

Large-Scale and Multi-Structure Database

Reviook Application Report

Application developed by  
Arancio Febbo Salvatore, Di Donato Mattia, Giorgi Matteo

Summary

[Introduction 4](#_Toc95898086)

[Functional and Non-Functional requirements 5](#_Toc95898087)

[Dataset 6](#_Toc95898088)

[Use Case Diagram 7](#_Toc95898089)

[Class Analysis 8](#_Toc95898090)

[Classes Relationship 9](#_Toc95898091)

[Classes Definitions 9](#_Toc95898092)

[Classes Attributes 9](#_Toc95898093)

[Query Analysis 11](#_Toc95898094)

[Data Model 16](#_Toc95898095)

[MongoDB – Document Organization 16](#_Toc95898096)

[Authors Collection 16](#_Toc95898097)

[Users Collection 17](#_Toc95898098)

[Books Collection 18](#_Toc95898099)

[Genres Collection 19](#_Toc95898100)

[Report Collection 20](#_Toc95898101)

[Log Collection 20](#_Toc95898102)

[Neo4J – Nodes Organization 22](#_Toc95898103)

[Implementation 23](#_Toc95898104)

[Cache 24](#_Toc95898105)

[JAVA Entity 24](#_Toc95898106)

[Database Consistency Management 27](#_Toc95898107)

[Database Organization on Machines 29](#_Toc95898108)

[Replica Configuration 29](#_Toc95898109)

[Replica crash 30](#_Toc95898110)

[Neo4J CRUD Operations 31](#_Toc95898111)

[Create 31](#_Toc95898112)

[Read 33](#_Toc95898113)

[Delete 34](#_Toc95898114)

[Crud operations MongoDB 36](#_Toc95898115)

[Neo4J Analytics & Suggestions 51](#_Toc95898116)

[MongoDB Analitycs Implementations 55](#_Toc95898117)

[Neo4J Index Analysis 59](#_Toc95898118)

[author\_username 59](#_Toc95898119)

[user\_username 60](#_Toc95898120)

[user\_username + book\_id 62](#_Toc95898121)

[Mongo Index Analysis 66](#_Toc95898122)

[Authors and Users Collections test 66](#_Toc95898123)

[Books Collection test 67](#_Toc95898124)

[Sharding proposal 70](#_Toc95898125)

# Introduction

To implement our project, we use Java and Maven with the support of Scene Builder to manage fxml files for the interface of the application.

Data have been stored into MongoDB and Neo4J database. MongoDB was used for store and retrieve information about Users, Authors and Books while the task of storing and retrieving social information that associate Users and Books was assigned to Neo4J.

This application was developed to provide authors the possibility to share their titles and receive from readers the feedback on their works, for this reason the name of the application came from the union of Review and Book. More in detail a user/author can register/login, can search for other users/authors profiles and view statistics of them profile, view suggested users and books, view a book, and leave a review about it. Users and authors have on their profile a useful section for the ranking analytics.

An additional actor of the application is the administrator that have the possibility to search and view authors/users/books profile as the other actor, but has also the capacity to delete reviews, books and user’s/author’s account.

[The application repository is on GitHub](https://github.com/mattiadido95/Reviook)

# Functional and Non-Functional requirements

This paragraph describes all the functional and not functional requirements that the Reviook application satisfy.

The application manages three kinds of actors: User, Author, Admin.

* A user has the access to basic function of the application
* An author can also add and delete his/her books
* An admin has all the function of di application except the social ones and can also decide to delete users/authors/reviews/books.

Functional Requirements

* An unregister user can only sign up to become a registered user.

A **user** is allowed to:

* Login/Register
* Logout
* View its homepage
* Delete or modify its account
* View Ranking statistics
* Browse information about:
  + Users
  + Authors
  + Books
* View users/authors profile
* Follow or unfollow another user/author
* View book profile
* Insert a book into read or to\_read list
* Review a book
* Like a review
* Report books/reviews
* Modify delete its own review
* View suggested:
  + Users
  + Authors
  + Books

An **author** is a **user** who is also allowed to:

* Add new books
* Delete its books

An **admin** is allowed to:

* Browse information about:
  + - Users
    - Authors
* Browse reviews and books reported
* Delete a book
* Delete a review
* Delete user/author account

Non-Functional requirements

* The application should be simple, intuitive and fast in response.
* The password must be protected and stored in an encrypted form and must be between 8-20 characters.
* The user’s and author’s username must be unique.
* The application should store locally information to reduce network traffic and DBs load.
* The code must be easy to maintain.
* Data may not be updated for up to 10 minutes

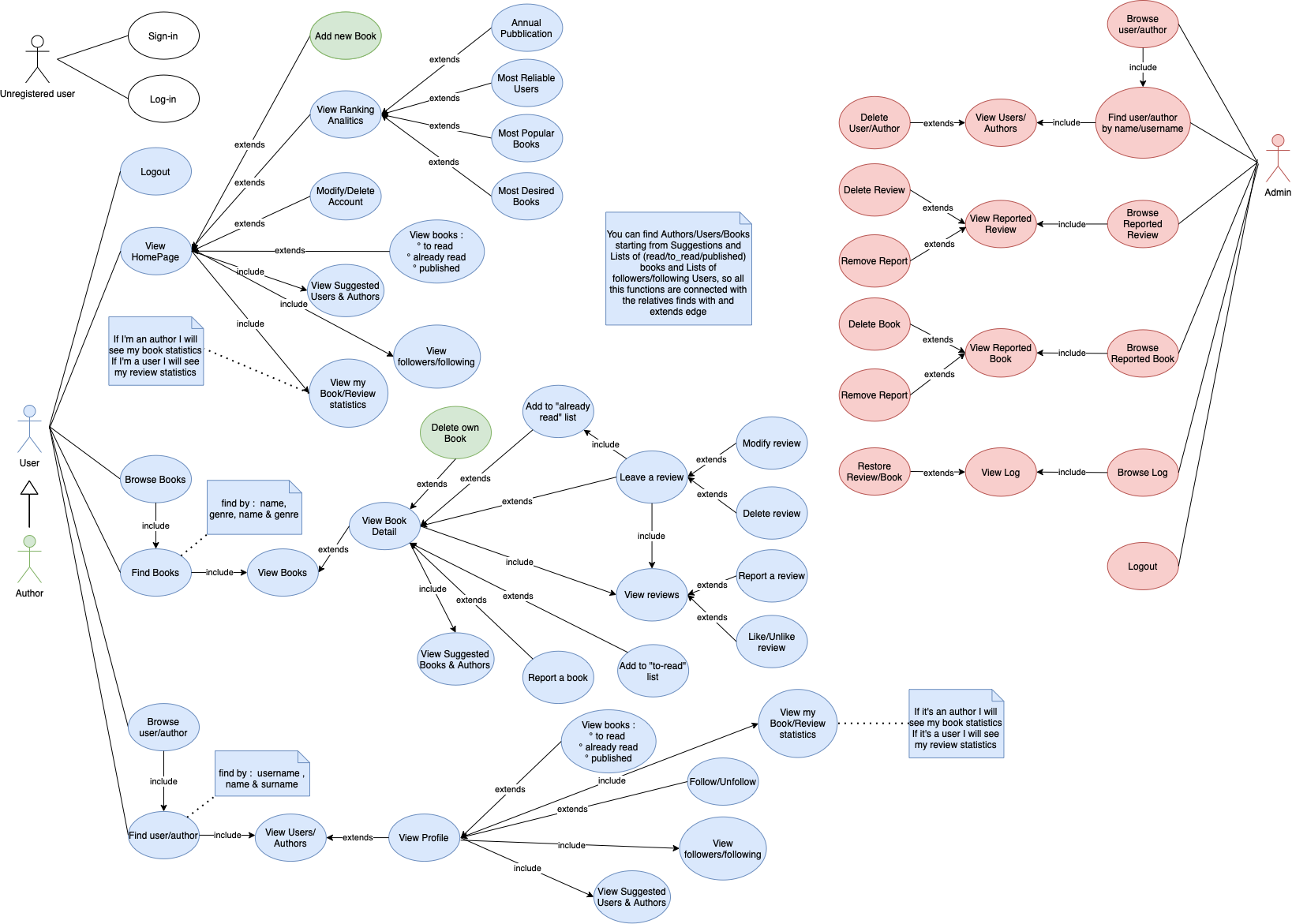
# Dataset

We collect information about books, reviews, authors and users from two different sources : <https://snap.stanford.edu/data/amazon-meta.html> <https://sites.google.com/eng.ucsd.edu/ucsdbookgraph/home> .

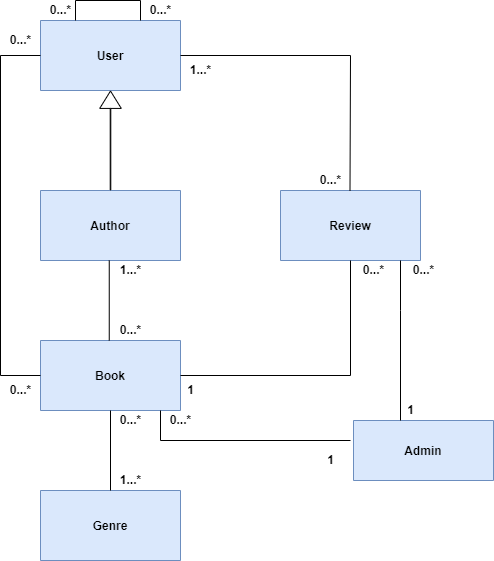
The first one is a plain text dataset of Amazon product. From it we extract information about books purchased and reviewed by users. We wrote a program ([GitHub](https://github.com/mattiadido95/JSONconverter)) to reorganize data in a json format and at the same time scrape some missing information. The second is a collection of data from goodreads site, it is already formatted in json and from it we extract similar information of the first one.

The reviews of two dataset were merged in case we found the same ISBN. The two datasets were quite big, so we decided to take from goodreads the books from 2013 to 2017 that have reviews. Some information about authors and users were generated randomly using [java-faker](https://github.com/DiUS/java-faker). We also used the UUID in order to create a different unique identifier for new entity that can been created during the execution of the application.

# Use Case Diagram



# Class Analysis



## Classes Relationship

1. Each User can create more than 1 Review for each Book
2. One Review is associated only to 1 User and 1 Book
3. An Author can publish more than 1 Book
4. A Book has 1 or many Authors
5. An Admin is connected to a Book or a Review if it has been reported
6. A Book ha 1 or many Genres
7. A User can follow 0 or many users
8. A User can have 0 or many books into his/her books list

## Classes Definitions

|  |  |
| --- | --- |
| Class | Description |
| User | A standard user who can interact with most of its functionalities |
| Author | A special user who can add new books |
| Admin | A user who has the task of checking the correct behavior of the users described above |
| Book | Contains information about a book published by an author |
| Review | Contains information about reviews left by users and authors |
| Genre | The genre of books |

## 

## Classes Attributes

|  |  |  |
| --- | --- | --- |
|  | User |  |
| Attribute | Type | Description |
| id | String | Unique string that identifies a user to match the review |
| name | String | User’s name |
| nickname | String | Unique string that identifies a user in the login |
| email | String | User’s e-mail |
| password | String | User’s password in hash |
| interactions | Interactions | Contains the list of followers and followers of the user |
| listReviewID | ArrayList<String> | List of the ids of the reviews to which the likes have been put |
| follower\_count | Integer | User's follower counter |
| statistics | ArrayList<Genre> | Average of reviews left by the user by genre |
| listBook | ListBooks | Contains the list of “read” and “to read” of the user |

|  |  |  |
| --- | --- | --- |
|  | Author |  |
| Attribute | Type | Description |
| same as the superclass | | |
| published | ArrayList<Book> | List of books published by the authors with all the appropriate characteristics |

|  |  |  |
| --- | --- | --- |
|  | Book |  |
| Attribute | Type | Description |
| isbn | String | Unique string that identifies a book (sometimes missing) |
| language\_code | String | Defines the language of the book |
| asin | String | Unique string that identifies a book (sometimes missing) |
| average\_rating | Double | Average rating of reviews |
| description | String | Book’s description |
| num\_pages | Integer | Book’s pages |
| publication\_day | Integer | Publication day of the book |
| publication\_month | Integer | Publication m month of the book |
| publication\_year | Integer | Publication year of the book |
| image\_url | String | Book cover |
| book\_id | String | Unique string that identifies a user to be associated with the review and authors |
| ratings\_count | Integer | Total reviews |
| title | String | Book title |
| authors | ArrayList<Author> | List of all possible authors and co-authors of a book |
| genres | ArrayList<String> | List of all possible genres of a book |
| reviews | ArrayList<Review> | List of all possible reviews of a book |

|  |  |  |
| --- | --- | --- |
|  | Review |  |
| Attribute | Type | Description |
| review\_id | String | Review id |
| date\_update | String | Date the review was last modified |
| user\_id | String | User id |
| username | String | Username id |
| rating | String | Rating of the review |
| review\_text | String | Text of the reviews |
| likes | Integer | Total number of likes for a review |
| liked | Boolean | It is set to true when the review id is present in the user's listReviewID attribute |

|  |  |  |
| --- | --- | --- |
|  | Genre |  |
| Attribute | Type | Description |
| type | String | Type of the genre |
| value | Double | Used for statistics |

