



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	4
2.4 UML use case diagram	5
2.5 UML class diagram	6
2.6 Data Modeling	7
3 Design	8
3.1 Document Database	8
3.2 MongoDB queries	12
3.3 Graph Database	15
3.4 GraphDB queries	16
3.5 Availability and Partition Tolerance	18
3.6 Redundancy	18
3.7 Replicas	18
3.8 Sharding	18
4 Implementation	19
4.1 Development Environment	19
4.2 Main Modules	19
4.3 Adopted Patterns and Techniques	25
5 Testing	26
5.1 Structural Testing	26
5.2 Functional Testing	27
6 Conclusion	29
6.1 Conclusion	29
6.2 Future Work	29

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, and interact with a vast collection of manga and anime. Unregistered users can browse the manga and anime, access basic details about each media content and utilize search and filtering options. Users can register and with registering, user can personalize their profiles and engage in a community with. The platform also offers a set of features for registered users, such as liking media contents, following other users, adding reviews and ratings. Additionally, users receive personalized suggestions based on their preferences and current trends. The site manager has access to detailed analytics about media contents and user activities.

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 in user interactions and popular genres.

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 created an account on the platform. They can browse media contents, search and filter, view media content details, and register/login.
- **Registered User:** A user who has created an account on the platform. They can perform all the actions of an unregistered user, as well as logout, manage their profile, explore other user profiles, interact with media contents/users, review media contents, and receive advanced recommendations.
- **Manager:** A registered user with administrative features. They can access an analytics dashboard, manage user accounts, content entries, and monitor trends.

Requirements

Unregistered User:

- Browse Media Contents:
 - View a list of available manga and anime on the home page.
 - Access basic details about each media content without logging in.
- Search and Filter:
 - Use the search bar to find specific manga or anime by title.
 - Utilize basic filtering options to refine the media content list.
- View Media Content Details:
 - Click on a media content to view detailed information, including synopsis and genre.
- Register/Login:
 - Access a registration page to create a new account.
 - Use valid credentials to log into the account.
- Explore Features:
 - Access information about the features available to registered users.

Registered User:

- Browse Media Contents:
 - View a list of available manga and anime on the home page.
 - Access basic details about each media content without logging in.
- Search and Filter:
 - Use the search bar to find specific media content by title.
 - Utilize basic filtering options to refine the media content list.
- View Media Content Details:
 - Click on a media content to view detailed information, including synopsis and genre.
- Logout:

- Ends the user’s session.
- Profile Management:
 - Edit and update personal information (e.g., profile picture, bio).
 - Change account password.
- Explore Other User Profiles:
 - View profiles of other registered users.
 - See their liked manga, anime and reviews.
- Interact with Media Contents/Users:
 - Like or dislike manga and anime to indicate preferences.
 - Follow/unfollow other users.
- Review Media Contents:
 - Add reviews and ratings to manga and anime.
 - View and edit own reviews.
- Advanced Recommendations:
 - Receive more refined media content suggestions based on detailed user interactions.
 - Receive users suggestions based on common interests.

Manager(Registered User with Administrative Features):

- Analytics Dashboard:
 - Access a comprehensive analytics dashboard with data on user engagement and media contents trends.
- User Management:
 - View and manage user accounts, including account activation and deactivation.
- Content Management:
 - Manage media content entries, including adding new manga and anime, updating information, and removing entries if necessary.
- Monitor Trends:
 - Monitor trends in user interactions, popular genres, and trending manga.

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.

Security

- Data Encryption: All user data, including passwords, should be securely encrypted during transmission and storage.

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.

