



Feyzan Colak, Flavio Messina, Noemi Cherchi
Large-scale and multi-structured databases project
2023-2024

Contents

| | |
|--|-----------|
| Contents | 1 |
| 1 Introduction | 2 |
| 2 Analysis | 3 |
| 2.1 Actors | 3 |
| 2.2 Requirements | 3 |
| 2.3 Non Functional Requirements | 5 |
| 2.4 UML use case diagram | 6 |
| 2.5 Scenarios | 7 |
| 2.6 UML class diagram | 13 |
| 2.7 Data Modeling | 14 |
| 3 Design | 15 |
| 3.1 Document Database | 15 |
| 3.2 Graph Database | 20 |
| 3.3 Availability and Partition Tolerance | 22 |
| 3.4 Redundancy | 22 |
| 3.5 Replicas | 22 |
| 3.6 Sharding | 22 |
| 4 Implementation | 23 |
| 4.1 Development Environment | 23 |
| 4.2 Main Modules | 23 |
| 4.3 Adopted Patterns and Techniques | 29 |
| 4.4 Description of Main Classes | 29 |
| 4.5 MongoDB queries | 33 |
| 4.6 GraphDB queries | 43 |
| 5 Testing | 52 |
| 5.1 Structural Testing | 52 |
| 5.2 Functional Testing | 55 |
| 5.3 Performance Testing | 56 |
| 6 Conclusion | 58 |
| 6.1 Conclusion | 58 |
| 6.2 Future Work | 58 |

Introduction

MangaVerse is a web application project developed for the Large-scale and multi-structured databases course of the University of Pisa. This web application aims to provide users with a comprehensive platform to explore, search, interact with a vast collection of manga and anime and interact with the other users.

The website is accessible without login providing a limited number of functionalities. Once a user logs in, the platform offers a wide range of features to personalize the user experience, in particular the social features. The application manages user and media content suggestions based on interactions, preferences and user information. Beside the user roles, the web application also has managerial roles. MangaVerse provides an analytics dashboards to track media contents and user activities also managing media contents and user accounts. These features allow manager to add, update, or remove manga and anime entries and monitor trends and rating.

Through its comprehensive set of features, MangaVerse aims to provide a community of manga and anime enthusiasts with a platform to explore, share, and engage with their favorite content. This platform enhances the user experience and facilitates deep engagement with both the content and the community.

Analysis

Actors

- **Unregistered User:** A visitor who has not logged in on the platform.
- **Registered User:** A user who has created an account on the platform.
- **Manager:** A registered user with administrative privileges.

Requirements

Unregistered User:

- Register/Login:
 - Create a new account to access additional features.
 - Use valid credentials (email and password) to log into the account.
- Browse Media Contents.
- Search and Filter Media Contents:
 - Find specific manga or anime by title.
 - Utilize basic filtering options to refine the media content list.
- View Media Content Trends.
- View Media Content:
 - View limited information about each media content.
- View Media Content Details:
 - View detailed information about each media content.
 - View reviews and ratings for each media content.
 - View number of likes for each media content.
- Browse Users.
- Search Users by Username.
- View User:
 - View limited information about each user.
- View User Details:
 - View detailed information about each user.
 - View anime and manga liked by the user.
 - View followers and following of the user.

Registered User:

- Logout.
- Browse Media Contents.
- Search and Filter Media Contents:

- Find specific manga or anime by title.
 - Utilize basic filtering options to refine the media content list.
- View Media Content Trends.
- View Media Content:
 - View limited information about each media content.
- View Media Content Details:
 - View detailed information about each media content.
 - View reviews and ratings for each media content.
 - View number of likes for each media content.
- Browse Users.
- Search Users by Username.
- View User:
 - View limited information about each user.
- View User Details:
 - View detailed information about each user.
 - View anime and manga liked by the user.
 - View followers and following of the user.
- Profile Management:
 - Edit and update personal information (e.g., profile picture, bio).
 - Delete own profile.
- Like/Unlike Media Contents.
- Follow/Unfollow Users.
- Review Media Contents:
 - Add comment and rating to manga and anime.
 - Edit/Delete own reviews.
- Advanced Recommendations:
 - Receive media content suggestions based on user interactions and personal information.
 - Receive users suggestions based on user interactions.

Manager(Registered User with Administrative Features):

- Logout.
- Browse Media Contents.
- Search and Filter Media Contents:
 - Find specific manga or anime by title.
 - Utilize basic filtering options to refine the media content list.
- View Media Content Trends.
- View Media Content:
 - View limited information about each media content.

- View Media Content Details:
 - View detailed information about each media content.
 - View reviews and ratings for each media content.
 - View number of likes for each media content.
- Browse Users.
- Search Users by Username.
- View User:
 - View limited information about each user.
- View User Details:
 - View detailed information about each user.
 - View anime and manga liked by the user.
 - View followers and following of the user.
- Analytics Dashboard:
 - View user analytics (distribution and app rating).
 - View manga analytics (trends and average rating).
 - View anime analytics (trends and average rating).
- Content Management:
 - Add new media content (manga and anime).
 - Update/Remove existing media content.

Non Functional Requirements

Performance

- Response Time: The system should have low latency, with pages loading within an acceptable timeframe.
- Scalability: The system should be able to handle an increasing number of users and data without significant degradation in performance.
- Concurrency: The application should support multiple users simultaneously without performance bottlenecks. For very high traffic scenarios, acceptable delays may be introduced.
- Availability: The system should be available 24/7, with minimal downtime for maintenance.
- Replication: The system should have data replication to ensure data availability and fault tolerance.

Security

- Controlled User Operations: Users should only be able to perform operations that they are authorized to do.

Data Integrity

- Data Consistency: The system should maintain data consistency across all components and databases.

User Interface

- Responsiveness: The user interface should be responsive, providing a consistent and seamless experience across various devices and screen sizes.
- Intuitiveness: The interface should be user-friendly, with clear navigation and easily understandable features.

Scenarios

| Use Case | Register Account |
|----------------------|--|
| Primary Actor | Unregistered User |
| Description | Allows user to sign up |
| Pre-conditions | The account must not be already exist |
| Main Event Steps | <ol style="list-style-type: none">1. The user navigates to the registration page.2. The user fills out the registration form with valid information.3. The system validates the entered information.4. After successful validation, the system created a new user account |
| Post-conditions | The user is registered to the system |
| Correlated Use Cases | - |

| Use Case | Log In |
|----------------------|---|
| Primary Actor | Registered User |
| Description | Allows user to login |
| Pre-conditions | The user must not be already logged in |
| Main Event Steps | <ol style="list-style-type: none">1. The user navigates to the login page.2. The system shows the login form to the user and asks for email and password3. The user enters valid login credentials.4. The system validates the entered credentials5. After successful validation, the system logs the user. |
| Post-conditions | <ol style="list-style-type: none">1. The user is authenticated and logged in2. The user is redirected to the main page for registered user. |
| Correlated Use Cases | - |

| Use Case | Log Out |
|----------------------|---|
| Primary Actor | Registered User |
| Description | Allows user to logged out |
| Pre-conditions | The user must be logged in |
| Main Event Steps | The system starts the logout process |
| Post-conditions | <p>The user is logged out.</p> <p>The user is directed to the main page</p> |
| Correlated Use Cases | - |

| Use Case | Browse Anime/Manga |
|----------|--------------------|
|----------|--------------------|

| | |
|----------------------|--|
| Primary Actor | Registered User, Unregistered User |
| Description | Allows both registered and unregistered users to browse anime and manga |
| Pre-conditions | - |
| Main Event Steps | 1. The user navigates to the browse section. 2. The system displays a list of available anime and manga. |
| Post-conditions | 1. The user will view the list of anime or manga 2. The user can filter or sort the list based on various criteria. |
| Correlated Use Cases | Find Anime/Manga |

| | |
|----------------------|---|
| Use Case | Find Anime/Manga |
| Primary Actor | Registered User, Unregistered User |
| Description | Allows users to search for specific anime or manga. |
| Pre-conditions | - |
| Main Event Steps | 1. The user navigates to the search section of anime/manga. 2. The user enters search criteria (e.g., title). 3. The system retrieves a list of available anime/manga. 4. The user can apply filters or sort the list. |
| Post-conditions | The system displays a result of the search The user views a list of anime/manga matching with the used criteria |
| Correlated Use Cases | View Anime/Manga |

| | |
|----------------------|--|
| Use Case | View Anime/Manga |
| Primary Actor | Registered User, Unregistered User |
| Description | Allows users to view details of a specific anime or manga |
| Pre-conditions | - |
| Main Event Steps | 1. The user selects an anime or manga from the list or search results. 2. The system displays pre information about the selected anime/manga. |
| Post-conditions | The user views pre information about the selected anime/manga |
| Correlated Use Cases | - |

| | |
|------------------|--|
| Use Case | View Anime/Manga Details |
| Primary Actor | Registered User, Unregistered User |
| Description | Allows users to view all the information about the specific anime or manga including reviews |
| Pre-conditions | - |
| Main Event Steps | 1. The user selects an anime or manga. |

| | |
|----------------------|--|
| | 2. The system retrieves detailed information, including reviews. |
| Post-conditions | |
| Correlated Use Cases | - |

| Use Case | View Anime/Manga Reviews |
|----------------------|---|
| Primary Actor | Registered User, Unregistered User |
| Description | Allows users to view reviews of a specific anime or manga |
| Pre-conditions | - |
| Main Event Steps | <ol style="list-style-type: none"> 1. The user navigates to the reviews section of a specific anime or manga. 2. The system retrieves the reviews. 3. The system displays the reviews to the user. |
| Post-conditions | The user views the reviews of the specific anime or manga |
| Correlated Use Cases | - |

| Use Case | Show Anime/Manga Suggestions |
|----------------------|---|
| Primary Actor | User |
| Description | Allows users to view anime or manga suggestions based on their likes or followings |
| Pre-conditions | User must be logged in |
| Main Event Steps | <ol style="list-style-type: none"> 1. The user navigates to the suggestions part of the main page. 2. The system retrieves suggestions based on user preferences and followings. 3. The system displays the suggestions to the user. |
| Post-conditions | The user views the suggested anime or manga |
| Correlated Use Cases | View Manga/Anime |

| Use Case | Browse Anime/Manga Trend |
|----------------------|---|
| Primary Actor | Registered User, Unregistered User |
| Description | Allows users to browse trending anime or manga |
| Pre-conditions | User must be logged in |
| Main Event Steps | <ol style="list-style-type: none"> 1. The user navigates to the trending section of the main page. 2. The system retrieves trending anime or manga. 3. The system displays the trending items to the user. |
| Post-conditions | - |
| Correlated Use Cases | View Anime/Manga |

| Use Case | Like Anime/Manga |
|----------|------------------|
|----------|------------------|

| | |
|----------------------|--|
| Primary Actor | User |
| Description | Allows users to like an anime or manga |
| Pre-conditions | User must be logged in |
| Main Event Steps | <ol style="list-style-type: none"> 1. The user clicks the like or unlike button. 2. The system updates the like status. |
| Post-conditions | <ol style="list-style-type: none"> 1. The anime or manga is liked or liked by the user and added to the liked anime/manga list 2. The appearance of like button is changed to filled version of the button |
| Correlated Use Cases | - |

| | |
|----------------------|--|
| Use Case | Add Review to Anime/Manga |
| Primary Actor | User |
| Description | Allows user to make comment or rating for specific anime/manga |
| Pre-conditions | User must be logged in |
| Main Event Steps | <ol style="list-style-type: none"> 1. The user selects an anime or manga. 2. The user navigates to the review section. 3. The user adds a review that can be either comment or rating the anime/-manga 4. The system ensures that at least one of the comment or rate value is filled 5. The system saves the review and updates the latest reviews |
| Post-conditions | The review is added to the specific anime/manga and visible on the page |
| Correlated Use Cases | - |

| | |
|----------------------|---|
| Use Case | Show User Suggestions |
| Primary Actor | User |
| Description | Allows users to view suggested users based on similar tastes or may know |
| Pre-conditions | User must be logged in |
| Main Event Steps | <ol style="list-style-type: none"> 1. The user navigates to the profile page's suggested users section 2. The system retrieves suggested users based on user preferences and followings 3. The system displays the suggested users to the user |
| Post-conditions | The user views the suggested users. |
| Correlated Use Cases | View User |

| | |
|-----------------|------------------|
| Use Case | Find User |
|-----------------|------------------|

| | |
|----------------------|--|
| Primary Actor | Registered User, Unregistered User |
| Description | Allows users to search for specific users according to their usernames |
| Pre-conditions | - |
| Main Event Steps | <ol style="list-style-type: none"> 1. The user navigates to the user user search section of the page. 2. The user enters search criteria. 3. The system retrieves matching users based on criteria. 4. The system displays the search results to the user. |
| Post-conditions | The user views the search results. |
| Correlated Use Cases | View User |

| | |
|----------------------|--|
| Use Case | Browse Followings/Followers |
| Primary Actor | Registered User, Unregistered User |
| Description | Allows users to browse their or others followers and followings |
| Pre-conditions | - |
| Main Event Steps | <ol style="list-style-type: none"> 1. The user navigates to the profile page. 2. The user selects the followers or followings section. 3. The system retrieves the list of followers or followings. 4. The system displays the list to the user. |
| Post-conditions | The user views their followers and followings. |
| Correlated Use Cases | Find User,View User Info |

| | |
|----------------------|---|
| Use Case | Follow User |
| Secondary Actor | - |
| Description | Allows users to follow another user |
| Pre-conditions | User must be logged in |
| Main Event Steps | <ol style="list-style-type: none"> 1. The user selects a user to follow 2. The user navigates to that user's profile page. 3. The systems displays the profile of other user. 4. The user clicks the follow button. 5. The system add the user to the following list of the other user and updates the count on the profile 6. The system updates the follow/unfollow status. |
| Post-conditions | The user starts to follow the other user |
| Correlated Use Cases | - |

| | |
|-----------------|------------------------------|
| Use Case | View User Information |
|-----------------|------------------------------|

| | |
|----------------------|---|
| Primary Actor | Registered User, Unregistered User |
| Description | Allows users to view detailed information about another user |
| Pre-conditions | - |
| Main Event Steps | <ol style="list-style-type: none"> 1. The user selects a user from the list or search results or browse 2. The system retrieves detailed information about the selected user. 3. The system displays the detailed information to the user. |
| Post-conditions | The user views detailed information about the selected user. |
| Correlated Use Cases | - |

| Use Case | View Analytics |
|----------------------|---|
| Primary Actor | Manager |
| Description | Allows managers to view analytics related to users and anime/manga |
| Pre-conditions | The user must be logged in with manager credentials |
| Main Event Steps | <ol style="list-style-type: none"> 1. The manager navigates to the analytics section of their page 2. The system retrieves analytics data. 3. The system displays the analytics data to the manager. |
| Post-conditions | The manager views the analytics data. |
| Correlated Use Cases | - |