# Query Analysis

## Volume Table

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Number of instances | Explanations |
| Book | Entity | 275232 | According to some online statistics in the 2013 the total amount of published books in US was 275232 |
| Author | Entity | 96008 | In Goodreads dataset we have a total of 96008 authors that published during 2013 |
| Review | Entity | 1466018 | In Goodreads dataset we have a total of 163752 books published in that year and a total of 15,7M of review in 18 years so if we decide to maintain this proportion we have that 163752 : 275232  (books) = 872222 : x (reviews) so 1466018 number of reviews |
| User | Entity | 43450 | Again, considering Goodreads we have a total number of users of 465323 registered in 18 years. Therefore 872222 : 1466018 (reviews) = 25851 : x (users) so 43450 number of users |
| Read / To-read | Relationship | 24 | An average reader can read 12 books per year |
| Follow / Following | Relationship | 300 | Very difficult to predict, but some online statistics says that it could be around 150 |

## MongoDB

|  |  |  |
| --- | --- | --- |
| Read Operation | | |
| Query | Frequency | Cost |
| Browse all books | Low  (users/authors don’t have much interest to find all books without any filter) | Very High (multiple reads)  For instance, at the end of the first year we will have 275232  of books |
| Browse books by title | High  (users/authors are usually interested to find a specific book) | Average  (if you don’t insert the complete title you will retrieve more than 1 book) |
| Browse books by genre | Average  (users/authors are usually interested to find book that belongs to a genre which he/she likes) | High (multiple reads)  For instance, if we consider 8 macro categories of books we have 275232  / 8 = 34404 books |
| Browse books by title and genre | Average  (users/authors can easily find a book knowing it’s category and a part or the complete title) | Low  (if there aren’t any other books with the same title, we have only 1 read) |
| Browse User/Author by name | High  (users/authors usually search each other for following or for view other’s profile) | Low  (considering the full name we might find homonyms but the number of reads are quite low) |
| View book information and related review | High  (each time a user/author search for a book, he/she probably also wants to learn more about it) | Average-High (multiple reads)  We have to retrieve the reviews of a books. If we take in consideration the instances of a year we have 1466018/ 275232 = 5 reviews for each books |
| View user/author profile | Average-High  (each time a user/author search for a another user, he/she probably also wants to learn more about him/her) | Low (1 read)  We can easily find in all the information related to a user after the browse |
| Ranks users with more frequent valid reviews based on the likes received | Average | High (complex aggregations) |
| View for each author, average rating on each published category | High  (these information are retrieved each time we wants to see a profile of an author) | High (complex aggregations) |
| View for each user, average rating given to each category | High  (these information are retrieved each time we wants to see a profile of a user) | High (complex aggregations) |
| View for each year and category the number of published books | Low-Average | High (complex aggregations) |

|  |  |  |
| --- | --- | --- |
| Write Operation | | |
| Query | Frequency | Cost |
| Insert a Book | Low-Average  275232/365 = 754 books every day | Average (create new document) |
| Insert a User | Low  43450/365 = 119 users every day | Low (create new document) |
| Insert an Author | Low  96008/365 = 263 authors every day | Low (create new document) |
| Insert a Review | High  1466018/365 = 4017 reviews every day | Low (add new document) |
| Modify a Review | Low | Low (modify a document) |
| Delete Review | Low | Low (delete a document) |
| Add like to Review | Low-Average | Low (modify 2 documents) |
| Remove like from Review | Very Low | Low (modify 2 documents) |
| Delete Book | Very Low | Low (delete a document) |
| Delete User | Low | High (delete a document and related document reviews) |
| Delete Author | Very Low | Very High (delete a document and related books with its reviews) |
| Report a Review | Very Low | Low (add new document) |
| Report a Book | Very Low | Low (add new document) |
| Delete reported Review/Book | Average | Low-Average  (2 documents deleted and 1 added) |
| Unreport Review/Book | Average | Low  (1 documents deleted and 1 added) |
| Restore Log | Very Low | Low (1 document deleted and 1 added) |
| Insert an Admin | Very Low | Low (add new document) |

## Neo4J

|  |  |  |
| --- | --- | --- |
| Read Operation | | |
| Query | Frequency | Cost |
| Browse Follower | Average-High  (every time you visit a User/Author profile) | Average  (multiple reads) |
| Browse Following | Average-High  (every time you visit a User/Author profile) | Average  (multiple reads) |
| Browse Published Books | Low-Average  (every time you want to see the published list) | Average  (multiple reads) |
| Browse Read Books | Low-Average  (every time you want to see the read list) | Average  (multiple reads) |
| Browse To Read Books | Low-Average  (every time you want to see the to-read list) | Average  (multiple reads) |
| View Suggested User | High  (each time you visit a User/Author homepage) | High  (multiple reads) |
| View Suggested Author | High  (each time you visit a User/Author/Book page) | High  (multiple reads) |
| View Suggested Books | High  (each time you visit a Book page) | High  (multiple reads) |
| Browse Most Popular/ Desired Books | Low  (every time you want to see ranking statistics) | High  (multiple reads) |

|  |  |  |
| --- | --- | --- |
| Write Operation | | |
| Query | Frequency | Cost |
| Insert a Book | Low-Average | Low-Average  (insert 1 node and 1 ore more WROTE relationship) |
| Insert a User | Low | Low  (insert 1 node) |
| Insert an Author | Low | Low  (insert 1 node) |
| Delete Book | Very Low | Low  (delete 1 node) |
| Delete User | Low | Low  (delete 1 node) |
| Delete Author | Very Low | Low  (delete 1 node and detach it’s related books) |
| Add Follow | Average-High | Low  (add new relationship) |
| Remove Follow | Very Low | Low  (delete relationship) |
| Add to read/read list | Average | Low  (add new relationship) |
| Remove from to read/read list | Very Low | Low  (delete relationship) |
| Restore Book | Very Low | Low-Average  (insert 1 node and 1 ore more WROTE relationship) |

# Data Model

This chapter discusses organizing databases. MongoDB and Neo4j were used for the design.

## MongoDB – Document Organization

In mongoDB we are stored five different collections:

* **Authors Collection: ~** 59.6k
* **Users Collection: ~** 151.0k
* **Books Collection: ~** 131.1k
* **Genres Collection:** 38
* **Report Collection**
* **Log Collection**

### Authors Collection



It contains the fundamental information of each author.

### Users Collection



It contains the fundamental information of each user.

### Books Collection



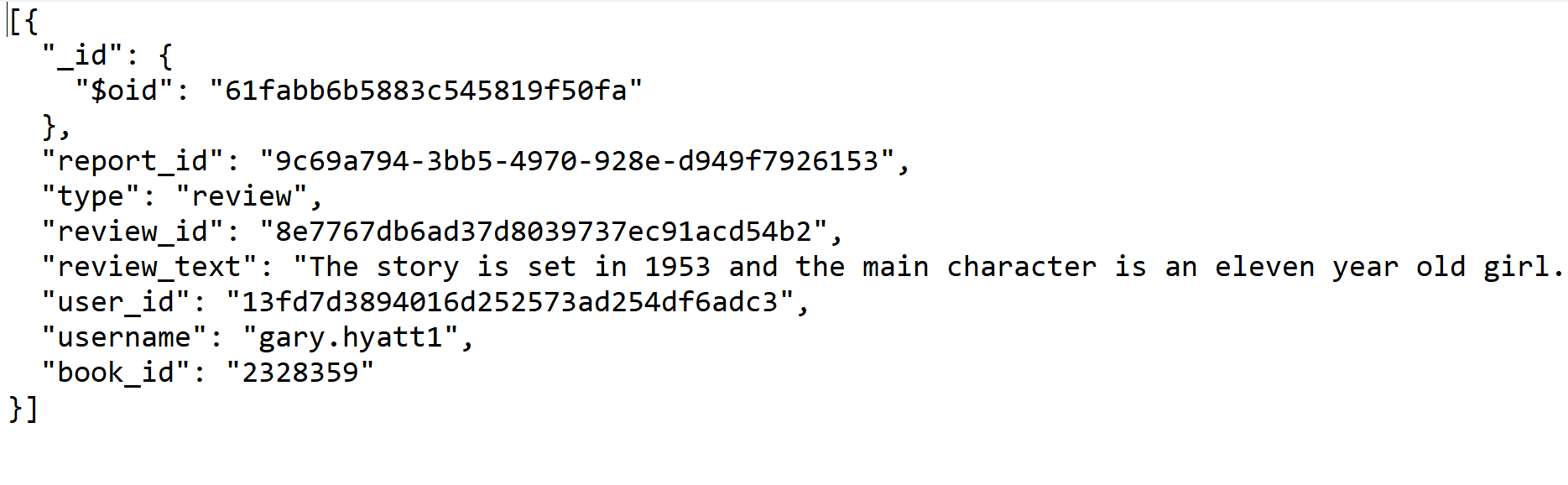
Contains the basic information of each book where some fields can be omitted. The book id, the title and at least one author are certainly required.

### Genres Collection

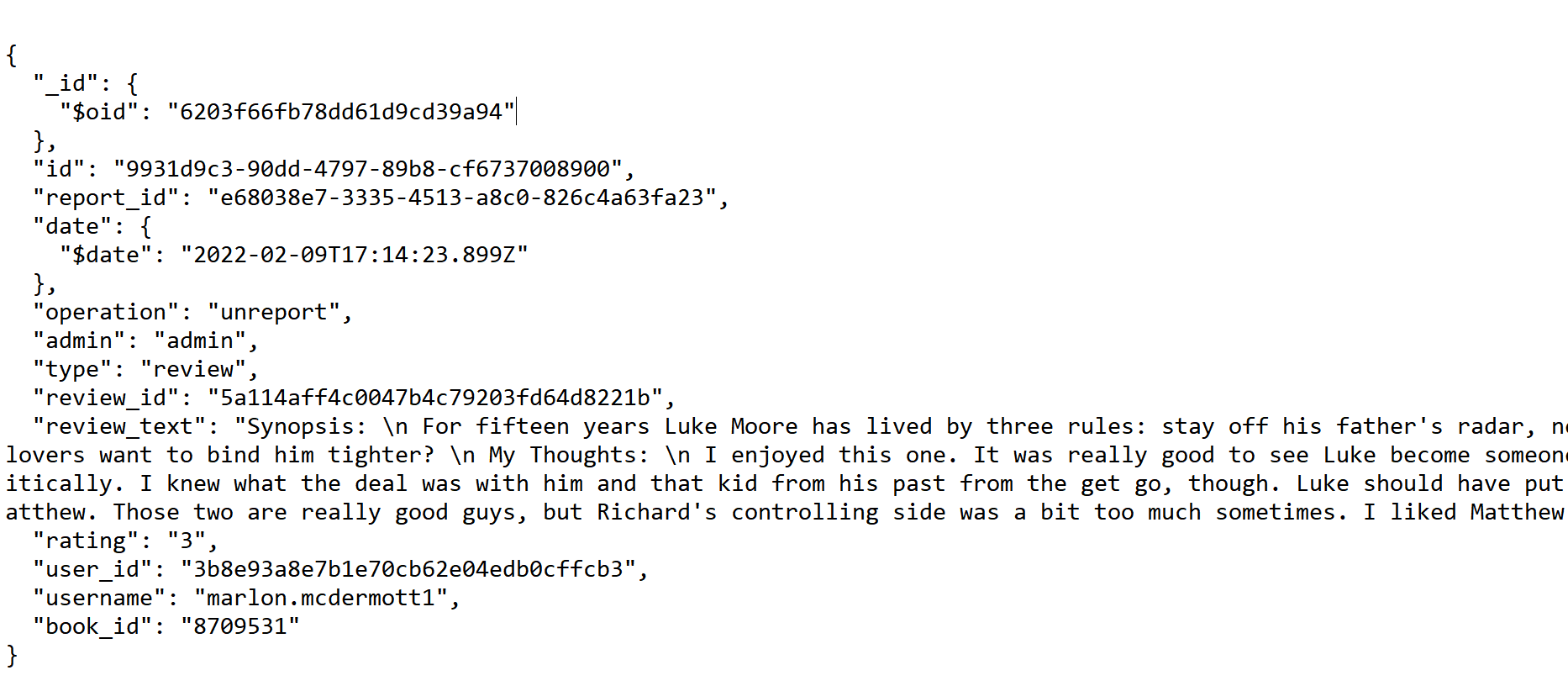


It holds all possible genres that can be used for books. It is the only static connection, as it cannot be modified by the application.