UML use case diagram

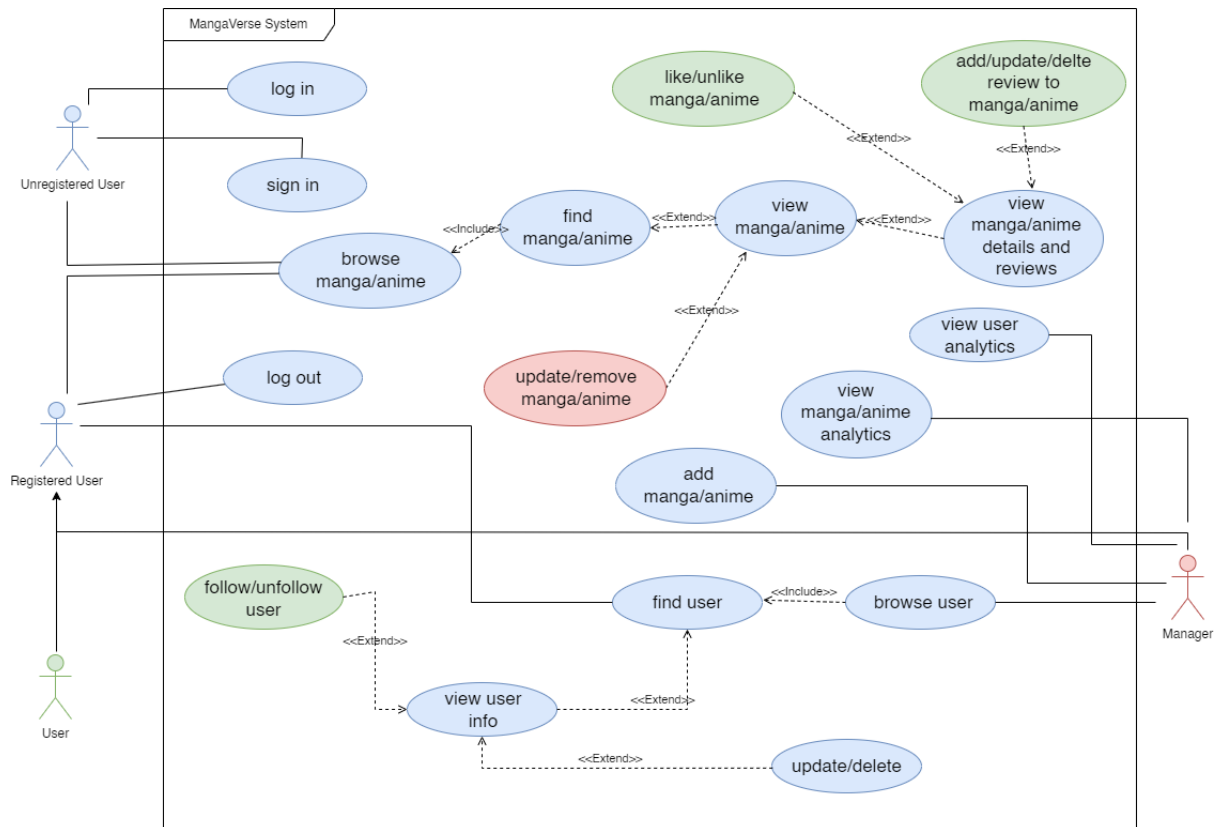


Figure 2.1: UML Class Diagram

UML class diagram

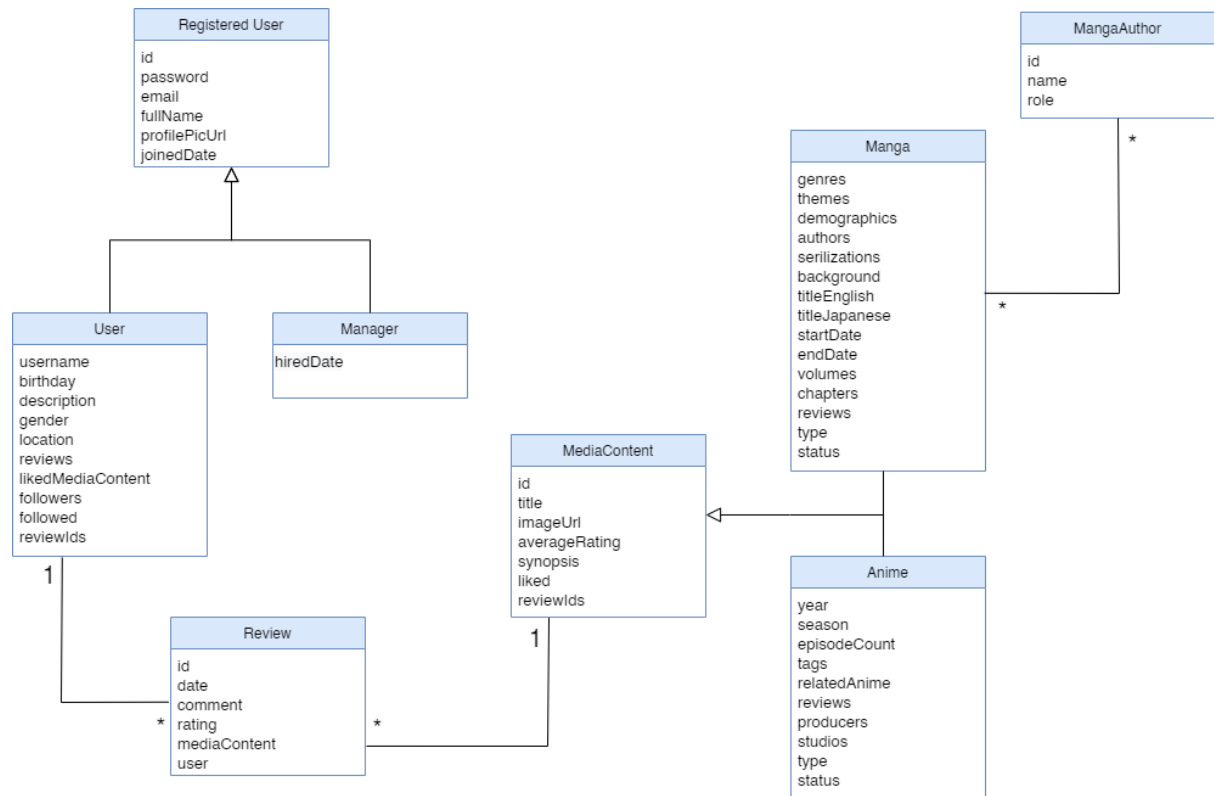


Figure 2.2: UML Class Diagram

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. MongoDB is a document database, which means it stores data in JSON-like documents. 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. This makes MongoDB a good choice for applications that need to scale quickly.

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 }
```

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 }
```

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

Indexes

We created two indexes in the reviews collection to improve query performance. One for the users id and another for the anime and manga id. This will allow us to quickly retrieve reviews for a specific user or media content.

MongoDB queries

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

USERS:

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

```
1 // Match stage to filter documents where 'criteriaOfSearch' exists
2 db.collection.aggregate([
3     {
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 ]);
29
```

Listing 3.1: GetDistribution

ANIME/MANGA:

- GetBestCriteria query, the criteria can be genres, demographics, themes, authors and serialization for manga; tags, producers, studios for anime:

```
1 db.collection.aggregate([
2     // Match stage to filter documents where 'criteria' exists and 'average_rating' is
3     // not null
4     {
5         $match: {
6             criteria: { $exists: true },
7             average_rating: { $ne: null }
8         }
9     },
10    // Unwind stage to deconstruct the 'criteria' array field
11    {
12        $unwind: "$" + criteria
13    },
14    // Group stage to calculate the average rating for each criteria
15    {
16        $group: {
17            _id: "$" + criteria,
18            criteria_average_rating: { $avg: "$average_rating" }
19        }
20    },
21    // Sort stage to sort documents by 'criteria_average_rating' in descending order
22    {
23        $sort: {
24            criteria_average_rating: -1
25        }
26    },
27    // Skip stage to skip the first 'pageOffset' documents
28 ]);
```

```

27     {
28         $skip: pageOffset
29     },
30     // Limit stage to limit the results to 25 documents
31     {
32         $limit: 25
33     }
34 ];
35