UML class diagram

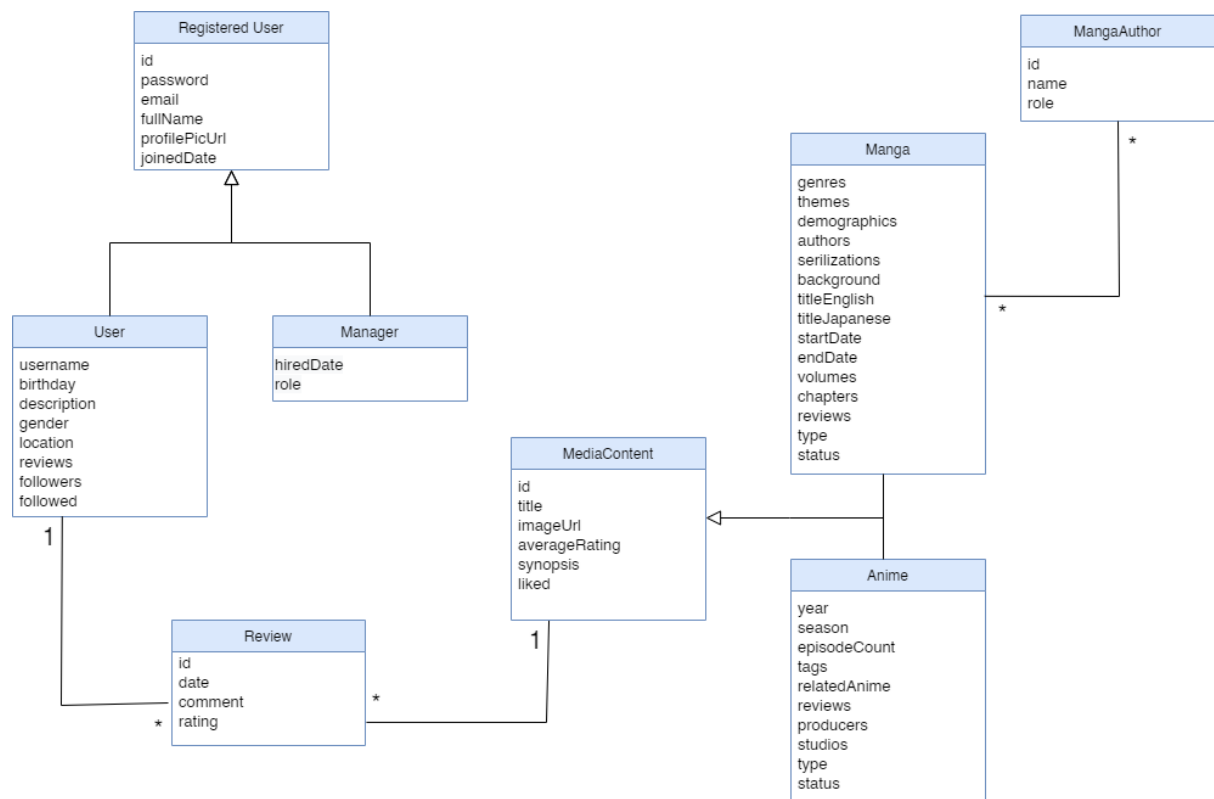


Figure 2.2: UML Class Diagram

Data Modeling

Data Collection

Sources: <https://www.kaggle.com/datasets/dbdmobile/myanimelist-dataset?select=users-score-2023.csv>, MyAnimeList.net, anilist.com, kitsu.io, livechart.me, anime-planet.com, nofity.moe, anisearch.com, anidb.net

Description: Manga, users and scores datasets were collected from MyAnimeList.net site using the official API and another unofficial API (Jikan). The anime datasets were collected from all the sources.

Variety: The datasets contain a variety of data types, including text, numbers, and dates. Anime are collected from 8 different sources. All the information is collected in 4 different csv files.

Volume: The datasets contain a large volume of data, with thousands of entries for anime, manga, users, and scores. The total size of the datasets is around 3 GB.

Data Cleaning and Preprocessing

Python scripts were used to clean and preprocess the data. The following steps were performed: reviews were created by merging the users and scores datasets, and creating comments about the media contents; the anime dataset was created by putting together the different sources; the manga dataset was created from MyAnimeList.net; the users dataset was cleaned and missing information, like email and password, was added.

Design

The web application needs to handle a big amount of data, so we decided to use a combination of different databases to store and manage the data. We will use a document database to store users, media contents and reviews data, and a graph database to store relationships between users and media content. This will allow us to efficiently store and retrieve data, as well as handle complex relationships between data.

Document Database

For the document database, we will use MongoDB. MongoDB is a NoSQL database that stores data in flexible, JSON-like documents. It is a popular choice for applications that require flexibility and scalability. These documents are flexible, meaning they can have different fields and structures. This makes MongoDB a good choice for applications that require flexibility in their data model. MongoDB is also a scalable database, meaning it can handle large amounts of data and traffic. It is designed to scale out, meaning you can add more servers to handle more traffic.

Collections The database will have the following collections:

- Anime: This collection will store information about anime, such as titles, tags, and synopsis.
- Manga: This collection will store information about manga, such as titles, genres, and authors.
- Reviews: This collection will store user ratings and comments for media content.
- Users: This collection will store user data, such as usernames, passwords, email addresses, gender and location.

MongoDB document example

Anime:

```
1 {
2   "_id": "65789bb52f5d29465d0abcfb",
3   "title": "0",
4   "type": "SPECIAL",
5   "episodes": 1,
6   "status": "FINISHED",
7   "picture": "https://cdn.myanimelist.net/images/anime/12/81160.jpg",
8   "tags": [
9     "drama",
10    "female protagonist",
11    "indefinite",
12    "music",
13    "present"
14  ],
15  "producers": "Sony Music Entertainment",
16  "studios": "Minakata Laboratory",
17  "synopsis": "This music video tells how a shy girl with a secret love and curiosity...",
18  "latest_reviews": [
19    {
20      "id": "657b301306c134f18884924c",
21      "date": "2023-10-03T22:00:00.000+00:00",
22      "rating": 4,
23      "user": {
24        "id": "6577877ce68376234760745c",
25        "username": "Tolstij_Trofim",
26        "picture": "https://thypix.com/wp-content/uploads/2021/10/manga-profile-picture-10..."
27      }
28    },
29  ],
30  "anime_season": {
31    "season": "FALL",
32    "year": 2013
33  },
34  "average_rating": 6.7,
35  "avg_rating_last_update": true,
36  "likes": 4
37 }
38
```

Manga:

```
1 {
2   "_id": "657ac61bb34f5514b91ea223",
3   "title": "Berserk",
4   "type": "MANGA",
5   "status": "ONGOING",
6   "genres": [
7     "Action",
8     "Adventure",
9     "Award Winning",
10    "Drama",
11    "Fantasy",
12    "Horror",
13    "Supernatural"
14  ],
15  "themes": [
16    "Gore",
17    "Military",
18    "Mythology",
19    "Psychological"
20  ],
21  "demographics": [
22    "SEINEN"
23  ],
24  "authors": [
25    {
26      "id": 1868,
27      "role": "Story & Art",
28      "name": "Kentarou Miura"
29    },
30    {
31      "serializations": "Young Animal"
32    }
33  ],
34  "synopsis": "Guts, a former mercenary now known as the \"Black Swordsman,\" is out fo...
35  ",
36  "title_english": "Berserk",
37  "start_date": "1989-08-25T00:00:00.000+00:00",
38  "picture": "https://cdn.myanimelist.net/images/manga/1/1578971.jpg",
39  "average_rating": 3.33,
40  "latest_reviews": [
41    {
42      "user": {
43        "id": "6577877be683762347605ce7",
44        "username": "calamity_razes",
45        "picture": "https://imgbox.com/7MaTkbQR"
46      },
47      "date": "2012-12-15T00:00:00.000+00:00",
48      "comment": "An insult to the art of manga; avoid at all costs.",
49      "id": "657b302206c134f18886f5ef"
50    },
51  ],
52  "anime_season": {
53    "season": "FALL",
54    "year": 2013
55  },
56  "average_rating": 6.7,
57  "avg_rating_last_update": true,
58  "likes": 4
59 }
```

Reviews:

```
1 {
2   "_id": "657b300806c134f18882f2f1",
3   "user": {
4     "id": "6577877be68376234760596d",
5     "username": "Dragon_Empress",
6     "picture": "images/account-icon.png",
7     "location": "Columbus, Georgia",
8     "birthday": "1987-07-29T00:00:00.000+00:00",
9     "rating": 7
10  },
11  "anime": {
12    "id": "65789bbc2f5d29465d0b18b7",
13    "title": "Slayers Revolution",
14    "date": "2023-07-23T06:27:54.000+00:00",
15    "comment": "Above-average quality in animation and soundtrack."
16  }
17 }
18
```

Users:

```
1 {
2   "_id": "6577877be683762347605859",
3   "email": "xdavis@example.com",
4   "password": "290cb38a679d5eb68d11b9eale21f48234eba6de19f95612dbcb70ce0c7e4e78",
5   "description": "Liberating the mind from stress with the power of anime zen.",
6   "picture": "https://thypix.com/wp-content/uploads/2021/10/manga-profile-picture-44",
7   "username": "Xinil",
8   "gender": "Male",
9   "birthday": "1985-03-04T00:00:00.000+00:00",
10  "location": "Libya",
11  "joined_on": "2014-05-29T00:00:00.000+00:00",
12  "app_rating": 5,
13  "followed": 40,
14  "followers": 29
15 }
16
```

The field "app_rating" is used to know the general satisfaction of the user with the application.

CRUD operations

- Create: This operation will allow users to create new documents in the database. For example, users can create new reviews for anime and manga.
- Read: This operation will allow users to read documents from the database. For example, users can read information about anime and manga and about other users.
- Update: This operation will allow users to update documents in the database. For example, users can update their reviews for anime and manga, they can also update their own profile, the manager can update media contents.
- Delete: This operation will allow users to delete documents from the database. For example, users can delete their reviews for anime and manga, the manager can delete media contents.

Graph Database

For the graph database, we will use Neo4j. Neo4j is a graph database that stores data in nodes and relationships. It is a popular choice for applications that require complex relationships between data. Neo4j is a graph database, which means it stores data in nodes and relationships. Nodes represent entities, such as users or products, and relationships represent connections between nodes. This makes Neo4j a good choice for applications that require complex relationships between data. Neo4j is also a scalable database, meaning it can handle large amounts of data and traffic. It is designed to scale out, meaning you can add more servers to handle more traffic. This makes Neo4j a good choice for applications that need to scale quickly.

Nodes

The database will have the following nodes:

- User: This node will store information about users, such as id, usernames, and picture.
- Anime: This node will store information about anime, such as id, titles and picture.
- Manga: This node will store information about manga, such as id, titles and picture.

Relationships

The database will have the following relationships:

- LIKE: This relationship will connect users to anime and manga nodes. It will store the date when the user liked the media content.
- FOLLOW: This relationship will connect users to other users.