### Report Collection

It includes all the reports made by users that will be viewed by the admin

### Log Collection





It allows admins to retrieve reviews and books that they or other admins have deleted.

# Neo4J – Nodes Organization

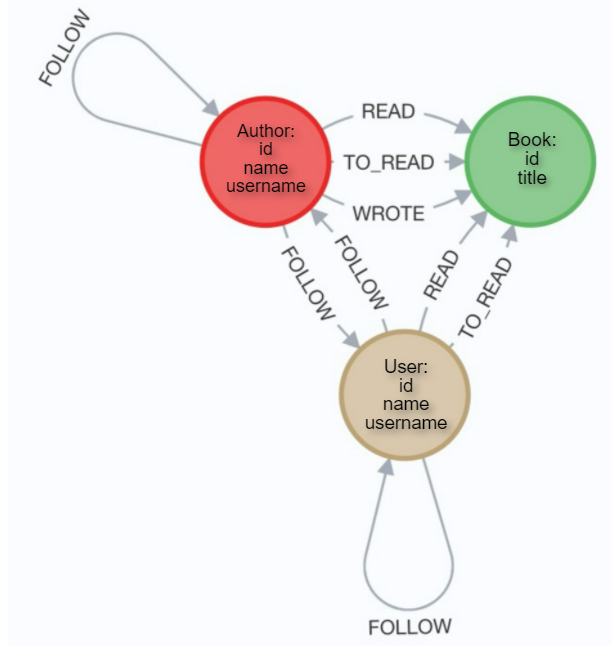
In neo4j there are ~ 340.000 nodes, ~ 240.000 relationships.

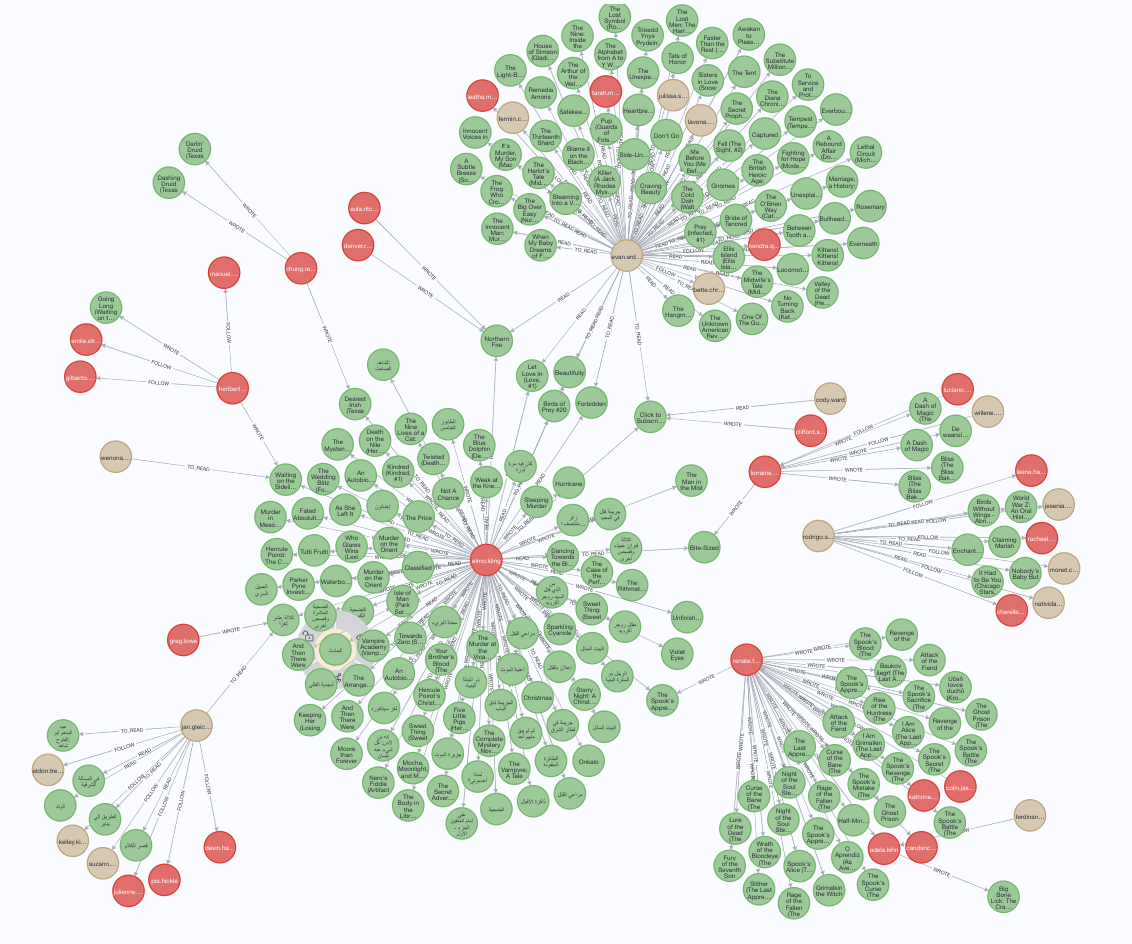
The structure of the nodes with their relations is described below:

* The "**User**" nodes represent the registered user and contain the user's id, username and name
* The "**Author**" nodes represent the registered user and contain the author’s id, username and name
* The "**Book**" nodes represent the registered user and contain the book’s id and title

Relationships:

* **User →: FOLLOW → User**, a User following a User who can be added and removed from the application. This relationship has no attributes.
* **User →: FOLLOW → Author**, a User following an Author who can be added and removed from the application. This relationship has no attributes.
* **Author→: FOLLOW → User**, an Author following a User who can be added and removed from the application. This relationship has no attributes.
* **Author→: FOLLOW → Author**, an Author following an Author who can be added and removed from the application. This relationship has no attributes.
* **User->: TO\_READ -> Book**, represents a User who wants to read a book.
* **Author->: TO\_READ -> Book**, represents an Author who wants to read a book.
* **User->: READ -> Book**, represents a User who has read wants to read a book.
* **Author->: READ -> Book**, represents an Author who has read wants to read a book.
* **Author ->: WROTE ->Book,** represents an author who has written a book.





# Implementation

The Reviook application is composed by four main package that guarantee order and code organization.

* **Package Manager:** this package is in charge of managing the main crud operations to communicate with the database.
* **Package Controllers:** this package is in charge of providing the methods that manage graphical interfaces and internal navigation of the application.
* **Package Components:** this is the package used for building custom elements needed for the GUI.
* **Package Entity:** this package contains all objects used to structure the application.

In detail:

***UserManager*** contains all the java methods used in the User or Author experience, this includes the crud operations to interact with other users.

***BookManager*** contains all the java methods used in the exploration of the collection of books.

***SearchManager*** contains specific methods used in the browsing experience from the Reviook search GUI.

***AdminManager*** contains specific methods used by admins to moderate the application.

***AuthorInterfaceController, UserInterfaceController and AdminController*** contains methods that manage the interaction between user and the GUI concerning authors, users and admin.

***BookDetailController, AddBookController, DialogNewReviewController and PreviewReviewController*** contains methods that are used to handle everything related to books in the GUI.

***LoginController, UpdateController, RegisterController and AddAdminController*** contains methods that are used in the user experience to handle operations related to sign-in and sign-up.

***SearchInterfaceController*** contains methods used in the GUI to search users, authors or books.

# Cache

In this application we immediately faced the problem of the slowness of crud operations. To solve this problem and make navigation in the application faster, we have decided to implement a sort of cache for our data.

In this way the logged user will not have to wait every time for the data to be loaded from the database because if the information is cached, the display will be instantaneous.

With this feature we solved the slow response time in Reviook, but this led to inconsistency problems since Reviook is a multi-user application.

To avoid this issue, we use a cache object into each session, this object contains the usual data with an added timestamp that will invalidate the cache if too old.

# JAVA Entity

|  |  |
| --- | --- |
| Class | Description |
| User | User registered in the application |
| Author | Author registered in the application |
| Book | A Book added in the application |
| Review | A review added in the application |
| Log | Entity used to keep track of admin operation |
| Report | Entity used to allow to report books or reviews |
| Cache | Entity used to speed up the application |

public class User {  
 private String id;  
 private String name;  
 private String nickname;  
 private String email;  
 private String password;  
 private Interaction interactions;  
 private ArrayList<String> listReviewID;  
 private Integer follower\_count;  
 private ArrayList<Genre> statistics;  
 private ListBooks listBooks;

/\* Constructors, Getters and Setters \*/

}

public class Author extends User {  
 private ArrayList<Book> published;

/\* Constructors, Getters and Setters \*/

}

public class Book {  
 private String isbn;  
 private String language\_code;  
 private String asin;  
 private Double average\_rating;  
 private String description;  
 private Integer num\_pages;  
 private Integer publication\_day;  
 private Integer publication\_month;  
 private Integer publication\_year;  
 private String image\_url;  
 private String book\_id;  
 private Integer ratings\_count;  
 private String title;  
 private ArrayList<Author> authors;  
 private ArrayList<String> genres;  
 private ArrayList<Review> reviews;

/\* Constructors, Getters and Setters \*/

}

public class Review {  
 private String review\_id;  
 private String date\_update;  
 private String user\_id;  
 private String username;  
 private String rating;  
 private String review\_text;  
 private Integer likes;  
 private Boolean liked;

/\* Constructors, Getters and Setters \*/

}

public class Log extends Report {  
 private String id;  
 private String operation;  
 private String admin;  
 private Date date;

/\* Constructors, Getters and Setters \*/

}

public class Report {  
 private String report\_id;  
 private String type;  
 private String isbn;  
 private String asin;  
 private String book\_id;  
 private String title;  
 private String description;  
 private Integer num\_pages;  
 private Integer publication\_day;  
 private Integer publication\_month;  
 private Integer publication\_year;  
 private String image\_url;  
 private ArrayList<Author> authors;  
 private ArrayList<String> genres;  
 private String review\_id;  
 private String review\_text;  
 private String rating;  
 private String user\_id;  
 private String username;

/\* Constructors, Getters and Setters \*/

}

public class Cache {  
 private ArrayList<Book> searchedBooks;  
 private ArrayList<User> searchedUsers;  
 private ArrayList<Author> searchedAuthors;  
 private String searchedTitle;  
 private String searchedGenres;  
 private String searchType;  
 private Date lastUpdate;  
 private int count;

/\* Constructors, Getters and Setters \*/

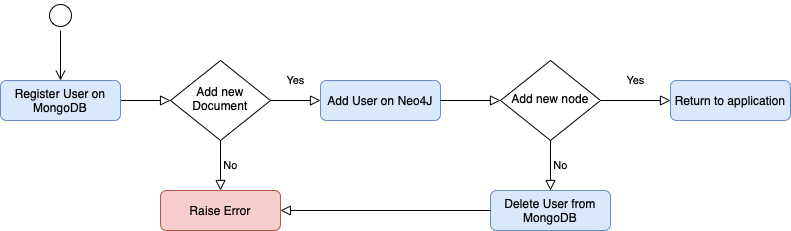
}

# Database Consistency Management

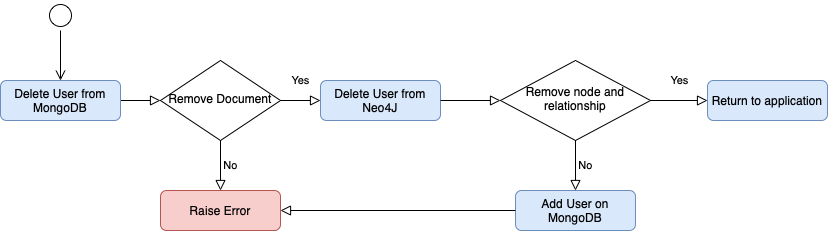
Some of the information contained into the two databases are duplicated. Some of the CRUD operations have been organized as follows, to maintain a state of consistency between the documents in MongoDB and the nodes/relationships in Neo4J.

The first write is committed on MongoDB, if it fails an exception is raised, otherwise the same operation is done on Neo4J. If the result of this last operation is positive the CRUD operation returns the control to the main program, otherwise it will try to undo the first operation on MongoDB.

