped: 0
86             }
87         }
88     },
89     command: {
90         find: 'user',
91         filter: {
92             date_of_birth: 'Mon Jan 09 2023 17:24:42 GMT+0000 (Western European Standard Time)',
93         },
94         '$db': 'rottenMovies',
95     },
96     serverInfo: {
97         host: 'Profile2022LARGE10',
98         port: 27017,
99         version: '6.0.3',
100        gitVersion: 'f803681c3ae19817d31958965850193de067c516'

```

```

101 },
102 serverParameters: {
103     internalQueryFacetBufferSizeBytes: 104857600,
104     internalQueryFacetMaxOutputDocSizeBytes: 104857600,
105     internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
106     internalDocumentSourceGroupMaxMemoryBytes: 104857600,
107     internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
108     internalQueryProhibitBlockingMergeOnMongoS: 0,
109     internalQueryMaxAddToSetBytes: 104857600,
110     internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
111 },
112 ok: 1,
113 '$clusterTime': {
114     clusterTime: Timestamp({ t: 1673285073, i: 1 }),
115     signature: {
116         hash: Binary(Buffer.from("0000000000000000000000000000000000000000000000000000000000000000", "hex"), 0),
117         keyId: Long("0")
118     }
119 },
120 operationTime: Timestamp({ t: 1673285073, i: 1 })
121 }

```

Listing 8.23: Test

## E Application code

### E .1 Pom.xml

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
4   <modelVersion>4.0.0</modelVersion>
5   <parent>
6     <groupId>org.springframework.boot</groupId>
7     <artifactId>spring-boot-starter-parent</artifactId>
8     <version>3.0.0</version>
9     <relativePath/> <!-- lookup parent from repository -->
10  </parent>
11  <groupId>it.unipi.dii.lsmsdb</groupId>
12  <artifactId>rottenMovies</artifactId>
13  <version>0.0.1-SNAPSHOT</version>
14  <name>rottenMovies</name>
15  <description>Project for the rotten movies service</description>
16  <properties>
17    <java.version>19</java.version>
18  </properties>
19  <dependencies>
20    <dependency>
21      <groupId>org.springframework.boot</groupId>
22      <artifactId>spring-boot-starter-thymeleaf</artifactId>
23    </dependency>
24    <dependency>
25      <groupId>org.springframework.boot</groupId>
26      <artifactId>spring-boot-starter-web</artifactId>
27    </dependency>

```

```

28      <dependency>
29          <groupId>org.springframework.boot</groupId>
30          <artifactId>spring-boot-starter-data-mongodb</artifactId>
31      </dependency>
32      <dependency>
33          <groupId>org.springframework.boot</groupId>
34          <artifactId>spring-boot-starter-data-mongodb-reactive</artifactId>
35      </dependency>
36      <dependency>
37          <groupId>org.springframework.boot</groupId>
38          <artifactId>spring-boot-starter-data-neo4j</artifactId>
39      </dependency>
40
41      <dependency>
42          <groupId>org.springframework.boot</groupId>
43          <artifactId>spring-boot-starter-test</artifactId>
44          <scope>test</scope>
45      </dependency>
46      <dependency>
47          <groupId>io.projectreactor</groupId>
48          <artifactId>reactor-test</artifactId>
49          <scope>test</scope>
50      </dependency>
51      <dependency>
52          <groupId>com.google.code.gson</groupId>
53          <artifactId>gson</artifactId>
54          <version>2.10</version>
55      </dependency>
56      <dependency>
57          <groupId>com.fasterxml.jackson.core</groupId>
58          <artifactId>jackson-annotations</artifactId>
59          <version>2.14.1</version>
60      </dependency>
61      <dependency>
62          <groupId>com.fasterxml.jackson.core</groupId>
63          <artifactId>jackson-databind</artifactId>
64          <version>2.14.0</version>
65      </dependency>
66      <dependency>
67          <groupId>org.neo4j.driver</groupId>
68          <artifactId>neo4j-java-driver</artifactId>
69          <version>5.3.0</version>
70      </dependency>
71  </dependencies>
72
73  <build>
74      <plugins>
75          <plugin>
76              <groupId>org.springframework.boot</groupId>
77              <artifactId>spring-boot-maven-plugin</artifactId>
78          </plugin>
79      </plugins>
80  </build>
81
82 </project>

```

Listing 8.24: Test