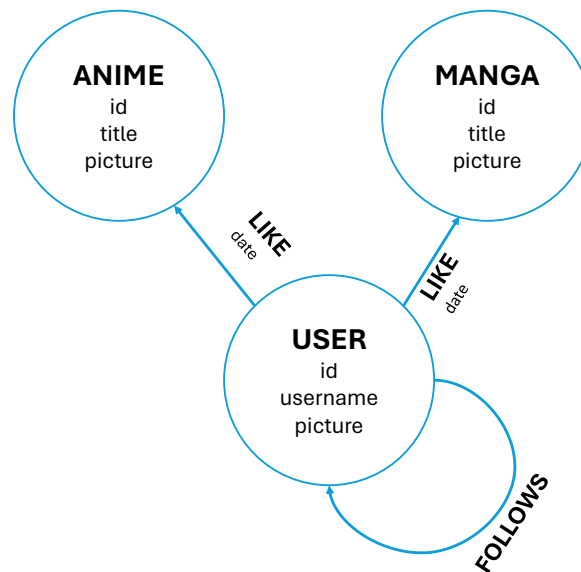


Figure 3.1: GraphDB

CRUD operations

- Create: This operation will allow users to create new nodes and relationships in the database. For example, users can create new relationships between users and media content:

A user can LIKE a media content:

```
1 MATCH (u:User {id: $userId}), (a:Anime {id: $animeId})
2 WHERE NOT (u)-[:LIKE]->(a)
3 CREATE (u)-[r:LIKE {date: $date} ]->(a)
4 RETURN r
5
```

Listing 3.1: Create Like Relationship

A user can FOLLOW another user:

```
1 MATCH (u:User {id: $userId}), (f:User {id: $followedUserId})
2 WHERE NOT (u)-[:FOLLOWS]->(f)
3 CREATE (u)-[r:FOLLOWS]->(f)
4 RETURN r
5
```

Listing 3.2: Create Follow Relationship

- Read: This operation will allow users to read nodes and relationships from the database. For example, users can read information about anime and manga and relationships between users and media content. A user can read the list of liked media contents:

```
1
2 MATCH (u:User {id: $userId})-[:LIKE]->(a:Anime)
3 RETURN a
4
```

Listing 3.3: Read Liked Media Contents

A user can read the list of followers:

```
1 MATCH (u:User {id: $userId})<-[:FOLLOWS]-(f:User)
2 RETURN f
3
```

Listing 3.4: Read Followers

- Update: This operation will allow users to update nodes and relationships in the database. For example, users can update their likes for anime and manga and relationships between users.
- Delete: This operation will allow users to delete nodes and relationships from the database. For example, users can delete their likes for anime and manga and relationships between users.

A user can unlike a media content:

```
1 MATCH (u:User {id: $userId})-[:LIKE]->(a:Anime {id: $animeId})
2 DELETE r
3 RETURN r
4
```

Listing 3.5: Delete Like Relationship

A user can unfollow another user:

```
1  MATCH (:User {id: $followerUserId})-[r:FOLLOWS]->(:User {id: $followingUserId})
2  DELETE r
3  RETURN r
4
```

Listing 3.6: Delete Follow Relationship

Availability and Partition Tolerance

MangaVerse, as a social network, gives priority to the AP configuration of the CAP theorem, ensuring Availability and Partition Tolerance. This allows users to access the application and interact with other users and media content, even if the data is not always consistent (Eventual Consistency).

Redundancy

The performance of the application is critical, so we need to ensure that the system is highly available and fault-tolerant. To achieve this, we gave priority to fast responses, rather than reducing memory consumption.

Latest reviews

In the anime and manga collections, there's a field containing the latest 5 reviews written for that specific media content, in this way it's fast to retrieve.

Average rating

In the anime and manga collections, there's a field containing the average rating of the media content, this field is updated every time a new review is written.

Number of likes

In the anime and manga collections, there's a field containing the number of likes, this field is updated every time a new like relationship is created or deleted.

Followers and Followings

In the user collection, there are fields containing the number of followers and followings, this field is updated every time a new follow relationship is created or deleted.

User field in Reviews

In the reviews collection, there's a field containing the user data, such as id, username, picture, and also location and birthday, which are used for suggestion purposes.

Review Ids A list of review ids is stored in the anime, manga and users collections, this is used to quickly retrieve the reviews of a media content and of a user.

Replicas

A cluster of three nodes is available for this project, allowing deployment of replicas: however, replicas were only implemented in MongoDB, as Neo4j required the Enterprise version for it. We have 3 replicas for MongoDB and 1 for Neo4J. In MongoDB we have one primary and two secondary replicas, the primary is used for write operations and the secondaries are used for read operations. This will allow us to distribute the load and improve the performance of the application. In case of failure of the primary node, one of the secondary nodes will be promoted to primary, ensuring high availability of the system.

Sharding

Sharding is a method for distributing data across multiple machines to meet the demands of data growth. As the size of the data increases, a single machine may not be sufficient to store the data nor provide an acceptable read and write throughput. Sharding solves the problem with horizontal scaling. Even if not implemented, the database design is ready for sharding, as the data is distributed in a way that allows for easy sharding. The user, anime, manga, and reviews collections are sharded by the user id, anime id, manga id, and review id, respectively. This will allow us to distribute the data across multiple machines and improve the performance of the application.

Implementation

Development Environment

To ensure efficient and successful Implementation of MangaVerse web application, choosing the appropriate development environment is one of the most important points of the project.

Programming Languages

- **Backend:** Java is the main programming language used in the project's backend development.
- **Frontend:** HTML, CSS, JavaScript are utilized for building user interface in the project.
- **Data Preprocessing:** Python and java were used in the project to conduct data preprocessing task with the help of its powerful libraries and ease of use features.

Database

- **Document Database:** MongoDB is used in the project to store and manage document-based data with the help of its flexibility and scalability features.
- **Graph Database:** Neo4j is used in the project to manage and query graph data and handle complex relationships and connections between user entities and media contents in an efficient way.

Integrated Development Environment

IntelliJ IDEA was used as an primary IDE. It is powerful Java integrated development environment for developing software in an efficient way.

Version Control

Github was used to provide a collaborative development with its version control system.

Web Server

Apache Tomcat was used as a web server to provide reliable environment for deploying and running the java based web application.

Build Automation

Maven was used as a build automation tool. It is used to manage the project's build, reporting, and documentation from a central piece of information.

Testing

JUnit was used as a testing framework for Java code. It is used to write and run repeatable automated tests. This ensures the reliability and efficiency of the codebase throughout the development process.

Main Modules

- Configuration
- Controller
- DAO (Data Access Objects)

- DTO (Data Transfer Objects)
- Model
- Service
- Utils
- User Interface

Configuration

Configuration module contains a class named *AppServletContextListener* which is responsible for initializing and managing database connections for the web application. The configuration class implements *ServletContextListener* interface. *@WebListener* annotation is used to provide listening for application lifecycle events. This annotation contains two methods, which are *contextInitialized(ServletContextEvent sce)* and *contextDestroyed(ServletContextEvent sce)*. The first method is called when the web application is started and the second method is called when the web application is shut down.

Database Connection Management: Database connection is provided with *openConnection()* and *closeConnection()* methods. They are both initialized for managing connection for MongoDB and Neo4j databases. Connections are managed with corresponding DAO classes which are *BaseMongoDBDAO* and *BaseNeo4jDAO*.

With using the configuration module for database connection, web application ensures robustness and reliability in its data access layer.

Controller

The controller modules plays a role as intermediary between the user requests and backend of the MangaVerse web application as servlet classes. They receives the user requests, process them and returns with the corresponding response. The controller classes are implemented using *HttpServlet* to handle user requests and responses. Within the scope of their intermediary role, the controller classes are responsible of being a bridge between the user interface and backend logic. When a user interacts with the web application, their actions are translated into a HTTP request and these requests are handled by the related servlet class in the controller module. To be able to do request translation in an efficient way each controller class extend '*HttpServlet*' and has various methods to handle HTTP requests like GET and POST. Each controller class utilized a switch-case structure to determine the action requested and invokes the appropriate handler method accordingly. This structure allows for clear and organized routing of request to their corresponding handler method. After processing the request, the servlet generates a requested response.

Example code snippet from *MediaContentServlet*:

```

1 {
2     protected void processRequest(HttpServletRequest request, HttpServletResponse response
3         ) throws ServletException, IOException {
4         String action = request.getParameter("action");
5
6         switch (action) {
7             case "toggleLike" -> handleToggleLike(request, response);
8             case "addReview" -> handleAddReview(request, response);
9             case "deleteReview" -> handleDeleteReview(request, response);
10            case "editReview" -> handleEditReview(request, response);
11            case "getMediaContent" -> handleGetMediaContentById(request, response);
12            case "getMediaContentByTitle" -> handleSearchMediaContentByTitle(request,
13                response);
14            case null, default -> handleLoadPage(request, response);
15        }
16    }
17 }

```

The controller module contains the following classes:

- Exception
NotAuthorizedException: This exception is thrown when the user is not authorized to access the requested resource.
- AuthServlet
The AuthServlet class handles the user authentication and authorization processes. It includes login, logout and sign up functions
- MainPageServlet
The MainPageServlet class is responsible for handling the main page of the web application. It includes the main page of the web application and the search functionality. It provides request related to displaying main page and searching media contents.
- ManagerServlet
The ManagerServlet class manages administrative requests in manager page. These request are primarily about manga, anime and user analytics such *averageRatingByMonth()*, *trendMediaContentByYear()*, *getBestCriteria()*...
- MediaContentServlet
The MediaContentServlet class is responsible for managing request related with media contents. These requests include like, adding, deleting or editing reviews and retrieving media content details.
- ProfileServlet
The ProfileServlet class is responsible for managing user profile related requests. These requests include updating user profile, following/unfollowing other users, getting user profile details such as liked anime and manga and user reviews.
- UserServlet
The UserServlet class is responsible for managing user related requests and interactions. These requests include retrieving followers list, following list and user information.

DAO (Data Access Objects)

The DAO module includes the logic for accessing and managing data in the database and provides data retrieval, storage and manipulation. This module includes classes with CRUD (create, read, update, delete) operations and query executions. It provides a layer of abstraction between the database and the rest of the application and ensures the separation of concerns. The DAO module contains the following classes:

- Enums
 - DataRepositoryEnum
- Exceptions
- Interfaces
 - MediaContentDAO
 - ReviewDAO
 - UserDAO
- Mongo
 - AnimeDAOMongoImpl
 - BaseMongoDBDAO
 - MangaDAOMongoImpl
 - ReviewDAOMongoImpl
 - UserDAOMongoImpl
- Neo4j
 - AnimeDAONeo4jImpl
 - BaseNeo4jDAO
 - MangaDAONeo4jImpl
 - UserDAONeo4jImpl
- DAOLocator

Example code snippet from MangaDAOMongoImpl:

```
1 {
2     //MongoDB queries
3     //Best genres/themes/demographics/authors based on the average rating
4     @Override
5     public Map<String, Double> getBestCriteria (String criteria, boolean isArray, int page
6     ) throws DAOException {
7         try {
8             MongoClient<Document> mangaCollection = getCollection(COLLECTION_NAME);
9             int pageOffset = (page-1)*Constants.PAGE_SIZE;
10
11             List<Bson> pipeline;
12             if (isArray) {
13                 pipeline = List.of(
14                     match(and(exists(criteria), ne("average_rating", null))),
15                     unwind("$" + criteria),
16                     group("$" + criteria, avg("criteria_average_rating", "
17 $average_rating")),
18                     sort(descending("criteria_average_rating")),
19                     skip(pageOffset),
20                     limit(25)
21                 );
22             } else {
23                 pipeline = List.of(
24                     match(Filters.exists(criteria)),
25                     group("$" + criteria, avg("criteria_average_rating", "
26 $average_rating")),
27                     sort(new Document("criteria_average_rating", -1)),
28                     skip(pageOffset),
29                     limit(25)
30                 );
31             }
32
33             List<Document> document = mangaCollection.aggregate(pipeline).into(new
34             ArrayList<>());
35             Map<String, Double> bestCriteria = new LinkedHashMap<>();
36             for (Document doc : document) {
37                 Double avgRating = doc.get("criteria_average_rating") instanceof Integer?
38                 doc.getInteger("criteria_average_rating").doubleValue() :
39                 doc.getDouble("criteria_average_rating");
40                 if (criteria.equals("authors")) {
41                     bestCriteria.put(doc.get("_id", Document.class).getString("name"),
42                     avgRating);
43                 } else {
44                     bestCriteria.put(doc.get("_id").toString(), avgRating);
45                 }
46             }
47
48             return bestCriteria;
49         } catch (Exception e) {
50             throw new DAOException(DAOExceptionType.GENERIC_ERROR, e.getMessage());
51         }
52     }
53 }
```

Example code snippet from UserDAONeo4jImpl:

```
1 {
2     /**
3      * Retrieves a list of users following a specific user from the Neo4j database.
4      *
5      * @param userId The ID of the user whose followers are to be retrieved.
6      * @param loggedUserId The ID of the user requesting the list of followers.
7      * @return A list of RegisteredUserDTO objects representing the followers of the
8      *         specified user.
9      * @throws DAOException If an error occurs while retrieving the followers list.
10     */
11     @Override
12     public List<UserSummaryDTO> getFirstNFollowers(String userId, String loggedUserId)
13         throws DAOException {
14         try (Session session = getSession()) {
15             StringBuilder queryBuilder = new StringBuilder("MATCH (follower:User)-[:FOLLOWS
16 ]->(:User {id: $userId}) ");
17             if (loggedUserId != null) {
18                 queryBuilder.append("WHERE follower.id <> $loggedUserId ");
19             }
20             queryBuilder.append("RETURN follower AS user ");
21             queryBuilder.append("ORDER BY follower.username ");
22             queryBuilder.append("LIMIT 10");
23             String query = queryBuilder.toString();
24
25             Map<String, Object> params = new HashMap<>();
26             params.put("userId", userId);
27             if (loggedUserId != null) {
28                 params.put("loggedUserId", loggedUserId);
29             }
30
31             List<Record> records = session.executeRead(
32                 tx -> tx.run(query, params).list()
33             );
34
35             return records.isEmpty() ? null : records.stream()
36                 .map(this::recordToUserSummaryDTO)
37                 .toList();
38
39         } catch (Neo4jException e) {
40             throw new DAOException(DAOExceptionType.DATABASE_ERROR, e.getMessage());
41
42         } catch (Exception e) {
43             throw new DAOException(DAOExceptionType.GENERIC_ERROR, e.getMessage());
44         }
45     }
46 }
```

DTO (Data Transfer Objects)

The DTO modules are the intermediary class between presentation layer and the DAO module in the web application. They transfer data structures between different layers and components of the application in a more standardized way.