* Add new User/Author



* Delete User/Author

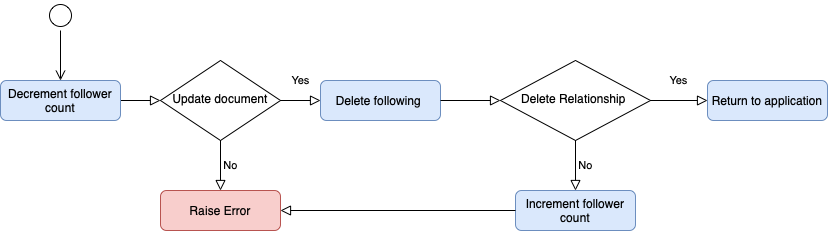


* Add Following

Immagine che contiene testo

Descrizione generata automaticamente

* Delete Following

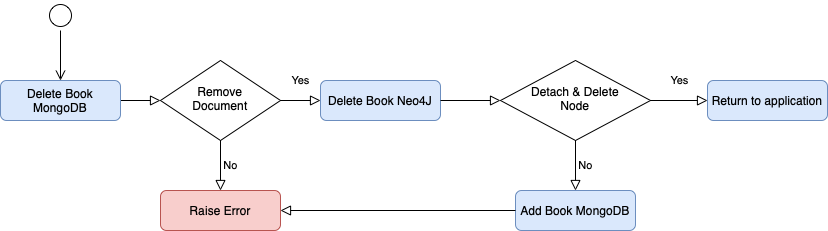


* Add new Book

Immagine che contiene testo

Descrizione generata automaticamente

* Delete Book



# Database Organization on Machines

We set up a MongoDB cluster to avoid the problem of the single point of failure and to improve the availability of our service. The cluster is composed by three replicas set hosted in different servers. In each of them, the mongod daemon runs at the same port, the primary server receives the client requests and manage communication into the cluster, on the other side the secondary servers maintain the replica updated to be ready to substitute the primary when it will crash.

As follows the organization of the virtual machines

|  |  |  |
| --- | --- | --- |
| VM | IP Address & Port | OS |
| Replica-0 | 172.16.4.102:27020 | Ubuntu 18.04.2 LTS |
| Replica-1 | 172.16.4.103:27020 | Ubuntu 18.04.2 LTS |
| Replica-2 | 172.16.4.104:27020 | Ubuntu 18.04.2 LTS |

## Replica Configuration

This is our replica configuration performed on one of the three Servers

rsconf1 = {\_id: "reviook",

members: [

{\_id: 0, host: "172.16.4.102:27020", priority: 1

},

{\_id: 1, host: "172.16.4.103:27020", priority: 2

},

{\_id: 2, host: "172.16.4.104:27020", priority: 5

}

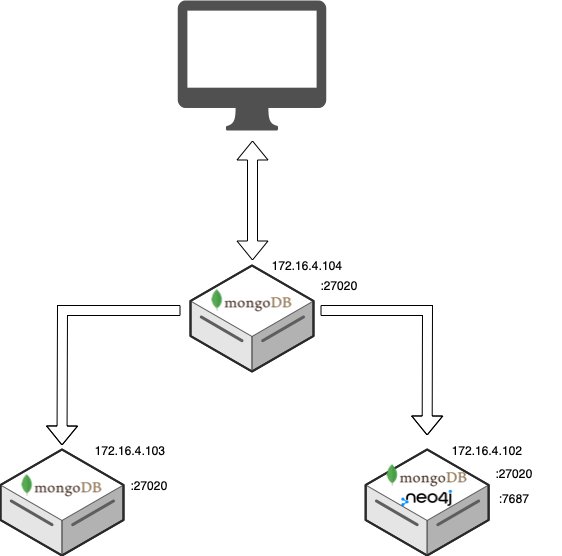
]

};

The replica-2 has the highest priority, so it will be the primary. Our application is mainly read oriented as we discover in the query analysis section, so we decided to make some other configurations on the application sides to speed up writes and reads to the detriment of the consistency. For instance, we preferred to read from the nearest replica (the one which has the lowest latency) and wait for the writing acknowledgement from only one replica.

uri = new ConnectionString("mongodb://172.16.4.102:27020,172.16.4.103:27020,172.16.4.104:27020/");  
MongoClientSettings msc = MongoClientSettings.*builder*()  
 .applyConnectionString(uri)  
 .readPreference(ReadPreference.*nearest*())  
 .retryWrites(true)  
 .writeConcern(WriteConcern.*W1*).build();

## Replica crash



If the primary crashes one of the secondaries will be elected as the new primary. We assigned priority on each replica to control the behavior of the election algorithm. For this reason, we decided to set up our Neo4J server on the Replica-0 that has the lowest priority to become the new primary, in this way we should have a better load balance on each machine.

At the start we have this configuration:

|  |  |  |
| --- | --- | --- |
| Primary | Secondary | Secondary |
| 172.16.4.104:27020 | 172.16.4.103:27020 | 172.16.4.102:27020 |

If the Primary crashes, we will have the following new configuration:

|  |  |  |
| --- | --- | --- |
| - | Primary | Secondary |
| 172.16.4.104:27020 | 172.16.4.103:27020 | 172.16.4.102:27020 |

# Neo4J CRUD Operations

## Create

* Add new user or author

public boolean addNewUsers(User user, String type) {  
 boolean result;  
 try (Session session = nd.getDriver().session()) {  
 result = session.writeTransaction((TransactionWork<Boolean>) tx -> {  
 tx.run("CREATE (ee:" + type + " { id: $id, name: $name, username: $username})", *parameters*("id", user.getId(), "name", user.getName(), "username", user.getNickname()));  
 return true;  
 });  
 }  
 return result;  
}

* Create follow relationship from username1 to username2

public boolean following(String username1, String type1, String username2, String type2) {  
 boolean result = false;  
 if (incrementFollowerCount(username2)) {  
 try (Session session = nd.getDriver().session()) {  
 result = session.writeTransaction((TransactionWork<Boolean>) tx -> {  
 tx.run("MATCH (n:" + type1 + "),(nn:" + type2 + ") WHERE n.username ='"

+ username1 + "' AND nn.username='"

+ username2 + "'" + "CREATE (n)-[:FOLLOW]->(nn)");  
 return true;  
 });  
 }  
 if (result != true)  
 decrementFollowerCount(username2);  
 }  
 return result;  
}

* Create if not exists to\_read relationship from username to book\_id

public boolean toReadAdd(String type, String username, String book\_id) {  
 boolean result = false;  
 try (Session session = nd.getDriver().session()) {  
 result = session.writeTransaction((TransactionWork<Boolean>) tx -> {  
 tx.run("MATCH (n:" + type + "),(nn:Book) WHERE n.username ='"

+ username + "' AND nn.id='" + book\_id + "'" +  
 "MERGE (n)-[:TO\_READ]->(nn)");  
 return true;  
 });  
 }  
 return result;  
}

* Create if not exists read relationship from username to book\_id

public boolean readAdd(String type, String username, String book\_id) {  
 boolean result = false;  
 try (Session session = nd.getDriver().session()) {  
 result = session.writeTransaction((TransactionWork<Boolean>) tx -> {  
 tx.run("MATCH (n:" + type + "),(nn:Book) WHERE n.username ='"

+ username + "' AND nn.id='" + book\_id + "'" +  
 "MERGE (n)-[:READ]->(nn)");  
 return true;  
 });  
 }  
 return result;  
}

* Create Book

public boolean addBookN4J(Book newBook) {  
 Boolean result;  
 try (Session session = *nd*.getDriver().session()) {  
 result = session.writeTransaction((TransactionWork<Boolean>) tx -> {  
 tx.run("CREATE (ee: Book { id : $id, title: $title})", *parameters*("id", newBook.getBook\_id(), "title", newBook.getTitle()));  
 for (int i = 0; i < newBook.getAuthors().size(); i++) {  
 tx.run("MATCH (dd:Author),(ee: Book) WHERE dd.id = '" + newBook.getAuthors().get(i).getId() + "' AND ee.id='" + newBook.getBook\_id() + "'" + "CREATE (dd)-[:WROTE]->(ee)");  
 }  
 return true;  
 });  
 }  
 return result;  
}

## Read

* Read read/to\_read/wrote relationship from username to book\_id

public ArrayList<Book> loadRelationsBook(String type, String username, String read) {  
 ArrayList<Book> readings = new ArrayList<Book>();  
 ArrayList<Book> books = new ArrayList<>();  
 try (Session session = nd.getDriver().session()) {  
 readings = session.readTransaction((TransactionWork<ArrayList<Book>>) tx -> {  
 Result result = tx.run("MATCH (ee:" + type + ")-[:" + read + "]->(book) where ee.username = '" + username + "' " +  
 "return book.title, book.id");  
 while (result.hasNext()) {  
 Record r = result.next();  
 books.add(new Book(((Record) r).get("book.title").asString(), ((Record) r).get("book.id").asString()));  
 }  
 return books;  
 });  
 }  
 return readings;  
}

* Read following relationship from username  
  public List<String> loadRelationsFollowing(String type, String username) {  
   List<String> relationship = new ArrayList<>();  
   try (Session session = nd.getDriver().session()) {  
   relationship = session.readTransaction((TransactionWork<List<String>>) tx -> {  
   Result result = tx.run("MATCH (ee:" + type + ")-[:FOLLOW]->(friends) where ee.username = '" + username + "' " +  
   "return friends.username as Friends");  
   ArrayList<String> following = new ArrayList<>();  
   while (result.hasNext()) {  
   Record r = result.next();  
   following.add(((Record) r).get("Friends").asString());  
   }  
   return following;  
   });  
   }  
   return relationship;  
  }
* Read follower relationship from username  
  public List<String> loadRelationsFollower(String type, String username) {  
   List<String> relationship = new ArrayList<>();  
   try (Session session = nd.getDriver().session()) {  
   relationship = session.readTransaction((TransactionWork<List<String>>) tx -> {  
   Result result = tx.run("MATCH (ee:" + type + ")<-[:FOLLOW]-(friends) where ee.username = '" + username + "' " +  
   "return friends.username as Friends");  
   ArrayList<String> follower = new ArrayList<>();  
   while (result.hasNext()) {  
   Record r = result.next();  
   follower.add(((Record) r).get("Friends").asString());  
   }  
   return follower;  
   });  
   }  
   return relationship;  
  }

## Delete

* Delete user or author

public boolean deleteUserN4J(User user, String type) {  
 boolean result;  
 String t = type.equals("author") ? "Author" : "User";  
 try (Session session = nd.getDriver().session()) {  
 result = session.writeTransaction((TransactionWork<Boolean>) tx -> {  
 tx.run("MATCH (n : " + t + " { username: '"

+ user.getNickname() + "'}) DETACH DELETE n");  
 return true;  
 });  
 }  
 return result;  
}

* Delete follow relationship from username1 to username2  
  public boolean deleteFollowing(String username1, String type1, String username2, String type2) {  
   boolean result = false;  
   if (decrementFollowerCount(username2)) {  
   try (Session session = nd.getDriver().session()) {  
   result = session.writeTransaction((TransactionWork<Boolean>) tx -> {  
   tx.run("MATCH (n:" + type1 + " { username: '"

+ username1 + "' })-[r:FOLLOW]-> " +  
 "(c :" + type2 + " { username: '"

+ username2 + "' })" +  
 "DELETE r");  
 return true;  
 });  
 }  
 if (result != true)  
 incrementFollowerCount(username2);  
 }  
 return result;  
}

* Delete to\_read/read relationship from user/author

public boolean removeBookFromList(String idBook, String Relation, String username, String Type) {  
 try (Session session = *nd*.getDriver().session()) {  
 session.writeTransaction((TransactionWork<Void>) tx -> {  
 tx.run("MATCH (n:" + Type + "{username: '"

+ username + "' })-[r:" + Relation + "]->" +  
 "(c : Book{id: '" + idBook + "'}) " +  
 "DELETE r");  
 return null;  
 });  
 } catch (Exception e) {  
 e.printStackTrace();  
 return false;  
 }  
 return true;  
}

* Delete book

public boolean deleteBookN4J(Book book) {  
 boolean result;  
 try (Session session = *nd*.getDriver().session()) {  
 result = session.writeTransaction((TransactionWork<Boolean>) tx -> {  
 tx.run("MATCH (n : Book { id: '" + book.getBook\_id() + "'}) DETACH DELETE n");  
 return true;  
 });  
 }  
 return result;  
}

# Crud operations MongoDB

Here we can find listed the crud operations performed by the application in the MongoDB.