Model

- Enums
- Media Content
 - Anime
 - Manga
 - Manga Author
 - Media Content
- Registered User
 - Mangager
 - Registered User
 - User
- Review

Service

Service module has also important role in the web application. The classes in the service module are responsible for containing the business logic and maintaining interaction between the DAO classes and the presentation layer. It handles complex operations with guarantying that the application's core functionalities are executed correctly. Some of the services that are provided in the service module are: *UserService*, *MediaContentService*, *ReviewService*, *TaskManager*, *ExecuterTaskService*. The package structure of Service module is as follows:

- enums
 - ExecuterTaskService
- exceptions
 - enums
 - BusinessExceptionType
 - BusinessException
- impl
 - async media tasks
 - CreateMediaTask
 - DeleteMediaTask
 - UpdateAverageRatingTask
 - UpdateMediaRedundancyTask
 - UpdateMediaTask
 - UpdateNumberOfLikesTask
 - async review tasks
 - RemoveDeletedMediaReviewsTask
 - RemoveDeletedUserReviewsTask
 - UpdateReviewRedundancyTask
 - async user tasks
 - CreateUserTask
 - DeleteUserTask
 - UpdateNumberOfFollowedTask
 - UpdateNumberOfFollowersTask
 - UpdateUserTask
 - AperiodicExecuterTaskServiceImpl
 - ErrorTaskManager
 - MediaContentServiceImpl

- PeriodicExecutorTaskServiceImpl
- ReviewServiceImpl
- UserServiceImpl
- interfaces
 - ExecuterTaskService
 - MediaContentService
 - ReviewService
 - Task
 - TaskManager
 - UserService
- ServiceLocator

Adopted Patterns and Techniques

Patterns

Techniques

Task Manager:

Task Manager class which is located in the service module of the system provides asynchronous task execution with using *PriorityBlockingQueue*. It helps to order the tasks according to their prioritizes. After that prioritization, it ensures that higher priority tasks will be executed first and if two tasks have the same priority the one which is created before will be executed first. While Task Manager class is able to start and stop the tasks within the functions inside, it can also take tasks to the queue in a thread-safe way. By using taskComparator for ordering the tasks, the system provides also effective scheduling and execution.

Aperiodic Executor Task Service:

Executor Task Service class which is located inside the service module of the system is an important part for providing the eventual consistency. Executing tasks in asynchronous way with threads guarantees eventual consistency across different collections, mongoDB and neo4j and different replicas. With the help of the Executor Task Service, tasks that are needed to be executed in an asynchronous way are handled by ensuring that changes propagate correctly across different part of the system. While using multiple databases and data replicas for this web application, it is important for maintain data integrity and eventual consistency. Executing the tasks in an asynchronous way using threads allows to perform operations without blocking the main execution flow. Aperiodic executor task service class is implemented by using the interface of executor service.

Description of Main Classes

Controller

| Class | Description |
|---------------------|---|
| AuthServlet | Handles business logic for authentication |
| MainPageServlet | Handles business logic for main page |
| ManagerServlet | Handles business logic for manager |
| MediaContentServlet | Handles business logic for media content |
| ProfileServlet | Handles business logic for profile |
| UserServlet | Handles business logic for user |

DAO

| Class | Sub-package | Description |
|--------------------|-------------|--|
| MediaContentDAO | interfaces | Collection of methods for media content database related entities on mongoDB |
| ReviewDAO B | interfaces | Collection of methods for review database related entities on mongoDB |
| UserDAO | interfaces | Collection of methods for user database related entities on mongoDB |
| AnimeDAOMongoImpl | mongo | Contains all the method implementation for the MongoDB database anime entities |
| BaseMongoDBDAO | mongo | Contains all the method implementations for the MongoDB database |
| MangaDAOMongoImpl | mongo | Contains all the method implementations for the MongoDB database manga entities |
| ReviewDAOMongoImpl | mongo | Contains all the method implementations for the MongoDB database review entities |
| UserDAOMongoImpl | mongo | Contains all the method implementations for the MongoDB database user entities |
| AnimeDAONeo4jImpl | neo4j | Contains all the method implementation for the Neo4j database anime entities |
| BaseNeo4jDAO | neo4j | Contains all the method implementations for the Neo4j database |
| MangaDAONeo4jImpl | neo4j | Contains all the method implementation for the Neo4j database manga entities |
| UserDAONeo4jImpl | neo4j | Contains all the method implementation for the Neo4j database user entities |
| DAOLocator | | Implements the locator pattern for accessing DAOs based on the specified data repository |

DTO

| Class | Sub-package | Description |
|-----------------|--------------|--|
| AnimeDTO | mediaContent | Represents data transfer object containing attributes for animes |
| MangaDTO | mediaContent | Represents data transfer object containing attributes for mangas |
| MediaContentDTO | interfaces | Defines common attributes for media content |
| DashboardDTO | statistics | Contains statistical data for the dashboard |
| MongoDBStats | statistics | Provides statistics specific to MongoDB |
| LoggedUserDTO | | Holds information about a logged-in user. |
| PageDTO | | Represents pagination details |

| Class | Sub-package | Description |
|---------------------|-------------|--|
| ReviewDTO | | Contains attributes for reviews |
| UserRegistrationDTO | | Holds data for user registration |
| UserSummaryDTO | | Provides a summary of user information |

Model

| Class | Sub-package | Description |
|----------------|----------------|--|
| Anime | mediaContent | Provides unique anime attributes by extending parent class MediaContent and related getter and setter methods. |
| Manga | mediaContent | Provides unique manga attributes by extending parent class MediaContent and related getter and setter methods. |
| MangaAuthor | mediaContent | Contains manga author attributes and related getter and setter methods. |
| MediaContent | mediaContent | Contains all the attributes used by types of media contents and their getter and setter methods. |
| Manager | registeredUser | Provides unique manager attributes by extending parent class RegisteredUser and related getter and setter methods. |
| RegisteredUser | registeredUser | Contains all the attributes used by types of registered users and their getter and setter methods. |
| User | registeredUser | Provides unique user attributes by extending parent class RegisteredUser and related getter and setter methods. |
| Review | | Contains review attributes and related getter and setter methods. |

Service

| Class | Sub-package | Description |
|---------------------------|----------------------------|---|
| CreateMediaTask | impl/ asinc_media_tasks | Implementation of methods for media task creation for MediaContentService |
| DeleteMediaTask B | impl/ asinc_media_tasks | Implementation of methods for media task deletion for MediaContentService |
| RefreshLatestReviewsTasks | impl/ asinc_media_tasks | Implementation of methods for refreshing latest reviews for MediaContentService |
| UpdateAverageRatingTask | impl/ asinc_media_tasks | Implementation of methods for updating average rating for MediaContentService |
| UpdateMediaRedundancyTask | impl/ asinc_media_tasks | Implementation of methods for updating media redundancy for MediaContentService |
| UpdateMediaTask | impl/ asinc_media_tasks | Implementation of methods for updating media for MediaContentService |
| UpdateNumberOfLikesTask | impl/ asinc_media_tasks | Implementation of methods for updating numbers of likes for MediaContentService |

| Class | Sub-package | Description |
|----------------------------------|-----------------------------|---|
| RemoveDeletedMediaReviewsTask | impl/ asinc_review_tasks | Implementation of methods for removing reviews of deleted media for ReviewService |
| RemoveDeletedUserReviewsTask | impl/ asinc_review_tasks | Implementation of methods for removing reviews of deleted user for ReviewService |
| UpdateReviewRedundancyTask | impl/ asinc_review_tasks | Implementation of methods for updating review redundancy for ReviewService |
| CreateUserTask | impl/ asinc_user_tasks | Implementation of methods for user creation for UserService |
| DeleteUserTask | impl/ asinc_user_tasks | Implementation of methods for user deletion for UserService |
| UpdateNumberOfFollowedTaskB | impl/ asinc_user_tasks | Implementation of methods for updating number of followed for UserService |
| UpdateNumberOfFollowersTask | impl/ asinc_user_tasks | Implementation of methods for updating number of followers for UserService |
| UpdateUserTask | impl/ asinc_user_tasks | Implementation of methods for updating user for MediaContentService |
| AperiodicExecutorTaskServiceImpl | impl | Implementation of aperiodic tasks for ExecutorTaskService |
| ErrorTaskManager | impl | Implementation of TaskManager interface to handle error |
| MediaContentServiceImpl | impl | Implementation of MediaContentService, providing media content operations |
| PeriodicExecutorTaskServiceImpl | impl | Implementation of periodic tasks for ExecutorTaskService |
| ReviewServiceImpl | impl | Implementation of ReviewService, providing review operations |
| UserServiceImpl | impl | Implementation of UserService, providing user operations |
| ExecutorTaskService | interfaces | Collection of methods for task management |
| MediaContentService | interfaces | Collection of methods for media content service |
| ReviewService | interfaces | Collection of methods for review service |
| Task | interfaces | Collection of methods for execution operations |
| TaskManager | interfaces | Collection of methods for managing task prioritization |
| UserService | interfaces | Collection of methods for user service |
| ServiceLocator | | Implements locator pattern for services |

MongoDB queries

Some of the most important MongoDB queries for analytic and suggestion purposes.

USER:

Get Distribution

GetDistribution query to get the user's location, birthday year that gave the highest rating to the application

- Java Implementation:

```
1 public Map<String, Integer> getDistribution(String criteria) throws DAOException {
2     try {
3         MongoClient
```

- Mongo Shell Query:

```
1 db.collection.aggregate([
2     {
3         // Match stage to filter documents where 'criteriaOfSearch' exists
4         $match: {
5             [criteriaOfSearch]: { $exists: true }
6         }
7     },
8     // Project stage to include 'criteriaOfSearch' and 'app_rating' fields
9     {
10         $project: {
```

```

11         [criteriaOfSearch]: 1,
12         app_rating: 1
13     }
14 },
15 // Group stage to count occurrences of each 'criteriaOfSearch'
16 {
17     $group: {
18         _id: "$" + criteriaOfSearch,
19         count: { $sum: 1 }
20     }
21 },
22 // Sort stage to sort documents by 'count' in descending order
23 {
24     $sort: {
25         count: -1
26     }
27 }
28 ]));

```

Average App Rating

Calculates the average application rating based on the specified search criteria

- Java Implementation:

```

1 public Map<String, Double> averageAppRating(String criteria) throws DAOException {
2     try {
3         MongoCollection<Document> usersCollection = getCollection(COLLECTION_NAME);
4
5         List<Bson> pipeline = List.of(
6             match(and(exists(criteria), exists("app_rating"))),
7             group("$" + criteria, avg("averageAppRating", "$app_rating")),
8             sort(descending("averageAppRating"))
9         );
10
11         List<Document> aggregationResult = usersCollection.aggregate(pipeline).into(new
12             ArrayList<>());
13         if (aggregationResult.isEmpty()) {
14             throw new MongoException("UserDAOMongoImpl: averageAppRating: No data found");
15         }
16
17         Map<String, Double> map = new LinkedHashMap<>();
18         for (Document doc : aggregationResult) {
19             map.put(doc.getString("_id"), doc.getDouble("averageAppRating"));
20         }
21         return map;
22     } catch (MongoException e) {
23         throw new DAOException(DAOExceptionType.DATABASE_ERROR, e.getMessage());
24     }
25     catch (Exception e) {
26         throw new DAOException(DAOExceptionType.GENERIC_ERROR, e.getMessage());
27     }
28 }

```

- Mongo Shell Query:

```

1 db.getCollection.aggregate([
2     {
3         // Match stage: Filters documents to include only those that
4         // have both the specified 'criteria' field and 'app_rating' field.
5         $match: {
6             $and: [
7                 { [criteria]: { $exists: true } },
8                 { app_rating: { $exists: true } }
9             ]
10        }
11    },
12    {
13        // Group stage: Groups the filtered documents by the 'criteria' field.
14        // Calculates the average value of 'app_rating' for each group.
15        $group: {

```

```

16         _id: "$" + criteria,
17         averageAppRating: { $avg: "$app_rating" }
18     },
19 },
20 {
21     // Sort stage: Sorts the groups in descending order by 'averageAppRating'.
22     $sort: {
23         averageAppRating: -1
24     }
25 }
26 ]).toArray();

```

Average App Rating By Age

Calculates the average app rating for users grouped by age ranges. The age ranges are defined as follows:

- 0-13 years
- 13-20 years
- 20-30 years
- 30-40 years
- 40-50 years
- 50+ years

- Java Implementation:

```

1 public Map<String, Double> averageAppRatingByAgeRange() throws DAOException {
2     try {
3         MongoCollection<Document> usersCollection = getCollection(COLLECTION_NAME);
4
5         // Define the boundaries for the age ranges and the output fields
6         List<Long> boundaries = Arrays.asList(0L, 13L, 20L, 30L, 40L, 50L);
7         BsonField[] outputFields = {
8             new BsonField("avg_app_rating", new Document("$avg", "$app_rating"))
9         };
10        BucketOptions options = new BucketOptions()
11            .defaultBucket(50L)
12            .output(outputFields);
13
14        List<Bson> pipeline = List.of(
15            match(and(exists("birthday"), exists("app_rating"))),
16            project(fields(
17                computed("age", new Document("$floor", new Document("$divide",
18                    Arrays.asList(
19                        new Document("$subtract", Arrays.asList(new Date(), "$birthday
20                        ")),
21                        1000L * 60 * 60 * 24 * 365
22                    )))),
23                include("app_rating")
24            )),
25            bucket("$age", boundaries, options)
26        );
27
28        List<Document> aggregationResult = usersCollection.aggregate(pipeline).into(new
29        ArrayList<>());
30
31        if (aggregationResult.isEmpty()) {
32            throw new MongoException("UserDAOMongoImpl: averageAppRatingByAgeRange: No data
33            found");
34        }
35
36        Map<String, Double> map = new LinkedHashMap<>();
37        for (Document doc : aggregationResult) {
38            String ageRange = convertIntegerToAgeRange(doc.getLong("_id"));
39            map.put(ageRange, doc.getDouble("avg_app_rating"));
40        }
41
42        return map;
43    } catch (MongoException e) {
44        throw new DAOException(DAOExceptionType.DATABASE_ERROR, e.getMessage());
45    } catch (Exception e) {
46        throw new DAOException(DAOExceptionType.GENERIC_ERROR, e.getMessage());
47    }
48 }

```

- Mongo Shell Query:

```

1 db.getCollection('COLLECTION_NAME').aggregate([
2     {
3         // Match stage: Filters documents to include only those that
4         // have both the 'birthday' field and 'app_rating' field.
5         $match: {
6             $and: [
7                 { birthday: { $exists: true } },
8                 { app_rating: { $exists: true } }
9             ]
10        },
11    },
12    {
13        // Project stage: Adds a new field 'age' calculated by subtracting
14        // the 'birthday' from the current date, converting the difference
15        // from milliseconds to years, and taking the floor of the result.
16        // Also includes the 'app_rating' field.
17        $project: {
18            age: {
19                $floor: {
20                    $divide: [
21                        { $subtract: [ new Date(), "$birthday" ] },
22                        1000 * 60 * 60 * 24 * 365
23                    ]
24                },
25            },
26            app_rating: 1
27        },
28    },
29    {
30        // Bucket stage: Groups the documents into buckets based on the 'age' field.
31        // Specifies boundaries for the buckets and assigns documents with an age
32        // outside these boundaries to the default bucket (50+ years).
33        // For each bucket, calculates the average value of 'app_rating'.
34        $bucket: {
35            groupBy: "$age",
36            boundaries: [0, 13, 20, 30, 40, 50],
37            default: 50,
38            output: {
39                avg_app_rating: { $avg: "$app_rating" }
40            }
41        },
42    }
43 ]).toArray();

```

REVIEW:

Get Media Content Rating By Year

Retrieves the average ratings for a specific media content (anime or manga) by year within a specified range. The aggregation pipeline performs the following steps:

1. Matches the reviews for the specified media content ID and date range, ensuring the reviews have a rating.
2. Groups the reviews by year and calculates the average rating for each year.
3. Projects the results to include the year and the calculated average rating.
4. Sorts the results by year in ascending order.

- Java Implementation:

```

1 public Map<String, Double> getMediaContentRatingByYear(MediaContentType type, String
    mediaContentId, int startYear, int endYear) throws DAOException {
2     try {
3         // Get media content rating by year
4         MongoCollection<Document> reviewCollection = getCollection(COLLECTION_NAME);
5
6         String nodeType = type.equals(MediaContentType.ANIME) ? "anime" : "manga";
7         Date startDate = ConverterUtils.localDateToDate(LocalDate.of(startYear, 1, 1));
8         Date endDate = ConverterUtils.localDateToDate(LocalDate.of(endYear + 1, 1, 1));
9         List<Bson> pipeline = List.of(

```

```

10         match(and(
11             eq(nodeType + ".id", new ObjectId(mediaContentId)),
12             exists("rating", true),
13             gte("date", startDate),
14             lt("date", endDate)
15         )),
16         group(new Document("$year", "$date"), avg("average_rating", "$rating")),
17         project(fields(
18             excludeId(),
19             computed("year", "$_id"),
20             include("average_rating")
21         )),
22         sort(ascending("year"))
23     );
24     List<Document> result = reviewCollection.aggregate(pipeline).into(new ArrayList<>());
25
26     // Initialize the result map with years and default values
27     Map<String, Double> resultMap = new LinkedHashMap<>();
28     for (int year = startYear; year <= endYear; year++) {
29         resultMap.put(String.valueOf(year), null);
30     }
31
32     // Populate the result map with the average ratings
33     for (Document document : result) {
34         Double averageRating = document.getDouble("average_rating");
35         Integer year = document.getInteger("year");
36         resultMap.put(String.valueOf(year), averageRating);
37     }
38     return resultMap;
39
40 } catch (MongoException e) {
41     throw new DAOException(DAOExceptionType.DATABASE_ERROR, e.getMessage());
42 } catch (Exception e) {
43     throw new DAOException(DAOExceptionType.GENERIC_ERROR, e.getMessage());
44 }
45 }

```

- Mongo Shell Query:

```

1 // Match stage to filter documents based on specified conditions
2 db.collection.aggregate([
3     {
4         // Filters documents to include only those where:
5         // 1. The nested 'id' field under 'nodeType' matches the specified 'mediaContentId'.
6         // 2. The 'rating' field exists.
7         // 3. The 'date' field is within the specified date range (startDate to endDate).
8         $match: {
9             [`${nodeType}.id`]: new ObjectId(mediaContentId),
10             rating: { $exists: true },
11             date: { $gte: startDate, $lt: endDate }
12         },
13     },
14     {
15         // Groups the filtered documents by year extracted from the 'date' field.
16         // Calculates the average value of 'rating' for each year.
17         $group: {
18             _id: { $year: "$date" },
19             average_rating: { $avg: "$rating" }
20         },
21     },
22     {
23         // Projects the result to include the 'year' and 'average_rating' fields,
24         // excluding the '_id' field.
25         $project: {
26             _id: 0,
27             year: "$_id",
28             average_rating: 1
29         },
30     },
31     {
32         // Sort stage to sort documents by year in ascending order
33         $sort: { year: 1 }
34     }
35 ])

```

```

34     }
35 });

```

Get Media Content Rating By Month

Retrieves the average ratings for a specific media content (anime or manga) by month for a specified year. The aggregation pipeline performs the following steps:

1. Matches the reviews for the specified media content ID and year, ensuring the reviews have a rating.
2. Groups the reviews by month and calculates the average rating for each month.
3. Projects the results to include the month and the calculated average rating.
4. Sorts the results by month in ascending order.

- Java Implementation:

```

1 public Map<String, Double> getMediaContentRatingByMonth(MediaContentType type, String
   mediaContentId, int year) throws DAOException {
2     try {
3         // Get media content rating by month
4         MongoCollection<Document> reviewCollection = getCollection(COLLECTION_NAME);
5
6         String nodeType = type.equals(MediaContentType.ANIME) ? "anime" : "manga";
7         Date startDate = ConverterUtils.localDateToDate(LocalDate.of(year, 1, 1));
8         Date endDate = ConverterUtils.localDateToDate(LocalDate.of(year + 1, 1, 1));
9         List<Bson> pipeline = List.of(
10             match(and(
11                 eq(nodeType + ".id", new ObjectId(mediaContentId)),
12                 exists("rating", true),
13                 gte("date", startDate),
14                 lt("date", endDate)
15             )),
16             group(new Document("$month", "$date"),
17                 avg("average_rating", "$rating")
18             ),
19             project(fields(
20                 excludeId(),
21                 computed("month", "$_id"),
22                 include("average_rating")
23             )),
24             sort(ascending("month"))
25         );
26         List<Document> result = reviewCollection.aggregate(pipeline).into(new ArrayList<>());
27
28         // Initialize the result map with months and default values
29         Map<String, Double> resultMap = new LinkedHashMap<>();
30         for (Month month : Month.values()) {
31             resultMap.put(month.getDisplayName(TextStyle.FULL, Locale.ENGLISH), null);
32         }
33
34         // Populate the result map with the average ratings
35         for (Document document : result) {
36             Object ratingObj = document.get("average_rating");
37             Double averageRating = ratingObj instanceof Integer ratingInt ? ratingInt.
doubleValue() : (Double) ratingObj;
38             Integer month = document.getInteger("month");
39             resultMap.put(Month.of(month).getDisplayName(TextStyle.FULL, Locale.ENGLISH),
averageRating);
40         }
41         return resultMap;
42     }
43     catch (MongoException e) {
44         throw new DAOException(DAOExceptionType.DATABASE_ERROR, e.getMessage());
45     }
46     catch (Exception e) {
47         throw new DAOException(DAOExceptionType.GENERIC_ERROR, e.getMessage());
48     }

```

- Mongo Shell Query:

```

1 db.getCollection.aggregate([
2     {
3         // Match stage: Filters documents to include only those that meet the specified
conditions.

```

```

4      $match: {
5          $and: [
6              // Includes documents where the nested 'id' field under 'nodeType' matches '
mediaContentId'.
7              { [nodeType + ".id"]: mediaContentId },
8              // Includes documents where the 'rating' field exists.
9              { rating: { $exists: true } },
10             // Includes documents where the 'date' field is greater than or equal to '
startDate'.
11             { date: { $gte: startDate } },
12             // Includes documents where the 'date' field is less than 'endDate'.
13             { date: { $lt: endDate } }
14         ]
15     },
16 },
17 {
18     // Group stage: Groups the filtered documents by the month extracted from the 'date'
field.
19     // Calculates the average value of 'rating' for each month.
20     $group: {
21         _id: { $month: "$date" },
22         average_rating: { $avg: "$rating" }
23     }
24 },
25 {
26     // Project stage: Shapes the output documents to include 'month' and 'average_rating'
fields.
27     // Excludes the '_id' field.
28     $project: {
29         _id: 0,
30         month: "$_id",
31         average_rating: 1
32     }
33 },
34 {
35     // Sort stage: Sorts the documents by 'month' in ascending order.
36     $sort: {
37         month: 1
38     }
39 }
40 ]).toArray();

```

Suggest Media Content

Suggests media content (anime or manga) based on user criteria (location or birthday year). The aggregation pipeline performs the following steps:

1. Matches the reviews with a rating, the specified media content type and the user criteria.
2. Groups the reviews by media content ID and calculates the average rating for each media content.
3. Projects the results to include the media content title and the calculated average rating.
4. Sorts the results by average rating in descending order.
5. Limits the results to 20 entries.

• Java Implementation:

```

1 public List<MediaContentDTO> suggestMediaContent(MediaContentType mediaContentType, String
criteriaType, String criteriaValue) throws DAOException {
2     try {
3         // Suggest media content based on user criteria
4         MongoCollection<Document> reviewCollection = getCollection(COLLECTION_NAME);
5         String nodeType = mediaContentType.equals(MediaContentType.ANIME) ? "anime" : "manga";
6
7         Bson filter = and(
8             exists("rating", true),
9             exists(nodeType, true)
10        );
11
12        if (criteriaType.equals("location")) {
13            filter = and(filter, eq("user.location", criteriaValue));
14        } else if (criteriaType.equals("birthday")) {
15            Date startDate = ConverterUtils.localDateToDate(LocalDate.of(Integer.parseInt(
criteriaValue), 1, 1));
16            Date endDate = ConverterUtils.localDateToDate(LocalDate.of(Integer.parseInt(
criteriaValue) + 1, 1, 1));

```



```

17         filter = and(filter, gte("user.birthday", startDate), lt("user.birthday", endDate)
18     );
19     } else {
20         throw new Exception("ReviewDAOMongoImpl: suggestMediaContent: Invalid criteria
21         type");
22     }
23
24     List<Bson> pipeline = new ArrayList<>(List.of(
25         match(filter),
26         group("$" + nodeType + ".id",
27             first("title", "$" + nodeType + ".title"),
28             avg("average_rating", "$rating")),
29         sort(descending("average_rating")),
30         project(include("title")),
31         limit(20));
32
33     List<Document> result = reviewCollection.aggregate(pipeline).into(new ArrayList<>());
34     if (result.isEmpty()) {
35         throw new MongoException("ReviewDAOMongoImpl: suggestMediaContent: No reviews
36         found");
37     }
38
39     List<MediaContentDTO> entries = new ArrayList<>();
40     for (Document document : result) {
41         String contentId = String.valueOf(document.getObjectId("_id"));
42         String title = document.getString("title");
43
44         MediaContentDTO mediaContentDTO;
45         if (nodeType.equals("anime")) {
46             mediaContentDTO = new AnimeDTO(contentId, title);
47         } else {
48             mediaContentDTO = new MangaDTO(contentId, title);
49         }
50         entries.add(mediaContentDTO);
51     }
52     return entries;
53 } catch (MongoException e) {
54     throw new DAOException(DAOExceptionType.DATABASE_ERROR, e.getMessage());
55 } catch (Exception e) {
56     throw new DAOException(DAOExceptionType.GENERIC_ERROR, e.getMessage());
57 }

```

- Mongo Shell Query:

```

1 db.collection.aggregate([
2     {
3         // Match documents based on a dynamic user criteria
4         // It dynamically matches documents where a field in the 'user' object
5         // (specified by 'criteriaType') has the value 'criteriaValue'.
6         $match: {
7             ["user." + criteriaType]: criteriaValue
8         }
9     },
10    {
11        // Group stage: Groups documents by the node type's ID.
12        // For each group, it retrieves the first title and calculates the average rating.
13        $group: {
14            _id: "$" + nodeType + ".id", // Group by the node type's ID
15            title: { $first: "$" + nodeType + ".title" }, // Get the first title in the group
16            average_rating: { $avg: "$rating" } // Calculate the average rating for the group
17        }
18    },
19    {
20        // Sort stage: Sorts the grouped documents by average rating in descending order.
21        $sort: { average_rating: -1 }
22    },
23    {
24        // Limit stage: Limits the number of results to a constant defined by 'Constants.
25        // PAGE_SIZE'.
26        $limit: Constants.PAGE_SIZE
27    }
28 ])

```

```

26     }
27     });

```

MANGA/ANIME:

Get Best Criteria

Retrieves the best criteria based on the average rating of the Anime objects in the MongoDB database.