* Create operations that allow us to manage books, author, users, reviews, logs and reports insertions:

public boolean addBookMongo(Book newBook) {   
    InsertOneResult result = null;   
    Calendar calendar = Calendar.*getInstance*();   
    calendar.setTime(new Date());   
   
    ArrayList<DBObject> authorsObj = new ArrayList<>();   
    for (Author a : newBook.getAuthors()) {   
        DBObject author = new BasicDBObject();   
        author.put("author\_name", (String) a.getName());   
        author.put("author\_role", ""); *// TODO da togliere*   
author.put("author\_id", (String) a.getId());   
        authorsObj.add(author);   
    }   
   
ArrayList<Review> reviews = new ArrayList<Review>();   
    Document doc = new Document("language\_code", newBook.getLanguage\_code())   
            .append("isbn", newBook.getIsbn())   
            .append("description", newBook.getDescription())   
            .append("num\_pages", newBook.getNum\_pages())   
            .append("publication\_day", newBook.getPublication\_day())   
            .append("publication\_month", newBook.getPublication\_month())   
            .append("publication\_year", newBook.getPublication\_year())   
            .append("image\_url", newBook.getImage\_url())   
            .append("book\_id", newBook.getBook\_id())   
            .append("title", newBook.getTitle())   
            .append("average\_rating", 0.0)   
            .append("ratings\_count", 0)   
            .append("genres", newBook.getGenres())   
            .append("authors", authorsObj)   
            .append("reviews", newBook.getReviews());   
    try {   
        result = *md*.getCollection(*bookCollection*).insertOne(doc);   
    } catch (Exception e) {   
        e.printStackTrace();   
    }   
    if (result != null)   
        return result.wasAcknowledged();   
    return false;   
} 

public boolean addReviewToBook(String reviewText, Integer ratingBook, String book\_id) {   
    MongoCollection<Document> book = *md*.getCollection(*bookCollection*);   
    Document newReview = new Document();   
    String reviewID = UUID.*randomUUID*().toString();   
    UpdateResult addReview, rateUpdated;   
    LocalDateTime now = LocalDateTime.*now*();   
    Date date = Date.*from*(now.atZone(ZoneId.*systemDefault*()).toInstant());   
    newReview.append("date\_updated", date);   
    newReview.append("review\_id", reviewID);   
    newReview.append("likes", 0);   
    newReview.append("rating", ratingBook);   
    newReview.append("review\_text", reviewText);   
    if (session.getLoggedUser() != null) {   
        newReview.append("user\_id", session.getLoggedUser().getId());   
        newReview.append("username", session.getLoggedUser().getNickname());   
    } else {   
        newReview.append("user\_id", session.getLoggedAuthor().getId());   
        newReview.append("username", session.getLoggedAuthor().getNickname());   
    }   
    Bson getBook = *eq*("book\_id", book\_id);   
    DBObject elem = new BasicDBObject("reviews", new BasicDBObject(newReview));   
    DBObject insertReview = new BasicDBObject("$push", elem);   
   
    try {   
        addReview = book.updateOne(getBook, (Bson) insertReview);   
   
        if (addReview.getModifiedCount() == 1) {   
            Book bookToUpdate = getBookByID(book\_id);   
            Double newRating = updateRating(bookToUpdate.getReviews());   
            rateUpdated = book.updateOne(getBook, Updates.*set*("average\_rating", newRating));   
            if (rateUpdated.getModifiedCount() == 1) {   
                if (session.getLoggedUser() != null)   
                    userManager.readAdd("User", session.getLoggedUser().getNickname(), book\_id);   
                else   
                    userManager.readAdd("Author", session.getLoggedAuthor().getNickname(), book\_id);   
                return true;   
            }   
        }   
    }catch (Exception e){   
        e.printStackTrace();   
    }   
    return false;   
}

public boolean register(User user, String type) {   
    ArrayList<String> liked\_review = new ArrayList<>();   
    InsertOneResult result = null;   
    try {   
        Document doc = new Document("name", user.getName())   
                .append("password", user.getPassword())   
                .append("follower\_count", 0)   
                .append("liked\_review", liked\_review)   
                .append("email", user.getEmail())   
                .append("username", user.getNickname());   
        if (type.equals("Author")) {   
            doc.append("author\_id", user.getId());   
            result = md.getCollection(*authorCollection*).insertOne(doc);   
        } else {   
            doc.append("user\_id", user.getId());   
            result = md.getCollection(*usersCollection*).insertOne(doc);   
        }   
    } catch (Exception e) {   
        e.printStackTrace();   
    }   
    if (result != null)   
        return result.wasAcknowledged();   
    return false;   
}

public boolean addLog(Report report, String operation) {   
    MongoCollection<Document> logs = md.getCollection("logs");   
    LocalDateTime now = LocalDateTime.*now*();   
    Date date = Date.*from*(now.atZone(ZoneId.*systemDefault*()).toInstant());   
    if (report.getType().equals("book")) {   
        ArrayList<DBObject> authorsList = new ArrayList<DBObject>();   
        for (Author a : report.getAuthors()) {   
            DBObject author = new BasicDBObject();   
            author.put("author\_name", (String) a.getName());   
            author.put("author\_username", (String) a.getNickname());   
            author.put("author\_id", (String) a.getId());   
            authorsList.add(author);   
        }   
        Document newLog = new Document("id", UUID.*randomUUID*().toString())   
                .append("report\_id", report.getReport\_id())   
                .append("date", date)   
                .append("operation", operation)   
                .append("admin", session.getAdmin())   
                .append("type", report.getType())   
                .append("isbn", report.getIsbn() != null ? report.getIsbn() : "")   
                .append("asin", report.getAsin() != null ? report.getAsin() : "")   
                .append("num\_pages", report.getNum\_pages())   
                .append("publication\_day", report.getPublication\_day())   
                .append("publication\_month", report.getPublication\_month())   
                .append("publication\_year", report.getPublication\_year())   
                .append("image\_url", report.getImage\_url())   
                .append("book\_id", report.getBook\_id())   
                .append("title", report.getTitle())   
                .append("description", report.getDescription())   
                .append("authors", authorsList)   
                .append("genres", report.getGenres());   
        InsertOneResult res = logs.insertOne(newLog);   
        if (!res.wasAcknowledged()) {   
            return false;   
        }   
    } else if (report.getType().equals("review")) {   
        Document newLog = new Document("id", UUID.*randomUUID*().toString())   
                .append("report\_id", report.getReport\_id())   
                .append("date", date)   
                .append("operation", operation)   
                .append("admin", session.getAdmin())   
                .append("type", report.getType())   
                .append("review\_id", report.getReview\_id())   
                .append("review\_text", report.getReview\_text())   
                .append("rating", report.getRating())   
                .append("user\_id", report.getUser\_id())   
                .append("username", report.getUsername())   
                .append("book\_id", report.getBook\_id());   
        InsertOneResult res = logs.insertOne(newLog);   
        if (!res.wasAcknowledged()) {   
            return false;   
        }   
    }   
    return true;   
}   
   
public boolean addNewAdmin(String username, String password) {   
    InsertOneResult result = null;   
    try {   
        Document doc = new Document("username", username).append("password", password);   
        result = md.getCollection(*adminCollection*).insertOne(doc);   
    } catch (Exception e) {   
        e.printStackTrace();   
    }   
    if (result != null)   
        return result.wasAcknowledged();   
    return false;   
}   
   
public boolean reportBook(Book book) {   
    MongoCollection<Document> reports = md.getCollection("reports");   
    InsertOneResult result = null;   
    try (MongoCursor<Document> cursor = reports.find(*and*(*eq*("book\_id", book.getBook\_id()), *eq*("type", "book"))).iterator()) {   
        if (!cursor.hasNext()) {   
            ArrayList<DBObject> authorsList = new ArrayList<DBObject>();   
            for (Author a : book.getAuthors()) {   
                DBObject author = new BasicDBObject();   
                author.put("author\_name", (String) a.getName());   
                author.put("author\_username", (String) a.getNickname());   
                author.put("author\_id", (String) a.getId());   
                authorsList.add(author);   
            }   
            Document newBook = new Document();   
            newBook.append("report\_id", UUID.*randomUUID*().toString());   
            newBook.append("type", "book");   
            newBook.append("isbn", book.getIsbn() != null ? book.getIsbn() : "");   
            newBook.append("asin", book.getAsin() != null ? book.getAsin() : "");   
            newBook.append("book\_id", book.getBook\_id());   
            newBook.append("title", book.getTitle());   
            newBook.append("description", book.getDescription());   
            newBook.append("num\_pages", book.getNum\_pages());   
            newBook.append("publication\_day", book.getPublication\_day());   
            newBook.append("publication\_month", book.getPublication\_month());   
            newBook.append("publication\_year", book.getPublication\_year());   
            newBook.append("image\_url", book.getImage\_url());   
            newBook.append("genres", book.getGenres());   
            newBook.append("authors", authorsList);   
            result = reports.insertOne(newBook);   
        }   
    } catch (Exception e) {   
        e.printStackTrace();   
    }   
    if (result != null)   
        return result.wasAcknowledged();   
    return false;   
}   
   
public boolean reportReview(Review review, String book\_id) {   
    MongoCollection<Document> reports = md.getCollection("reports");   
    InsertOneResult result = null;   
    try (MongoCursor<Document> cursor = reports.find(*and*(*eq*("review\_id", review.getReview\_id()), *eq*("type", "review"))).iterator()) {   
        if (!cursor.hasNext()) {   
            Document newReview = new Document();   
            newReview.append("report\_id", UUID.*randomUUID*().toString());   
            newReview.append("type", "review");   
            newReview.append("review\_id", review.getReview\_id());   
            newReview.append("review\_text", review.getReview\_text());   
            newReview.append("rating", review.getRating());   
            newReview.append("user\_id", review.getUser\_id());   
            newReview.append("username", review.getUsername());   
            newReview.append("book\_id", book\_id);   
            result = reports.insertOne(newReview);   
        }   
    } catch (Exception e) {   
        e.printStackTrace();   
    }   
    if (result != null)   
        return result.wasAcknowledged();   
    return false;   
}

* Read operations that allow us to get books, author, users, reviews, logs and view them:

public Book getBookByID(String book\_id) {   
    MongoCollection<Document> books = *md*.getCollection(*bookCollection*);   
    Document book = new Document();   
    try (MongoCursor<Document> cursor = books.find(*eq*("book\_id", book\_id)).iterator()) {   
        if (!cursor.hasNext()) {   
            return null;   
        }   
        book = cursor.next();   
    } catch (Exception e) {   
        e.printStackTrace();   
    }   
    ArrayList<Author> authorsLis = new ArrayList<>();   
    ArrayList<Review> reviewsList = new ArrayList<>();   
    ArrayList<Document> authors = (ArrayList<Document>) book.get("authors");   
    ArrayList<Document> reviews = (ArrayList<Document>) book.get("reviews");   
    ArrayList<String> genres = (ArrayList<String>) book.get("genres");   
   
    for (Document r : reviews) {   
        String date;   
        if(r.get("date\_updated") == null) {   
            if (r.get("date\_added") == null) {   
                date = null;   
            }else{   
                date = r.get("date\_added").toString();   
            }   
        }else {   
            date = r.get("date\_updated").toString();   
        }   
        reviewsList.add(new Review(   
                r.getString("username"),   
                r.getString("review\_id"),   
                date,   
                r.get("likes") == null ? r.getInteger("helpful") : r.getInteger("likes"),   
                r.getString("user\_id"),   
                r.get("rating").toString(),   
                r.getString("review\_text")   
        ));   
    }   
    for (Document a : authors) {   
        Author author = new Author(   
                a.getString("author\_id"),   
                a.getString("author\_name"),   
                "",   
                "",   
                "",   
                null,   
                0   
        );   
        authorsLis.add(author);   
    }   
   
    Book outputBook = new Book(   
            book.get("isbn") == null ? null : book.getString("isbn"),   
            book.get("language\_code") == null ? null : book.getString("language\_code"),   
            book.get("asin") == null ? null : book.getString("asin"),   
            book.get("average\_rating").toString().equals("") ? Double.*valueOf*(0) : Double.*valueOf*(book.get("average\_rating").toString()),   
            book.get("description") == null ? null : book.getString("description"),   
            book.get("num\_pages") == null ? null : book.getInteger("num\_pages"),   
            book.get("publication\_day") == null ? null : book.getInteger("publication\_day"),   
            book.get("publication\_month") == null ? null : book.getInteger("publication\_month"),   
            book.get("publication\_year") == null ? null : book.getInteger("publication\_year"),   
            book.get("image\_url") == null ? null : book.getString("image\_url"),   
            book.getString("book\_id"),   
            book.getInteger("ratings\_count"),   
            book.getString("title"),   
            authorsLis,   
            genres,   
            reviewsList   
    );   
    return outputBook;   
}

public ArrayList<Book> searchBooks(String searchField, String genre) {   
    MongoCollection<Document> books = md.getCollection(*bookCollection*);   
    MongoCursor<Document> cursor = null;   
    ArrayList<Book> result = new ArrayList<>();   
   
    boolean titleSearch = true;   
    boolean genresSearch = true;   
   
    if (searchField == null || searchField.equals(""))   
        titleSearch = false;   
    if (genre == null || genre.equals(""))   
        genresSearch = false;   
   