- Java Implementation:

```

1 public Map<String, Double> getBestCriteria (String criteria, boolean isArray, int page) throws
   DAOException {
2     try {
3         MongoCollection<Document> animeCollection = getCollection(COLLECTION_NAME);
4         int pageOffset = (page - 1) * Constants.PAGE_SIZE;
5
6         List<Bson> pipeline;
7         if (isArray) {
8             pipeline = List.of(
9                 match(and(exists(criteria), ne("average_rating", null))),
10                unwind("$" + criteria),
11                group("$" + criteria, avg("criteria_average_rating", "$average_rating")),
12                sort(descending("criteria_average_rating")),
13                skip(pageOffset),
14                limit(25)
15            );
16        } else {
17            pipeline = List.of(
18                match(Filters.exists(criteria)),
19                group("$" + criteria, avg("criteria_average_rating", "$average_rating")),
20                sort(new Document("criteria_average_rating", -1)),
21                skip(pageOffset),
22                limit(25)
23            );
24        }
25
26        List<Document> document = animeCollection.aggregate(pipeline).into(new ArrayList<>());
27
28        Map<String, Double> bestCriteria = new LinkedHashMap<>();
29        for (Document doc : document) {
30            Double avgRating = doc.get("criteria_average_rating") instanceof Integer?
31                doc.getInteger("criteria_average_rating").doubleValue() :
32                doc.getDouble("criteria_average_rating");
33            bestCriteria.put(doc.get("_id").toString(), avgRating);
34        }
35        return bestCriteria;
36    } catch (Exception e) {
37        throw new DAOException(DAOExceptionType.GENERIC_ERROR, e.getMessage());
38    }
39 }
40 }

```

- Mongo Shell Query:

```

1 db.collection.aggregate([
2     // Match stage to filter documents where 'criteria' exists and 'average_rating' is not
   null
3     {
4         $match: {
5             criteria: { $exists: true },
6             average_rating: { $ne: null }
7         }
8     },
9     // Unwind stage to deconstruct the 'criteria' array field
10    {
11        $unwind: "$" + criteria
12    },
13    // Group stage to calculate the average rating for each criteria
14    {

```

```

15     $group: {
16         // Group by each individual 'criteria' element
17         _id: "$" + criteria,
18         // Calculate the average rating for each 'criteria'
19         criteria_average_rating: { $avg: "$average_rating" }
20     }
21 },
22 // Sort stage to sort documents by 'criteria_average_rating' in descending order
23 {
24     $sort: {
25         criteria_average_rating: -1
26     }
27 },
28 // Skip stage to skip the first 'pageOffset' documents
29 {
30     $skip: pageOffset
31 },
32 // Limit stage to limit the results to 25 documents
33 {
34     $limit: 25
35 }
36 ]);

```

GraphDB queries

Some of the most important Neo4j queries for analytic and suggestion purposes.

USERS:

Suggest User By Common Likes

Retrieves a list of suggested users for a specific user based on common likes from the Neo4j database. The method performs the following steps:

1. Retrieve users who like the same media content as the specified user in the last 6 month.
2. Retrieve users who like the same media content as the specified user in the last year.
3. Retrieve users who like the same media content as the specified user.

- Java Implementation:

```
1 public List<UserSummaryDTO> suggestUsersByCommonLikes(String userId, Integer limit,
2     MediaContentType type) throws DAOException {
3     try (Session session = getSession()) {
4         if (type == null) {
5             throw new IllegalArgumentException("Media content type must be specified");
6         }
7
8         int n = limit == null ? 5 : limit;
9         int remaining;
10
11         StringBuilder queryBuilder = new StringBuilder();
12         if (type == MediaContentType.ANIME)
13             queryBuilder.append("MATCH (u:User {id: $userId})-[r:LIKE]->(media:Anime)-[:LIKE
14 ]-(suggested:User) ");
15         else
16             queryBuilder.append("MATCH (u:User {id: $userId})-[r:LIKE]->(media:Manga)-[:LIKE
17 ]-(suggested:User) ");
18         queryBuilder.append("""
19             WHERE u <> suggested AND r.date >= date($date)
20             WITH suggested, COUNT(DISTINCT media) AS commonLikes
21             WHERE commonLikes > $min
22             RETURN suggested AS user, commonLikes
23             ORDER BY commonLikes DESC
24             LIMIT $n
25             """);
26         String query1 = queryBuilder.toString();
27         Value params1 = parameters("userId", userId, "n", n, "date", LocalDate.now().
28             minusMonths(6), "min", 5);
29
30         List<UserSummaryDTO> suggested = session.executeRead(
31             tx -> tx.run(query1, params1).list()
32         ).stream()
33             .map(this::recordToUserSummaryDTO)
34             .collect(Collectors.toList());
35
36         remaining = n - suggested.size();
37
38         if (remaining > 0) {
39             Value params2 = parameters("userId", userId, "n", n, "date", LocalDate.now().
40                 minusYears(1), "min", 5);
41
42             List<Record> records = session.executeRead(tx -> tx.run(query1, params2).list());
43             for (Record record : records) {
44                 UserSummaryDTO userDTO = recordToUserSummaryDTO(record);
45                 if (!suggested.contains(userDTO))
46                     suggested.add(userDTO);
47                 if (suggested.size() == n)
48                     break;
49             }
50
51             remaining = n - suggested.size();
52         }
53     }
```

```

50         if(remaining > 0) {
51             StringBuilder queryBuilder3 = new StringBuilder();
52             if (type == MediaContentType.ANIME)
53                 queryBuilder3.append("MATCH (u:User {id: $userId})-[r:LIKE]->(media:Anime)<-[:LIKE]-(suggested:User) ");
54             else
55                 queryBuilder3.append("MATCH (u:User {id: $userId})-[r:LIKE]->(media:Manga)<-[:LIKE]-(suggested:User) ");
56             queryBuilder3.append("""
57                 WHERE u <> suggested
58                 WITH suggested, COUNT(DISTINCT media) AS commonLikes
59                 RETURN suggested AS user, commonLikes
60                 ORDER BY commonLikes DESC
61                 LIMIT $n
62                 """);
63             String query2 = queryBuilder3.toString();
64             Value params3 = parameters("userId", userId, "n", n);
65
66             List<Record> records = session.executeRead(tx -> tx.run(query2, params3).list());
67             for (Record record : records) {
68                 UserSummaryDTO userDTO = recordToUserSummaryDTO(record);
69                 if (!suggested.contains(userDTO))
70                     suggested.add(userDTO);
71                 if (suggested.size() == n)
72                     break;
73             }
74         }
75
76         return suggested.isEmpty() ? null : suggested;
77
78     } catch (Neo4jException e) {
79         throw new DAOException(DAOExceptionType.DATABASE_ERROR, e.getMessage());
80
81     } catch (Exception e) {
82         throw new DAOException(DAOExceptionType.GENERIC_ERROR, e.getMessage());
83     }
84 }

```

- Neo4j Query:

```

1 // Match the user with the given userId who has liked media of type Manga
2 MATCH (u:User {id: $userId})-[r:LIKE]->(media:Manga)<-[:LIKE]-(suggested:User)
3 // Filter out the original user from the suggested users and only consider likes after the
  specified date
4 WHERE u <> suggested AND r.date >= $date
5 // Pass the suggested users and count of distinct media liked by both users to the next stage
6 WITH suggested, COUNT(DISTINCT media) AS commonLikes
7 // Filter out users with common likes less than or equal to the minimum threshold
8 WHERE commonLikes > $min
9 // Return the suggested users and the count of common likes
10 RETURN suggested AS user, commonLikes
11 // Order the results by the count of common likes in descending order
12 ORDER BY commonLikes DESC
13 // Limit the number of results to the specified maximum
14 LIMIT $n

```

Suggest Users By Common Followings

Retrieves a list of suggested users for a specific user based on common followings from the Neo4j database. The method performs the following steps:

1. Retrieve users that follow user's followings and have more than 5 common followings.
2. Retrieve users that are followed by user's followings and have more than 5 connections.
3. Retrieve users that follow user's followings.

- Java Implementation:

```

1 public List<UserSummaryDTO> suggestUsersByCommonFollowings(String userId, Integer limit)
  throws DAOException {
2     try (Session session = getSession()) {
3         int n = limit == null ? 5 : limit;

```

```

4         int remaining;
5
6         // suggest users that follow user's followings and have more than 5 common followings
7         String query = ""
8             MATCH (u:User {id: $userId})-[:FOLLOWS]->(following:User)<-[:FOLLOWS]-(
9             suggested:User)
10             WHERE NOT (u)-[:FOLLOWS]->(suggested) AND u <> suggested
11             WITH suggested, COUNT(DISTINCT following) AS commonFollowings
12             WHERE commonFollowings > 5
13             RETURN suggested as user
14             ORDER BY commonFollowings DESC
15             LIMIT $n
16             "";
17         Value params = parameters("userId", userId, "n", n);
18
19         List<UserSummaryDTO> suggested = session.executeRead(
20             tx -> tx.run(query, params).list()
21         ).stream()
22             .map(this::recordToUserSummaryDTO)
23             .collect(Collectors.toList());
24
25         remaining = n - suggested.size();
26
27         // if there are not enough suggestions, suggest users that are followed by the user's
28         // followings and have more than 5 connections
29         if (remaining > 0) {
30             String query2 = ""
31                 MATCH (u:User {id: $userId})-[:FOLLOWS]->(following:User)-[:FOLLOWS]->(
32                 suggested:User)
33                 WHERE NOT (u)-[:FOLLOWS]->(suggested) AND u <> suggested
34                 WITH suggested, COUNT(DISTINCT following) AS commonUsers
35                 WHERE commonUsers > 5
36                 RETURN suggested as user
37                 ORDER BY commonUsers DESC
38                 LIMIT $n
39                 "";
40             Value params2 = parameters("userId", userId, "n", n);
41
42             List<Record> records = session.executeRead(tx -> tx.run(query2, params2).list());
43             for (Record record : records) {
44                 UserSummaryDTO userDTO = recordToUserSummaryDTO(record);
45                 if (!suggested.contains(userDTO))
46                     suggested.add(userDTO);
47                 if (suggested.size() == n)
48                     break;
49             }
50
51             remaining = n - suggested.size();
52
53             // if there are still not enough suggestions, suggest users that follow the user's
54             // followings
55             if (remaining > 0) {
56                 String query3 = ""
57                     MATCH (u:User {id: $userId})-[:FOLLOWS]->(following:User)<-[:FOLLOWS]-(
58                     suggested:User)
59                     WHERE NOT (u)-[:FOLLOWS]->(suggested) AND u <> suggested
60                     WITH suggested, COUNT(DISTINCT following) AS commonFollowings
61                     RETURN suggested as user
62                     ORDER BY commonFollowings DESC
63                     LIMIT $n
64                     "";
65                 Value params3 = parameters("userId", userId, "n", n);
66
67                 List<Record> records = session.executeRead(tx -> tx.run(query3, params3).list());
68                 for (Record record : records) {
69                     UserSummaryDTO userDTO = recordToUserSummaryDTO(record);
70                     if (!suggested.contains(userDTO))
71                         suggested.add(userDTO);
72                     if (suggested.size() == n)
73                         break;
74                 }
75             }
76         }

```

```

72         return suggested.isEmpty() ? null : suggested;
73     } catch (Neo4jException e) {
74         throw new DAOException(DAOExceptionType.DATABASE_ERROR, e.getMessage());
75     } catch (Exception e) {
76         throw new DAOException(DAOExceptionType.GENERIC_ERROR, e.getMessage());
77     }
78 }
79 }
80 }
81 }

```

- Neo4j Query:

```

1 // Match the user with the given userId who follows other users
2 MATCH (u:User {id: $userId})-[:FOLLOWS]->(following:User)<-[:FOLLOWS]-(suggested:User)
3 // Ensure that the suggested user is not already followed by the original user and they are
  not the same user
4 WHERE NOT (u)-[:FOLLOWS]->(suggested) AND u <> suggested
5 // Calculate the count of distinct users that both the original user and suggested user follow
6 WITH suggested, COUNT(DISTINCT following) AS commonFollowers
7 // Filter out suggested users who have less than or equal to 5 common followers
8 WHERE commonFollowers > 5
9 // Return the suggested users along with the count of common followers
10 RETURN suggested as user, commonFollowers
11 // Order the results by the count of common followers in descending order
12 ORDER BY commonFollowers DESC
13 // Limit the number of results to the specified maximum
14 LIMIT $n

```

ANIME/MANGA:

Get Trend Media Content By Year

Retrieves a list of trending MangaDTO objects for a specific year from the Neo4j database.

- Java Implementation:

```

1 public Map<MediaContentDTO, Integer> getTrendMediaContentByYear(int year, Integer limit)
  throws DAOException {
2     int n = limit == null ? 5 : limit;
3     try (Session session = getSession()) {
4         LocalDate startDate = LocalDate.of(year, 1, 1);
5         LocalDate endDate = LocalDate.of(year + 1, 1, 1);
6
7         String query = """
8         MATCH (m:Manga)<-[:LIKE]-(u:User)
9         WHERE r.date >= date($startDate) AND r.date < date($endDate)
10        WITH m, count(r) AS numLikes
11        ORDER BY numLikes DESC
12        RETURN m AS manga, numLikes
13        LIMIT $n
14        """;
15
16        Value params = parameters("startDate", startDate, "endDate", endDate, "n", n);
17
18        Map<MediaContentDTO, Integer> result = new LinkedHashMap<>();
19        session.executeRead(
20            tx -> tx.run(query, params).list()
21        ).forEach(record -> {
22            MangaDTO mangaDTO = (MangaDTO) recordToMediaContentDTO(record);
23            Integer likes = record.get("numLikes").asInt();
24            result.put(mangaDTO, likes);
25        });
26
27        return result;
28    } catch (Neo4jException e) {
29        throw new DAOException(DAOExceptionType.DATABASE_ERROR, e.getMessage());
30    } catch (Exception e) {
31        throw new DAOException(DAOExceptionType.GENERIC_ERROR, e.getMessage());
32    }
33 }

```

```

34     }
35 }

```

- Neo4j Query:

```

1 // Match Anime nodes that are liked by Users, with a LIKE relationship and a date constraint
2 MATCH (a:Anime)-[r:LIKE]-(u:User)
3 WHERE r.date >= $startDate AND r.date < $endDate
4 // Aggregate the results to count the number of likes for each Anime node
5 WITH a, count(r) AS numLikes
6 // Order the Anime nodes by the number of likes in descending order
7 ORDER BY numLikes DESC
8 // Return the Anime node and the number of likes, limiting the results to the specified
   maximum
9 RETURN a AS anime, numLikes
10 LIMIT $n

```

Get Media Content Trend By Likes

Retrieves a list of trending MangaDTO objects by likes from the Neo4j database. The method performs the following steps:

1. Retrieve the trending Manga by likes in the last 6 months.
2. If there are not enough trending Manga, retrieve more results from the last year.
3. If there are still not enough trending Manga, retrieve more results from the last 5 years.

- Java Implementation:

```

1 public List<MediaContentDTO> getMediaContentTrendByLikes(Integer limit) throws DAOException {
2     try (Session session = getSession()) {
3         int n = limit == null ? 5 : limit;
4         int remaining;
5         LocalDate now = LocalDate.now();
6
7         // Try to get trending content based on likes in the last 6 months
8         String query1 = ""
9             MATCH (u:User)-[r:LIKE]->(m:Manga)
10            WHERE r.date >= date($startDate)
11            WITH m, COUNT(r) AS numLikes
12            WHERE numLikes > 10
13            RETURN m AS manga, numLikes
14            ORDER BY numLikes DESC, m.title ASC
15            LIMIT $n
16            "";
17
18         Value params1 = parameters("startDate", now.minusMonths(6), "n", n);
19         List<MediaContentDTO> trendingContent = session.executeRead(
20             tx -> tx.run(query1, params1).list()
21             ).stream()
22             .map(record -> (MangaDTO) recordToMediaContentDTO(record))
23             .collect(Collectors.toList());
24
25         remaining = n - trendingContent.size();
26
27         // If not enough results, add more results from the last year
28         if (remaining > 0) {
29             Value params2 = parameters("startDate", now.minusYears(1), "n", remaining);
30
31             List<Record> records = session.executeRead(tx -> tx.run(query1, params2).list());
32             for (Record record : records) {
33                 MangaDTO mangaDTO = (MangaDTO) recordToMediaContentDTO(record);
34                 if (!trendingContent.contains(mangaDTO))
35                     trendingContent.add(mangaDTO);
36                 if (trendingContent.size() == n)
37                     break;
38             }
39
40             remaining = n - trendingContent.size();
41         }
42
43         // If still not enough results, add more results from the last 5 years

```



```

44     if (remaining > 0) {
45         String query2 = """
46         MATCH (u:User)-[r:LIKE]->(m:Manga)
47         WHERE r.date >= date($startDate)
48         WITH m, COUNT(r) AS numLikes
49         RETURN m AS manga, numLikes
50         ORDER BY numLikes DESC, m.title ASC
51         LIMIT $n
52         """;
53         Value params3 = parameters("startDate", now.minusYears(5), "n", remaining);
54
55         List<Record> records = session.executeRead(tx -> tx.run(query2, params3).list());
56         for (Record record : records) {
57             MangaDTO mangaDTO = (MangaDTO) recordToMediaContentDTO(record);
58             if (!trendingContent.contains(mangaDTO))
59                 trendingContent.add(mangaDTO);
60             if (trendingContent.size() == n)
61                 break;
62         }
63     }
64
65     return trendingContent.isEmpty() ? null : trendingContent;
66
67 } catch (Neo4jException e) {
68     throw new DAOException(DAOExceptionType.DATABASE_ERROR, e.getMessage());
69
70 } catch (Exception e) {
71     throw new DAOException(DAOExceptionType.GENERIC_ERROR, e.getMessage());
72 }
73 }

```

- Neo4j Query:

```

1 // Match User nodes who have liked Anime nodes with a LIKE relationship and date constraint
2 MATCH (u:User)-[r:LIKE]->(a:Anime)
3 WHERE r.date >= $startDate
4 // Aggregate the results to count the number of LIKE relationships for each Anime node
5 WITH a, COUNT(r) AS numLikes
6 // Order the Anime nodes by the number of likes in descending order
7 ORDER BY numLikes DESC
8 // Return the Anime node and the number of likes, limiting the results to the specified
   maximum
9 RETURN a AS anime, numLikes
10 LIMIT $n

```

Get Suggested By Followings

Retrieves a list of suggested MangaDTO objects for a user from the Neo4j database. The method performs the following steps:

1. Retrieve Manga that the user's followings have liked in the last 6 months.
2. If there are not enough suggestions, retrieve Manga that the user's followings have liked in the last 2 years.
3. If there are still not enough suggestions, retrieve Manga that the user's followings have liked.

- Java Implementation:

```

1 public List<MediaContentDTO> getSuggestedByFollowings(String userId, Integer limit) throws
   DAOException {
2     try (Session session = getSession()) {
3         int n = limit == null ? 5 : limit;
4         int remaining;
5         LocalDate now = LocalDate.now();
6
7         // try to get suggestions based on likes in the last 6 months
8         String query1 = """
9         MATCH (u:User {id: $userId})-[:FOLLOWS]->(f:User)-[r:LIKE]->(m:Manga)
10        WHERE NOT (u)-[:LIKE]->(m) AND r.date >= date($startDate)
11        WITH m, COUNT(DISTINCT f) AS num_likes
12        RETURN m AS manga
13        ORDER BY num_likes DESC, m.title ASC
14        LIMIT $n

```

```

15         """;
16         Value params1 = parameters("userId", userId, "n", n, "startDate", now.minusMonths(6));
17
18         List<MediaContentDTO> suggested = session.executeRead(
19             tx -> tx.run(query1, params1).list()
20             ).stream()
21             .map(record -> (MangaDTO) recordToMediaContentDTO(record))
22             .collect(Collectors.toList());
23
24         remaining = n - suggested.size();
25
26         // if there are not enough suggestions, add more results from the last 2 years
27         if (remaining > 0) {
28             Value params2 = parameters("userId", userId, "n", n, "startDate", now.minusYears
29 (2));
30
31             List<Record> records = session.executeRead(tx -> tx.run(query1, params2).list());
32             for (Record record : records) {
33                 MangaDTO mangaDTO = (MangaDTO) recordToMediaContentDTO(record);
34                 if (!suggested.contains(mangaDTO))
35                     suggested.add(mangaDTO);
36                 if (suggested.size() == n)
37                     break;
38             }
39
40             remaining = n - suggested.size();
41         }
42
43         // if there are still not enough suggestions, add more results based on all likes
44         if (remaining > 0) {
45             String query2 = """
46                 MATCH (u:User {id: $userId})-[:FOLLOWS]->(f:User)-[r:LIKE]->(m:Manga)
47                 WHERE NOT (u)-[:LIKE]->(m)
48                 WITH m, COUNT(DISTINCT f) AS num_likes
49                 RETURN m AS manga
50                 ORDER BY num_likes DESC, m.title ASC
51                 LIMIT $n
52             """;
53             Value params3 = parameters("userId", userId, "n", n);
54
55             List<Record> records = session.executeRead(tx -> tx.run(query2, params3).list());
56             for (Record record : records) {
57                 MangaDTO mangaDTO = (MangaDTO) recordToMediaContentDTO(record);
58                 if (!suggested.contains(mangaDTO))
59                     suggested.add(mangaDTO);
60                 if (suggested.size() == n)
61                     break;
62             }
63
64             return suggested.isEmpty() ? null : suggested;
65
66         } catch (Neo4jException e) {
67             throw new DAOException(DAOExceptionType.DATABASE_ERROR, e.getMessage());
68
69         } catch (Exception e) {
70             throw new DAOException(DAOExceptionType.GENERIC_ERROR, e.getMessage());
71         }
72     }

```

• Neo4j Query:

```

1 // Match the User node with the specified userId who follows other User nodes,
2 // and those followed Users who have liked Anime nodes with a LIKE relationship and date
3 // constraint.
4 MATCH (u:User {id: $userId})-[:FOLLOWS]->(f:User)-[r:LIKE]->(a:Anime)
5 // Ensure that the User does not directly like the Anime and that the LIKE relationship's date
6 // is within the specified range.
7 WHERE NOT (u)-[:LIKE]->(a) AND r.date >= $startDate
8 // Aggregate the results to count the number of distinct Users who have liked each Anime node.
9 WITH a, COUNT(DISTINCT f) AS num_likes
10 // Return the Anime node, ordered by the number of likes in descending order.

```

```

9 RETURN a AS anime
10 ORDER BY num_likes DESC
11 // Limit the number of results returned to the specified maximum.
12 LIMIT $n

```

Get Suggested By Likes

Retrieves a list of suggested MangaDTO objects for a user from the Neo4j database. The method performs the following steps:

1. Retrieve Manga that other users with similar taste have liked in the last 6 months.
2. If there are not enough suggestions, retrieve Manga that other users with similar taste have liked in the last 2 years.
3. If there are still not enough suggestions, retrieve Manga that other users with similar taste have liked.

• Java Implementation:

```

1 public List<MediaContentDTO> getSuggestedByLikes(String userId, Integer limit) throws
   DAOException {
2     try (Session session = getSession()) {
3         int n = limit == null ? 5 : limit;
4         int remaining;
5         LocalDate today = LocalDate.now();
6
7         // Try to get suggestions based on likes in the last 6 months
8         String query1 = """
9             MATCH (u:User {id: $userId})-[r1:LIKE]->(m:Manga)<-[:LIKE]-(f:User)
10            WHERE r1.date >= $startDate
11            WITH u, f, COUNT(m) AS common_likes
12            ORDER BY common_likes DESC
13            LIMIT 20
14            MATCH (f)-[:LIKE]->(m2:Manga)
15            WHERE NOT (u)-[:LIKE]->(m2)
16            WITH m2, COUNT(DISTINCT f) AS num_likes
17            RETURN m2 AS manga
18            ORDER BY num_likes DESC, m2.title ASC
19            LIMIT $n
20            """;
21         Value params1 = parameters("userId", userId, "n", n, "startDate", today.minusMonths(6)
22         );
23         List<MediaContentDTO> suggested = session.executeRead(
24             tx -> tx.run(query1, params1).list()
25             ).stream()
26             .map(record -> (MangaDTO) recordToMediaContentDTO(record))
27             .collect(Collectors.toList());
28
29         remaining = n - suggested.size();
30
31         // If there are not enough suggestions, add more results from the last 2 years
32         if (remaining > 0) {
33             Value params2 = parameters("userId", userId, "n", n, "startDate", today.minusYears
34             (2));
35             List<Record> records = session.executeRead(tx -> tx.run(query1, params2).list());
36             for (Record record : records) {
37                 MangaDTO mangaDTO = (MangaDTO) recordToMediaContentDTO(record);
38                 if (!suggested.contains(mangaDTO))
39                     suggested.add(mangaDTO);
40                 if (suggested.size() == n)
41                     break;
42             }
43
44             remaining = n - suggested.size();
45         }
46
47         // If there are not enough suggestions, add more results based on all likes
48         if (remaining > 0) {
49             String query2 = """
50                 MATCH (u:User {id: $userId})-[r1:LIKE]->(m:Manga)<-[:LIKE]-(f:User)
51                 WITH u, f, COUNT(m) AS common_likes
52                 ORDER BY common_likes DESC