    Bson titleFilter;   
    Bson genreFilter;   
   
    try {   
        *//global research*   
if (!titleSearch && !genresSearch)   
            cursor = books.find().iterator();   
            *//search by title*   
else if (titleSearch && !genresSearch) {   
            titleFilter = *text*("\"" + searchField + "\"", new TextSearchOptions().caseSensitive(false));   
            cursor = books.find(titleFilter).iterator();   
        }   
        *//search by genre*   
else if (!titleSearch && genresSearch) {   
            genreFilter = *in*("genres", genre);   
            cursor = books.find(genreFilter).iterator();   
        }   
        *//search by title & genre*   
else {   
            titleFilter = *match*(*text*(searchField, new TextSearchOptions().caseSensitive(false)));   
            genreFilter = *match*(*in*("genres", genre));   
            cursor = books.aggregate(Arrays.*asList*(titleFilter, genreFilter)).iterator();   
        }   
   
        while (cursor.hasNext()) {   
            Document book = cursor.next();   
            *//System.out.println("documento->" + document);*   
   
ArrayList<Author> authorsLis = new ArrayList<>();   
            ArrayList<Review> reviewsList = new ArrayList<>();   
            ArrayList<Document> authors = (ArrayList<Document>) book.get("authors");   
            ArrayList<Document> reviews = (ArrayList<Document>) book.get("reviews");   
            ArrayList<String> genres = (ArrayList<String>) book.get("genres");   
   
            for (Document r : reviews) {   
                String date;   
                if(r.get("date\_updated") == null) {   
                    if (r.get("date\_added") == null) {   
                        date = null;   
                    }else{   
                        date = r.get("date\_added").toString();   
                    }   
                }else {   
                    date = r.get("date\_updated").toString();   
                }   
                reviewsList.add(new Review(   
                        r.getString("username"),   
                        r.getString("review\_id"),   
                        date,   
                        r.get("likes") == null ? r.getInteger("helpful") : r.getInteger("likes"),   
                        r.getString("user\_id"),   
                        r.get("rating").toString(),   
                        r.getString("review\_text")   
                ));   
            }   
            for (Document a : authors) {   
                Author author = new Author(   
                        a.getString("author\_id"),   
                        a.getString("author\_name"),   
                        "",   
                        "",   
                        "",   
                        null,   
                        0   
                );   
                authorsLis.add(author);   
                *//authorsLis.add(a.getString("author\_name"));*   
}   
   
            result.add(new Book(   
                    book.get("isbn") == null ? null : book.getString("isbn"),   
                    book.get("language\_code") == null ? null : book.getString("language\_code"),   
                    book.get("asin") == null ? null : book.getString("asin"),   
                    book.get("average\_rating").toString().equals("") ? Double.*valueOf*(0) : Double.*valueOf*(book.get("average\_rating").toString()),   
                    book.get("description") == null ? null : book.getString("description"),   
                    book.get("num\_pages") == null ? null : book.getInteger("num\_pages"),   
                    book.get("publication\_day") == null ? null : book.getInteger("publication\_day"),   
                    book.get("publication\_month") == null ? null : book.getInteger("publication\_month"),   
                    book.get("publication\_year") == null ? null : book.getInteger("publication\_year"),   
                    book.get("image\_url") == null ? null : book.getString("image\_url"),   
                    book.getString("book\_id"),   
                    book.getInteger("ratings\_count"),   
                    book.getString("title"),   
                    authorsLis,   
                    genres,   
                    reviewsList   
            ));   
        }   
    } catch (Exception e) {   
        e.printStackTrace();   
    } finally {   
        if (cursor != null)   
            cursor.close();   
    }   
    return result;   
}   
   
public ArrayList<User> searchUser(String Username) {   
    MongoCollection<Document> user = md.getCollection(*usersCollection*);   
    List<Document> queryResults;   
    ArrayList<User> result = new ArrayList<>();   
    *//search on exact username*   
try {   
        if (Username.equals("")) {   
            queryResults = user.find().into(new ArrayList());   
        } else {   
            queryResults = user.find(*eq*("username", Username)).into(new ArrayList());   
        }   
   
        for (Document r : queryResults) {   
            ArrayList<String> listReviewID = (ArrayList<String>) r.get("liked\_review");   
            result.add(new User(r.getString("user\_id"), r.get("name").toString(), r.get("username").toString(), r.get("email").toString(), r.get("password").toString(), listReviewID, (Integer) r.get("follower\_count")));   
        }   
   
        *//search on name or surname*   
if (!Username.equals("")) {   
            queryResults = user.find(*text*("\"" + Username + "\"", new TextSearchOptions().caseSensitive(false))).into(new ArrayList());   
            User us;   
            for (Document r : queryResults) {   
                ArrayList<String> listReviewID = (ArrayList<String>) r.get("liked\_review");   
                us = new User(r.getString("user\_id"), r.get("name").toString(), r.get("username").toString(), r.get("email").toString(), r.get("password").toString(), listReviewID, (Integer) r.get("follower\_count"));   
                if (!result.contains(us))   
                    result.add(us);   
            }   
        }   
    } catch (Exception e) {   
        e.printStackTrace();   
    }   
    return result;   
}   
   
public ArrayList<Author> searchAuthor(String Username) {   
    MongoCollection<Document> author = md.getCollection(*authorCollection*);   
    List<Document> queryResults;   
    ArrayList<Author> result = new ArrayList<>();   
   
    *//search on exact username*   
try {   
        if (Username.equals("")) {   
            queryResults = author.find().into(new ArrayList());   
        } else {   
            queryResults = author.find(*eq*("username", Username)).into(new ArrayList());   
        }   
   
        for (Document r : queryResults) {   
            ArrayList<String> listReviewID = (ArrayList<String>) r.get("liked\_review");   
            result.add(new Author(r.getString("author\_id"), r.get("name").toString(), r.get("username").toString(), r.get("email").toString(), r.get("password").toString(), listReviewID, (Integer) r.get("follower\_count")));   
        }   
   
        if (!Username.equals("")) {   
            *//search on name or surname*   
queryResults = author.find(*text*("\"" + Username + "\"", new TextSearchOptions().caseSensitive(false))).into(new ArrayList());   
            Author auth;   
            for (Document r : queryResults) {   
                ArrayList<String> listReviewID = (ArrayList<String>) r.get("liked\_review");   
                auth = new Author(r.getString("author\_id"), r.get("name").toString(), r.get("username").toString(), r.get("email").toString(), r.get("password").toString(), listReviewID, (Integer) r.get("follower\_count"));   
                if (!result.contains(auth))   
                    result.add(auth);   
            }   
        }   
    } catch (Exception e) {   
        e.printStackTrace();   
    }   
    return result;   
}

public ArrayList<Report> loadReviewReported() {   
    ArrayList<Report> reportedReview = new ArrayList<>();   
    try {   
        MongoCollection<Document> reports = md.getCollection(*reportsCollection*);   
        List<Document> queryResults;   
        queryResults = reports.find().into(new ArrayList<>());   
        for (Document r : queryResults) {   
            if (r.getString("type").equals("review")) {   
                reportedReview.add(new Report(   
                        r.getString("report\_id"),   
                        r.getString("type"),   
                        "",   
                        "",   
                        r.getString("book\_id"),   
                        "",   
                        "",   
                        0,   
                        0,   
                        0,   
                        0,   
                        "",   
                        r.getString("review\_id"),   
                        r.getString("review\_text"),   
                        r.get("rating").toString(),   
                        r.getString("user\_id"),   
                        r.getString("username"),   
                        null,   
                        null   
                ));   
            }   
        }   
    } catch (Exception e) {   
        e.printStackTrace();   
    }   
    return reportedReview;   
}   
   
public ArrayList<Report> loadBookReported() {   
    ArrayList<Report> reportedBook = new ArrayList<>();   
    try {   
        MongoCollection<Document> reports = md.getCollection(*reportsCollection*);   
        List<Document> queryResults;   
        queryResults = reports.find().into(new ArrayList<>());   
        ArrayList<Author> authorsLis = new ArrayList<>();   
        for (Document r : queryResults) {   
            if (r.getString("type").equals("book")) {   
                ArrayList<Document> authors = (ArrayList<Document>) r.get("authors");   
                ArrayList<String> genres = (ArrayList<String>) r.get("genres");   
                for (Document a : authors) {   
                    Author author = new Author(   
                            a.getString("author\_id"),   
                            a.getString("author\_name"),   
                            a.getString("author\_username"),   
                            "",   
                            "",   
                            null,   
                            0   
                    );   
                    authorsLis.add(author);   
                }   
                reportedBook.add(new Report(   
                        r.getString("report\_id"),   
                        r.getString("type"),   
                        r.getString("isbn"),   
                        r.getString("asin"),   
                        r.getString("book\_id"),   
                        r.getString("title"),   
                        r.getString("description"),   
                        (Integer) r.get("num\_pages"),   
                        (Integer) r.get("publication\_day"),   
                        (Integer) r.get("publication\_month"),   
                        (Integer) r.get("publication\_year"),   
                        r.getString("image\_url"),   
                        "",   
                        "",   
                        "",   
                        "",   
                        "",   
                        authorsLis,   
                        genres   
                ));   
            }   
        }   
    } catch (Exception e) {   
        e.printStackTrace();   
    }   
    return reportedBook;   
}

public ArrayList<Log> loadLogs() {   
    ArrayList<Log> logsList = new ArrayList<>();   
    try {   
        MongoCollection<Document> logs = md.getCollection(*logsCollection*);   
        List<Document> results = logs.find().sort(*descending*("date")).into(new ArrayList<>());   
        for (Document l : results) {   
            if (l.getString("type").equals("review")) {   
                logsList.add(   
                        new Log(   
                                l.getString("id"),   
                                l.getDate("date"),   
                                l.getString("operation"),   
                                l.getString("admin"),   
                                l.getString("report\_id"),   
                                l.getString("type"),   
                                "",   
                                "",   
                                l.getString("book\_id"),   
                                "",   
                                "",   
                                0,   
                                0,   
                                0,   
                                0,   
                                "",   
                                l.getString("review\_id"),   
                                l.getString("review\_text"),   
                                l.getString("rating"),   
                                l.getString("user\_id"),   
                                l.getString("username"),   
                                null,   
                                null   
                        )   
                );   
            } else if (l.getString("type").equals("book")) {   
                ArrayList<Author> authorsLis = new ArrayList<>();   
                ArrayList<Document> authors = (ArrayList<Document>) l.get("authors");   
                ArrayList<String> genres = (ArrayList<String>) l.get("genres");   
                for (Document a : authors) {   
                    Author author = new Author(   
                            a.getString("author\_id"),   
                            a.getString("author\_name"),   
                            a.getString("author\_username"),   
                            "",   
                            "",   
                            null,   
                            0   
                    );   
                    authorsLis.add(author);   
                }   
                logsList.add(   
                        new Log(   
                                l.getString("id"),   
                                l.getDate("date"),   
                                l.getString("operation"),   
                                l.getString("admin"),   
                                l.getString("report\_id"),   
                                l.getString("type"),   
                                l.getString("isbn"),   
                                l.getString("asin"),   
                                l.getString("book\_id"),   
                                l.getString("title"),   
                                l.getString("description"),   
                                (Integer) l.get("num\_pages"),   
                                (Integer) l.get("publication\_day"),   
                                (Integer) l.get("publication\_month"),   
                                (Integer) l.get("publication\_year"),   
                                l.getString("image\_url"),   
                                "",   
                                "",   
                                "",   
                                "",   
                                "",   
                                authorsLis,   
                                genres   
                        )   
                );   
            }   
        }   
    } catch (Exception e) {   
        e.printStackTrace();   
    }   
    return logsList;   
}