```

```

53         MATCH (f)-[:LIKE]->(m2:Manga)
54         WHERE NOT (u)-[:LIKE]->(m2)
55         WITH m2, COUNT(DISTINCT f) AS num_likes
56         RETURN m2 AS manga
57         ORDER BY num_likes DESC, m2.title ASC
58         LIMIT $n
59         """;
60     Value params3 = parameters("userId", userId, "n", n);
61
62     List<Record> records = session.executeRead(tx -> tx.run(query2, params3).list());
63     for (Record record : records) {
64         MangaDTO mangaDTO = (MangaDTO) recordToMediaContentDTO(record);
65         if (!suggested.contains(mangaDTO))
66             suggested.add(mangaDTO);
67         if (suggested.size() == n)
68             break;
69     }
70 }
71
72     return suggested.isEmpty() ? null : suggested;
73
74 } catch (Neo4jException e) {
75     throw new DAOException(DAOExceptionType.DATABASE_ERROR, e.getMessage());
76
77 } catch (Exception e) {
78     throw new DAOException(DAOExceptionType.GENERIC_ERROR, e.getMessage());
79 }
80 }

```

- Neo4j Query:

```

1 // Match the User node with the specified userId who likes Anime nodes (r1) and those Anime
  nodes are liked by other Users (f).
2 MATCH (u:User {id: $userId})-[:LIKE]->(a:Anime)-[:LIKE]->(f:User)
3 // Ensure that the User's LIKE relationship's date is within the specified range.
4 WHERE r1.date >= $startDate
5 // Aggregate the results to count the number of common Anime nodes liked by both the User (u)
  and other Users (f).
6 WITH u, f, COUNT(a) AS common_likes
7 // Order the results by the number of common likes in descending order and limit to 20 results
  .
8 ORDER BY common_likes DESC
9 LIMIT 20
10 // Match Users (f) who like Anime nodes (a2) that are not liked by the User (u).
11 MATCH (f)-[:LIKE]->(a2:Anime)
12 WHERE NOT (u)-[:LIKE]->(a2)
13 // Aggregate the results to count the number of distinct Users (f) who like each Anime node (
  a2).
14 WITH a2, COUNT(DISTINCT f) AS num_likes
15 // Return the Anime node (a2) ordered by the number of likes by distinct Users (num_likes) in
  descending order.
16 RETURN a2 AS anime
17 ORDER BY num_likes DESC
18 // Limit the number of Anime nodes returned to the specified maximum ($n).
19 LIMIT $n

```

Testing

Testing is a substantial part of the MangaVerse web application project. Testing helps to ensure application's reliability, performance and correctness. To be able to conduct efficient testing process, two kind of tests are preformed. They are JUnit testing as a structural testing and functional testing.

Structural Testing

Structural testing also with other name white-box testing is based on testing the internal structure of the working application and it guarantees that the methods are working as expected. JUnit testing framework is used to conduct structural testing. JUnit testing is performed by testing different modules of the application such as DAOs and services. With that process each methods efficiency and correctness is guaranteed. Some examples of JUnit testing are shown below.

Example code snippet from AnimeDAOMongoImplTest:

```
1 class AnimeDAOMongoImplTest {
2
3   @BeforeEach
4   public void setUp() throws Exception {
5       BaseMongoDBDAO.openConnection();
6   }
7
8   @AfterEach
9   public void tearDown() throws Exception {
10       BaseMongoDBDAO.closeConnection();
11   }
12
13   // test 1 : search for an anime by name
14   // test 2 : search for an anime by filters
15   @Test
16   void searchTest() {
17       AnimeDAOMongoImpl animeDAO = new AnimeDAOMongoImpl();
18
19       // test 1
20       System.out.println("Search by title");
21       assertDoesNotThrow(() -> {
22           List<MediaContentDTO> animeList = animeDAO.search(List.of(Pair.of("title", "Attack
23               on Titan")), Map.of("title", 1), 1, false).getEntries();
24           for (MediaContentDTO anime : animeList) {
25               System.out.println("Id: " + anime.getId() + ", Title: " + anime.getTitle());
26           }
27       });
28
29       // test 2
30       System.out.println("Search by filters");
31       assertDoesNotThrow(() -> {
32           for (int i = 1; i < 5; i++) {
33               PageDTO<MediaContentDTO> animePage = animeDAO.search(List.of(Pair.of("$in", Map
34                   .of("tags", List.of("school clubs", "manwha"))), Map.of("title", 1), i, false);
35               if (!animePage.getEntries().isEmpty()) {
36                   for (MediaContentDTO anime : animePage.getEntries()) {
37                       System.out.println("Id: " + anime.getId() + ", Title: " + anime.
38                           getTitle());
39                   }
40               }
41           }
42       });
43   }
44 }
```

Example code snippet from Neo4jDAOImplTest:

```
1 public class Neo4JDAOImplTest {
2
3     @BeforeEach
4     public void setUp() throws Exception {
5         BaseMongoDBDAO.openConnection();
6         BaseNeo4JDAO.openConnection();
7     }
8
9     @AfterEach
10    public void tearDown() throws DAOException {
11        BaseMongoDBDAO.closeConnection();
12        BaseNeo4JDAO.closeConnection();
13    }
14
15    @Test
16    public void testFollowUser() throws DAOException {
17        try {
18            UserDAONeo4JImpl neo4JDAO = new UserDAONeo4JImpl();
19            neo4JDAO.follow("6577877be68376234760585a", "6577877be683762347605859");
20        } catch (DAOException e) {
21            fail("Exception not expected: " + e.getMessage());
22        }
23    }
24
25    @Test
26    public void testUnlikeAnime() throws DAOException {
27        try {
28            AnimeDAONeo4JImpl dao = new AnimeDAONeo4JImpl();
29            dao.unlike("6577877be68376234760585f", "65789bb52f5d29465d0abd09");
30        } catch (DAOException e) {
31            fail("Exception not expected: " + e.getMessage());
32        }
33    }
34 }
35
36
```

Example code snippet from ReviewServiceImpl:

```
1 class ReviewServiceImplTest {
2     private static final ExecutorTaskService aperiodicTaskService = ServiceLocator.
3         getExecutorTaskService(ExecutorTaskServiceType.APERIODIC);
4     private static final TaskManager errorTaskManager = ServiceLocator.
5         getErrorsTaskManager();
6     @BeforeEach
7     public void setUp() throws Exception {
8         BaseMongoDBDAO.openConnection();
9         BaseNeo4JDAO.openConnection();
10        aperiodicTaskService.start();
11        errorTaskManager.start();
12    }
13
14    @AfterEach
15    public void tearDown() throws Exception {
16        BaseMongoDBDAO.closeConnection();
17        BaseNeo4JDAO.closeConnection();
18        aperiodicTaskService.stop();
19        errorTaskManager.stop();
20    }
21
22    @Test
23    void updateReview() {
24        ReviewServiceImpl reviewService = new ReviewServiceImpl();
25        try {
26            ReviewDTO reviewAnime = createSampleAnimeReview();
27            assertDoesNotThrow(() -> reviewService.addReview(reviewAnime));
28            reviewAnime.setComment("This is an updated test review");
29            reviewAnime.setRating(4);
30            assertDoesNotThrow(() -> reviewService.updateReview(reviewAnime));
31        }
32    }
33
34 }
35
36
```

```

28         System.out.println("Anime review updated: " + reviewAnime);
29
30         ReviewDTO reviewManga = createSampleMangaReview();
31         assertDoesNotThrow(() -> reviewService.addReview(reviewManga));
32         reviewManga.setComment("This is an updated test review");
33         reviewManga.setRating(4);
34         assertDoesNotThrow(() -> reviewService.updateReview(reviewManga));
35         System.out.println("Manga review updated: " + reviewManga);
36     } catch (BusinessException e) {
37         fail(e);
38     }
39 }
40 }
41

```

Functional Testing

Functional testing also with other name black-box testing is based on testing the application's external functionalities. It checks the application from end-user's perspective. It ensures that specified requirements are provided efficiently by the web application and expected outcome is created. With the help of the use cases and real world scenarios, functional testing is conducted. Some examples of functional testing are shown below.

Table 5.1: Functional Test Cases

| Id | Description | Input | Expected Output | Output | Outcome |
|-----------|---|---|---|---------------|----------------|
| User_01 | Login with correct information | email: nmiller@example.com, password: f6d6b3ffecb44a... | The user logs in successfully | | |
| User_02 | Login with wrong information | email: wrong@example.com, password: wrong | The user is not able to log in successfully | | |
| User_03 | Signup with all the mandatory info are filled | | | | |
| User_04 | Signup with missing info | | | | |
| User_05 | Update user information | description: manga lover | User profile is updated with new info. | | |
| User_06 | Follow another user | - | User is followed. | | |
| User_07 | Unfollow another user | - | User is unfollowed. | | |
| User_08 | Search manga by title | title: "Slam Dunk" | The list of manga which includes the words of "Slam Dunk" is shown. | | |
| User_09 | Search manga by detailed filtering | | | | |
| User_10 | Like anime | - | The anime is liked | | |
| User_11 | Add review to anime | review: "I like the anime" | The review is added to the anime and displayed in the anime page | | |
| User_12 | Update review | review: "I dont like this anime anymore" | The review is updated with the new one. | | |

| Id | Description | Input | Expected Output | Output | Outcome |
|-----------|---|--------------|--|---------------|----------------|
| Admin_01 | See users distribution analytics | - | | | |
| Admin_02 | See manga analytics for get average rating by month | Year:2020 | Average rating for each month in 2020 is displayed in the page | | |
| Admin_03 | See anime analytics for get trend media content by year | | | | |
| Admin_04 | | | | | |
| Admin_05 | | | | | |

Performance Testing

Performance testing is conducted to ensure that MangaVerse web application is able to handle the a high volume of operations efficiently and provides a smooth experience for users. It is important to test the application's performance to ensure that it can handle the expected load. Performance testing is applied on MongoDB and Neo4j databases. Specifically, the aim of the performance testing is to see the impacts of indexing on CRUD operations and aggregation operations.

MongoDB Performance Testing

Indexes are used in MongoDB to improve the performance of queries. Indexes are used to quickly locate data without having to search every document in a collection. This limiting the search with indexes, results can get with faster response time. For the mongoDB performance testing, *username* is used to indexing Users collection and *title* is used to indexing Anime and Manga collections. The tests are conducted for creating new documents and searching for documents. As it can be shown in the table below, there are significant improvement in operation times with indexing, especially for search tasks

Table 5.2: MongoDB Performance Test Results

| Collection | Operation | Index | Time (ms) | Total Keys Examined | Total Docs Examined |
|-------------------|------------------|--------------|------------------|----------------------------|----------------------------|
| Users | Insert | No | 3 | - | - |
| | Search | No | 17 | 0 | 10007 |
| | Insert | Yes | 5 | - | - |
| | Search | Yes | 5 | 1 | 1 |
| Anime | Insert | No | 3 | - | - |
| | Search | No | 88 | 0 | 30113 |
| | Insert | Yes | 3 | - | - |

Continued on next page

Table 5.2 – continued from previous page

| Collection | Operation | Index | Time (ms) | Total Keys Examined | Total Docs Examined |
|------------|-----------|-------|-----------|---------------------|---------------------|
| Manga | Search | Yes | 10 | 0 | 1 |
| | Insert | No | 3 | - | - |
| | Search | No | 141 | 0 | 41677 |
| | Insert | Yes | 3 | - | - |
| | Search | Yes | 10 | 1 | 1 |

Neo4j Performance Testing

Similarly, also for Neo4j database, performance testing is conducted to observe the impacts of indexing on some CRUD operations. Indexing with ids is used for both anime, manga and users nodes. The tests are conducted for creating new nodes and searching for nodes. The test result in the table below shows that indexing has a significant impact on operations in Neo4j database especially for search. Indexing time decreases with using indexes for search tasks because it prevents to check all the nodes in the database.

Table 5.3: Neo4j Performance Test Results

| Collection | Operation | Index | Time (ms) |
|------------|-----------|-------|-----------|
| Anime | Insert | No | 5 |
| | Insert | Yes | 5 |
| | Search | No | 45 |
| | Search | Yes | 2 |
| Manga | Insert | No | 7 |
| | Insert | Yes | 5 |
| | Search | No | 46 |
| | Search | Yes | 9 |
| Users | Insert | No | 56 |
| | Insert | Yes | 9 |
| | Search | No | 40 |
| | Search | Yes | 3 |

Conclusion

Conclusion

The MangaVerse is a web application project that provides a comprehensive web application for dynamic social platform for manga and anime enthusiasts. The web application allows users to explore, search media content and be in contact with other users by review system. Having a user-friendly interface, the application is designed to have a robust set of features. The applications offers functionalities for both unregistrered user and registered user including manager purposes such as browse media content, personalized recommendations, profile management and analytics checking for management purposes.

Beside the functional requirements, the application has also well-defined development process and architecture using different technologies and techniques. While java is used for main backend development programming language, as a database MongoDB and Neo4j are used.

Future Work

For the future: manager will be able to update add delete anime and manga and delete user.

Security

- Data Encryption: All user data, including passwords, should be securely encrypted during transmission and storage.
- Delete user accounts if necessary.user management