* Update operations that allow us to keep updated books, author, users, reviews, logs and reports collections:

public boolean updatePassword(String newPassword) {   
    MongoCollection<Document> user = md.getCollection(session.getIsAuthor() ? *authorCollection* : *usersCollection*);   
    UpdateResult updateResult = null;   
    String username;   
    try {   
        if (session.getIsAuthor())   
            username = session.getLoggedAuthor().getNickname();   
        else   
            username = session.getLoggedUser().getNickname();   
        updateResult = user.updateOne(*eq*("username", username), Updates.*set*("password", newPassword));   
    } catch (Exception e) {   
        e.printStackTrace();   
    }   
    if (updateResult != null & updateResult.getModifiedCount() == 1)   
        return true;   
    return false;   
}

public Double updateRating(ArrayList<Review> reviews) {   
    Double ratingSum = 0.0;   
    if (reviews.size() > 0) {   
        for (Review r : reviews) {   
            ratingSum += Double.*parseDouble*(r.getRating());   
        }   
        return ratingSum / reviews.size();   
    } else {   
        return ratingSum;   
    }   
}

* Delete operations that allow us to delete books, author, users, reviews, logs and reports:

public boolean deleteBookMongo(Book book) {   
    MongoCollection<Document> books = *md*.getCollection(*bookCollection*);   
    DeleteResult deleteResult = null;   
    Book backup = getBookByID(book.getBook\_id());   
    try {   
        deleteResult = books.deleteOne(*eq*("book\_id", book.getBook\_id()));   
    }catch (Exception e){   
        e.printStackTrace();   
    }   
    if(deleteResult != null && deleteResult.getDeletedCount() == 1)   
        return true;   
    return false;   
}

public boolean deleteReview(String review\_id, String book\_id) {   
    MongoCollection<Document> books = *md*.getCollection(*bookCollection*);   
    Bson getBook = *eq*("book\_id", book\_id);   
   
    try {   
        UpdateResult removeReview = books.updateOne(getBook, Updates.*pull*("reviews", new Document("review\_id", review\_id)));   
        Book bookToUpdate = getBookByID(book\_id);   
        if (bookToUpdate == null) {   
            return true;   
        }   
        if (removeReview.getModifiedCount() == 1) {   
            Double newRating = updateRating(bookToUpdate.getReviews());   
            UpdateResult updateAvgRating = books.updateOne(getBook, Updates.*set*("average\_rating", newRating));   
            removeLikeReview(review\_id, book\_id);   
            if (updateAvgRating.getModifiedCount() == 1) {   
                return true;   
            }   
        }   
    }catch (Exception e){   
        e.printStackTrace();   
    }   
    return false;   
}

public boolean deleteUserMongo(User userDel, String type) {   
    MongoCollection<Document> user = md.getCollection(type.equals("author") ? *authorCollection* : *usersCollection*);   
    DeleteResult deleteResult = null;   
    try {   
        deleteResult = user.deleteOne(*eq*("username", userDel.getNickname()));   
    } catch (Exception e) {   
        e.printStackTrace();   
    }   
    if (deleteResult != null && deleteResult.getDeletedCount() == 1)   
        return true;   
    return false;   
}

public boolean deleteReport(Report report, Boolean unreport) {   
    MongoCollection<Document> reports = md.getCollection(*reportsCollection*);   
    DeleteResult result = null;   
    try {   
        result = reports.deleteOne(*eq*("report\_id", report.getReport\_id()));   
        if (unreport) {   
            if (addLog(report, "unreport")) {   
                return true;   
            } else {   
                return false;   
            }   
        } else {   
            if (addLog(report, "delete")) {   
                return true;   
            } else {   
                return false;   
            }   
        }   
    } catch (Exception e) {   
        e.printStackTrace();   
    }   
    if (result != null) {   
        return result.wasAcknowledged();   
    }   
    return false;   
}

public boolean deleteLog(Log log) {   
    MongoCollection<Document> reports = md.getCollection(*logsCollection*);   
    DeleteResult result = null;   
    try {   
        result = reports.deleteOne(*eq*("id", log.getId()));   
    } catch (Exception e) {   
        e.printStackTrace();   
    }   
    if (result != null) {   
        return result.wasAcknowledged();   
    }   
    return false;   
}

# Neo4J Analytics & Suggestions

* Users that are connected to the visualized user by a read/to\_read relationship through a book will be suggested to me If they are not already in my follow list

public ArrayList<User> similarUsers(String username, String type, String myUsername, String myType) {  
 ArrayList<User> suggestion = new ArrayList<>();  
 ArrayList<User> queryResult = new ArrayList<>();  
  
 try (Session session = nd.getDriver().session()) {  
 suggestion = session.readTransaction((TransactionWork<ArrayList<User>>) tx -> {  
 Result result = tx.run("MATCH (u:" + myType + "{username:'" + myUsername + "'}) " +  
 "OPTIONAL MATCH (u)-[:FOLLOW]->(f:User) " +  
 "WITH u, collect(f) as followed " +  
 "MATCH (u1:" + type + "{username:'"

+ username + "'})-[:READ|:TO\_READ]->(b:Book)<-[]-(u2:User) " +  
 "WHERE u1<>u2 AND u<>u2 AND NOT u2 IN followed " +  
 "RETURN DISTINCT u2.id,u2.name,u2.username");  
 while (result.hasNext()) {  
 Record r = result.next();  
 queryResult.add(new User(r.get("u2.id").asString(), r.get("u2.name").asString(), r.get("u2.username").asString(), "", "", new ArrayList<>(), 0));  
 }  
 return queryResult;  
 });  
 }  
 return suggestion;  
}

* Authors that are connected to the visualized user by a read/to\_read relationship through a book will be suggested to me If they are not already in my follow list

public ArrayList<Author> similarAuthors(String username, String type, String myUsername, String myType) {  
 ArrayList<Author> suggestion;  
 ArrayList<Author> queryResult = new ArrayList<>();  
 try (Session session = nd.getDriver().session()) {  
 suggestion = (ArrayList<Author>) session.readTransaction((TransactionWork<ArrayList<Author>>) tx -> {  
 Result result = tx.run("MATCH (u:" + myType + "{username:'"

+ myUsername + "'}) " +  
 "OPTIONAL MATCH (u)-[:FOLLOW]->(f:Author) " +  
 "WITH u, collect(f) as followed " +  
 "MATCH (u1:" + type + "{username:'"

+ username + "'})-[:READ|:TO\_READ]->(b:Book)<-[:READ|:TO\_READ]-(a:Author) " +  
 "WHERE u1<>a AND u<>a AND NOT a IN followed " +  
 "RETURN DISTINCT a.id,a.name,a.username");  
 while (result.hasNext()) {  
 Record r = result.next();  
 queryResult.add(new Author(r.get("a.id").asString(), r.get("a.name").asString(), r.get("a.username").asString(), "", "", new ArrayList<>(), 0));  
 }  
 return queryResult;  
 });  
 }  
 return suggestion;  
}

* Books that are connected to the visualized book by a wrote relationship through the authors (of the visualized book) will be suggested to me If they are not already in my read/to\_read list

public ArrayList<Book> similarBooks(String book\_id,String myUsername, String myType) {  
 ArrayList<Book> suggestion = new ArrayList<>();  
 ArrayList<Book> queryResult = new ArrayList<>();  
  
 try (Session session = *nd*.getDriver().session()) {  
 suggestion = session.readTransaction((TransactionWork<ArrayList<Book>>) tx -> {  
 Result result = tx.run("OPTIONAL MATCH (u1:" + myType + "{username:'" + myUsername + "'})-[:READ|:TO\_READ]->(b:Book) " +  
 "WITH collect(b) as readings " +  
 "MATCH (b1:Book)<-[:WROTE]-(a:Author)-[:WROTE]->(b2:Book) " +  
 "WHERE b1.id = '" + book\_id + "' AND b1<>b2 " +  
 "AND NOT b2 IN readings " +  
 "RETURN DISTINCT b2.id,b2.title");  
 while (result.hasNext()) {  
 Record r = result.next();  
 queryResult.add(new Book(r.get("b2.title").asString(), r.get("b2.id").asString()));  
 }  
 return queryResult;  
 });  
 }  
 return suggestion;  
}

* Authors that worked together with the authors, of the visualized book, on another paper will be suggested to me If they are not already in follow list

public ArrayList<Author> suggestedAuthors(String book\_id, String myUsername, String myType) {  
 ArrayList<Author> suggestion;  
 ArrayList<Author> queryResult = new ArrayList<>();  
  
 try (Session session = *nd*.getDriver().session()) {  
 suggestion = (ArrayList<Author>) session.readTransaction((TransactionWork<ArrayList<Author>>) tx -> {  
 Result result = tx.run("MATCH (u1:" + myType + "{username:'"

+ myUsername + "'}) " +  
 "OPTIONAL MATCH (u1)-[:FOLLOW]->(f:Author) " +  
 "WITH u1,collect(f) as followed " +  
 "MATCH (b1:Book)<-[:WROTE]-(a1:Author)-[:WROTE]->(b2:Book)<-[:WROTE]-(a2:Author) " +  
 "WHERE b1.id = '" + book\_id + "' AND b1<>b2 AND a1<>a2 " +  
 "AND NOT a2 IN followed AND u1<>a2 " +  
 "RETURN DISTINCT a2.id,a2.name,a2.username");  
 while (result.hasNext()) {  
 Record r = result.next();  
 queryResult.add(new Author(r.get("a2.id").asString(), r.get("a2.name").asString(), r.get("a2.username").asString(), "", "", new ArrayList<>(), 0));  
 }  
 return queryResult;  
 });  
 }  
 return suggestion;  
}

* Get the Top desired(to\_read)/popular(read) books

public ArrayList<RankingObject> topBooks(String type, Integer limit) {  
 ArrayList<RankingObject> books;  
 ArrayList<RankingObject> queryResult = new ArrayList<>();  
  
 try (Session session = *nd*.getDriver().session()) {  
 books = (ArrayList<RankingObject>) session.readTransaction((TransactionWork<ArrayList<RankingObject>>) tx -> {  
 Result result = tx.run("MATCH (b:Book)<-[r:" + type + "]-() " +  
 "RETURN b.id,b.title,count(r) as count " +  
 "ORDER BY count DESC " +  
 "LIMIT " + limit + "");  
 while (result.hasNext()) {  
 Record r = result.next();  
 queryResult.add(new RankingObject(r.get("b.title").asString(), Integer.*valueOf*(r.get("count").toString())));  
 }  
 return queryResult;  
 });  
 }  
 return books;  
}

# MongoDB Analitycs Implementations

* Average rating per category about author’s books

This analytics gives us informations about the average rating about each category in which an author has written books.

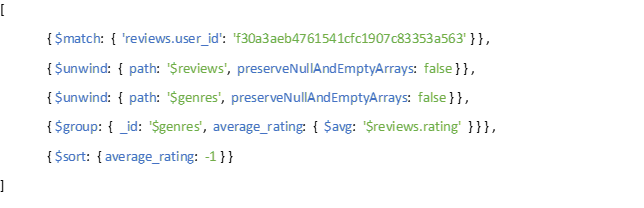


The Mongo Java Driver code:

public ArrayList<Genre> averageRatingCategoryAuthor(String username) {  
 MongoCollection<Document> author = md.getCollection(*authorCollection*);  
 MongoCollection<Document> books = md.getCollection(*bookCollection*);  
 String author\_id = null;  
 ArrayList<Genre> topRated = new ArrayList<>();  
 Bson getAuthor;  
 Bson unwindGenres;  
 Bson groupGenres;  
 Bson sortAvg;  
try (MongoCursor<Document> cursor = author.find(*eq*("username", username)).iterator()) {  
 while (cursor.hasNext()) {  
 author\_id = cursor.next().getString("author\_id");  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 return topRated;  
 }  
 if (author\_id == null)  
 return null;  
 getAuthor = *match*(*eq*("authors.author\_id", author\_id));  
 unwindGenres = *unwind*("$genres", new UnwindOptions().preserveNullAndEmptyArrays(false));  
 groupGenres = *group*("$genres", *avg*("average\_rating", "$average\_rating"));  
 sortAvg = *sort*(*orderBy*(*descending*("average\_rating")));  
 try (MongoCursor<Document> cursor = books.aggregate(Arrays.*asList*(getAuthor, unwindGenres, groupGenres, sortAvg)).iterator()) {  
 while (cursor.hasNext()) {  
 Document stat = cursor.next();  
 Double avg = Math.*round*((stat.getDouble("average\_rating")) \* 100) / 100.0;  
 Genre genre = new Genre(stat.getString("\_id"), Double.*valueOf*(avg));  
 topRated.add(genre);  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 return topRated;  
}

* Average review rating grouped by books categories

This analytics gives us informations about the average rating about each category in which an user has left a review.



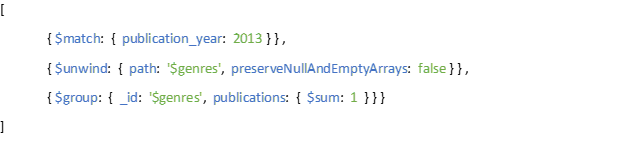
The Mongo Java Driver code:

public ArrayList<Genre> averageRatingCategoryUser(String username) {  
 MongoCollection<Document> books = md.getCollection(*bookCollection*);  
 ArrayList<Genre> topRated = new ArrayList<>();  
 Bson getUser;  
 Bson unwindReviews;  
 Bson unwindGenres;  
 Bson groupGenres;  
 Bson sortAvg;  
 getUser = *match*(*eq*("reviews.username", username));  
 unwindReviews = *unwind*("$reviews", new UnwindOptions().preserveNullAndEmptyArrays(false));  
 unwindGenres = *unwind*("$genres", new UnwindOptions().preserveNullAndEmptyArrays(false));  
 groupGenres = *group*("$genres", *avg*("average\_rating", "$reviews.rating"));  
 sortAvg = *sort*(*orderBy*(*descending*("average\_rating")));  
 try (MongoCursor<Document> cursor = books.aggregate(Arrays.*asList*(unwindReviews, getUser, unwindGenres, groupGenres, sortAvg)).iterator()) {  
 while (cursor.hasNext()) {  
 Document stat = cursor.next();  
 Double avg = Math.*round*((stat.getDouble("average\_rating")) \* 100) / 100.0;  
 Genre genre = new Genre(stat.getString("\_id"), Double.*valueOf*(avg));  
 topRated.add(genre);  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 return topRated;

}

* Books published

This analytics gives us informations about the number of book published for each category filtered by year

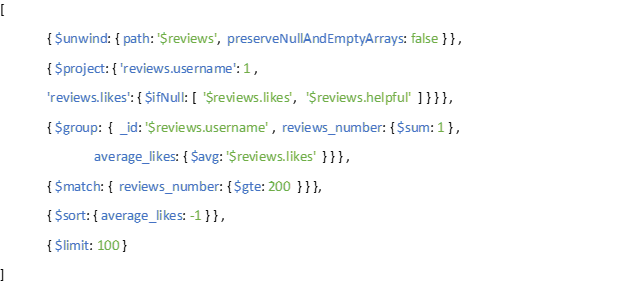


The Mongo Java Driver code:

public ArrayList<Genre> searchRankBook(Integer year) {  
 MongoCollection<Document> bookGenres = *md*.getCollection(*bookCollection*);  
  
 ArrayList<Genre> genres = new ArrayList<>();  
 Bson match = *match*(*in*("publication\_year", year));  
 Bson unwind = *unwind*("$genres");  
 Bson group = *group*("$genres", *sum*("counter", 1));  
  
 try (MongoCursor<Document> result = bookGenres.aggregate(Arrays.*asList*(match, unwind, group)).iterator()) {  
  
 while (result.hasNext()) {  
 Document y = result.next();  
 genres.add(new Genre(y.getString("\_id"), Double.*valueOf*(y.get("counter").toString())));  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
  
 return genres;  
}

* Users rank

This analytics gives us the users rank based on the average review rating received



The Mongo Java Driver code:

public ArrayList<RankingObject> rankReview() {  
 MongoCollection<Document> book = *md*.getCollection(*bookCollection*);  
 ArrayList<RankingObject> users = new ArrayList<>();  
  
 Bson unwindReviews = *unwind*("$reviews");  
 *//project likes : ($likes != null) $likes : $helpful*  
Bson projectLikes = new Document("$project",  
 new Document("reviews.username", 1L)  
 .append("reviews.likes",  
 new Document("$ifNull", Arrays.*asList*("$reviews.likes", "$reviews.helpful"))));  
 Bson groupUsername = *group*("$reviews.username", *sum*("reviews\_number", 1), *avg*("average\_likes", "$reviews.likes"));  
 Bson matchGreaterThan200 = *match*(*gte*("reviews\_number", 200));  
 Bson sort = *sort*(*orderBy*(*descending*("average\_likes")));  
 Bson limit = *limit*(100);  
  
 try (MongoCursor<Document> result = book.aggregate(Arrays.*asList*(unwindReviews, projectLikes, groupUsername, matchGreaterThan200, sort, limit)).iterator()) {  
  
 while (result.hasNext()) {  
 Document document = result.next();  
 users.add(new RankingObject(document.getString("\_id"),  
 document.getInteger("reviews\_number"),  
 document.getDouble("average\_likes")));  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 return users;  
  
}

# Neo4J Index Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| Constraint Name | Constraint  Type | Constraint  Lable | Constraint  Properties |
| user\_username | BTREE [UNIQUE] | :User | username |
| author\_username | BTREE [UNIQUE] | :Author | username |
| book\_id | BTREE [UNIQUE] | :Book | id |

## author\_username

Match (a:Author{username:"elmo.kling"})-[f:FOLLOW]->(p) return p;

Without constrain

Immagine che contiene tavolo

Descrizione generata automaticamente

With constrain Immagine che contiene tavolo

Descrizione generata automaticamente

## user\_username

Match (a:User{username:"rodrigo.sporer"})-[f:FOLLOW]->(p) return p;

Without constrain

Immagine che contiene tavolo

Descrizione generata automaticamente

With constrain

Immagine che contiene tavolo

Descrizione generata automaticamente

MATCH (u: User {username: 'rodrigo.sporer'})   
 OPTIONAL MATCH (u)-[:FOLLOW]->(f:User)   
 WITH u, collect(f) as followed   
 MATCH (u1: User {username:'rodrigo.sporer'})-[:READ|:TO\_READ]->(b:Book)<-[]-(u2:User)   
 WHERE u1<>u2 AND u<>u2 AND NOT u2 IN followed   
 RETURN DISTINCT u2.id,u2.name,u2.username

Without constrain

Immagine che contiene tavolo

Descrizione generata automaticamente

With constrain

Immagine che contiene tavolo

Descrizione generata automaticamente

## user\_username + book\_id

OPTIONAL MATCH (u1:User {username:'rodrigo.sporer'})-[:READ|:TO\_READ]->(b:Book)   
 WITH collect(b) as readings   
 MATCH (b1:Book)<-[:WROTE]-(a:Author)-[:WROTE]->(b2:Book)   
 WHERE b1.id = '1819013' AND b1<>b2   
 AND NOT b2 IN readings   
 RETURN DISTINCT b2.id,b2.title

Without constrains

Immagine che contiene tavolo

Descrizione generata automaticamente

With user\_username constrain

Immagine che contiene tavolo

Descrizione generata automaticamente

With user\_username and book\_id constrain

Immagine che contiene tavolo

Descrizione generata automaticamente

MATCH (u1: User {username: 'rodrigo.sporer'})   
 OPTIONAL MATCH (u1)-[:FOLLOW]->(f:Author)   
 WITH u1,collect(f) as followed   
 MATCH (b1:Book)<-[:WROTE]-(a1:Author)-[:WROTE]->(b2:Book)<-[:WROTE]-(a2:Author)   
 WHERE b1.id = '1819013' AND b1<>b2 AND a1<>a2   
 AND NOT a2 IN followed AND u1<>a2   
 RETURN DISTINCT a2.id,a2.name,a2.username

Without constrains

Immagine che contiene tavolo

Descrizione generata automaticamente

With user\_username constrain

Immagine che contiene tavolo

Descrizione generata automaticamente

With user\_username and book\_id constrain

Immagine che contiene tavolo

Descrizione generata automaticamente

# Mongo Index Analysis

We decided to use indexes in order to speed up the application. We performed some test to measure the speed improvement obtained by using indexes. This test is carried out using the *explain()* function offered by MongoDB.

All operations have been tested on virtual machines.

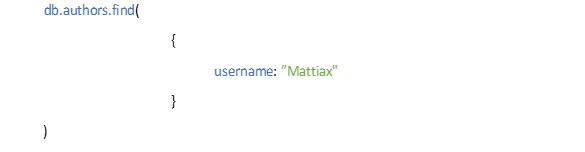
|  |  |  |  |
| --- | --- | --- | --- |
| Index Name | Index Type | Collection | Attributes |
| Username\_author | Unique | Author | username |
| Username\_user | Unique | Users | username |
| Book\_ID | Unique | Books | Book\_id |
| Book\_genres |  | Books | genres |
| Book\_title | Compound | Books | title |
| Find\_review |  | Books | reviews.review\_id |

## Authors and Users Collections test

We consider introducing the index on the attributes “username” in the authors and users collections to improve operations that require it.

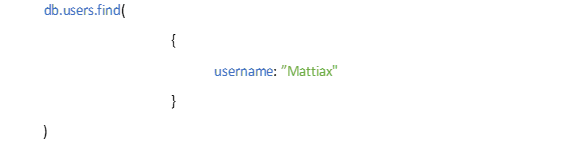
For example we have the login operations:

* Login as author



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Index | Documents returned | Index keys examined | Documents examined | Execution Time (ms) |
| False | 1 | 0 | 59605 | 218 |
| True | 1 | 1 | 1 | 0 |

* Login as user



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Index | Documents returned | Index keys examined | Documents examined | Execution Time (ms) |
| False | 1 | 0 | 150985 | 694 |
| True | 1 | 1 | 1 | 0 |

## Books Collection test

We consider introducing the index on the attributes “genres” in the books collection to improve operations that filter book by genres.

For example, we have the search by genre operation:

* Find book by genre

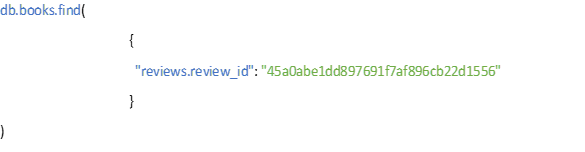


|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Index | Documents returned | Index keys examined | Documents examined | Execution Time (ms) |
| False | 29948 | 0 | 131111 | 226 |
| True | 29948 | 29948 | 29948 | 85 |

We consider to introduce also the index on the attributes “reviews.review\_id” in the books collection to improve operations that return review document.

For example, we have the research operation of review by id:

* Find review by review\_id

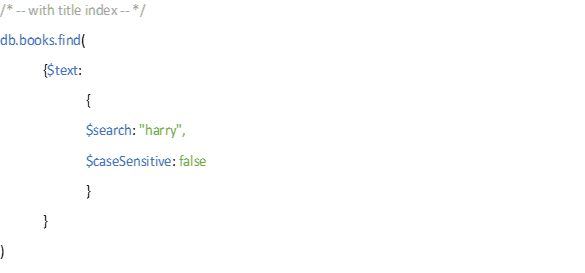


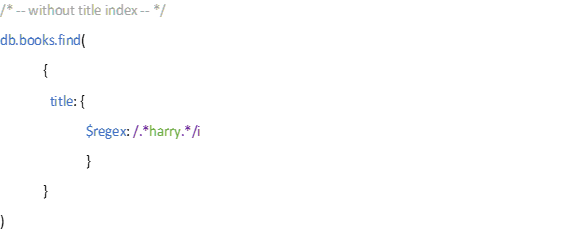
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Index | Documents returned | Index keys examined | Documents examined | Execution Time (ms) |
| False | 1 | 0 | 131110 | 523 |
| True | 1 | 1 | 1 | 0 |

We consider introducing also the index on the attribute “title” in the books collection to improve operations that filter books by title.

For example, we have the research operation by title:

* Find book by title





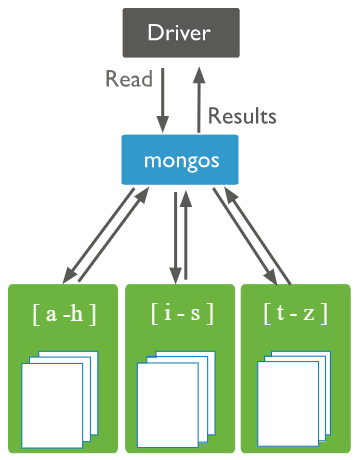
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Index | Documents returned | Index keys examined | Documents examined | Execution Time (ms) |
| False | 331 | 0 | 131112 | 285 |
| True | 331 | 331 | 331 | 2 |

# Sharding proposal

We propose to implement a *Range Based Strategy* about the dataset portion formed by the largest collections Users, Authors and Books to speed up reading operations.

This idea is based on the objective of speeding up document search operations; these operations are performed on username field for Users and Authors collections and on title field for Books collection.

Immagine che contiene testo, dispositivo, calibro

Descrizione generata automaticamenteWe decided to split the dataset portion in 3 chunks, each chunk contains a dataset portion defined by a range. The ranges are *[a-h], [i-s],[t-z]*.

We also think to use 3 replicas set for each shard in order to avoid failures and maintain high availability.

Chart, pie chart

Description automatically generated               Chart, pie chart

Description automatically generated

Chart, pie chart

Description automatically generated