```

Listing 3.2: GetBestCriteria

REVIEWS:

- GetMediaContentRatingByYear query to get the average rating of media content by year:

```

1  // Match stage to filter documents based on specified conditions
2  db.collection.aggregate([
3      {
4          $match: {
5              [`${nodeType}.id`]: new ObjectId(mediaContentId),
6              rating: { $exists: true },
7              date: { $gte: startDate, $lt: endDate }
8          }
9      },
10     // Group stage to group documents by year and calculate the average rating
11     {
12         $group: {
13             _id: { $year: "$date" },
14             average_rating: { $avg: "$rating" }
15         }
16     },
17     // Project stage to shape the output documents
18     {
19         $project: {
20             _id: 0,
21             year: "$_id",
22             average_rating: 1
23         }
24     },
25     // Sort stage to sort documents by year in ascending order
26     {
27         $sort: { year: 1 }
28     }
29 ];
30

```

Listing 3.3: GetMediaContentRatingByYear

- SuggestMediaContent query to suggest media content based on common criteria, like birthday or location:

```

1  db.collection.aggregate([
2      {
3          // Match documents based on a dynamic user criteria
4          $match: {
5              ["user." + criteriaType]: criteriaValue
6          }
7      },
8      {
9          // Group documents by node type ID and calculate the first title and average rating
10         $group: {
11             _id: "$" + nodeType + ".id", // Group by the node type's ID
12             title: { $first: "$" + nodeType + ".title" }, // Get the first title in the group
13             average_rating: { $avg: "$rating" } // Calculate the average rating for the group
14         }
15     },
16     {
17         // Sort the grouped documents by average rating in descending order
18         $sort: { average_rating: -1 }
19     }
20 ];

```

```
19     },
20     {
21         // Limit the number of results to the page size constant
22         $limit: Constants.PAGE_SIZE
23     }
24 ];
```

Listing 3.4: SuggestMediaContent

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.

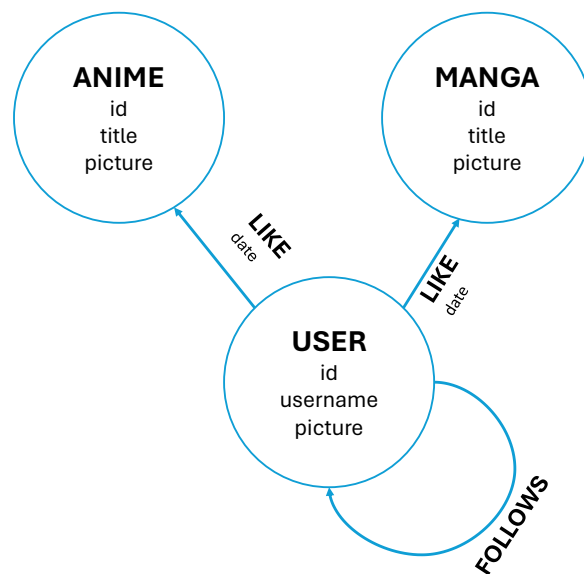


Figure 3.1: GraphDB

GraphDB queries

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

USERS:

- Suggest other users followings in common:

```
1
2 MATCH (u:User {id: $userId})-[:FOLLOWS]->(following:User)<-[:FOLLOWS]-(suggested:User)
3 WHERE NOT (u)-[:FOLLOWS]->(suggested) AND u <> suggested
4 WITH suggested, COUNT(DISTINCT following) AS commonFollowers
5 WHERE commonFollowers > 5
6 RETURN suggested as user, commonFollowers
7 ORDER BY commonFollowers DESC
8 LIMIT $n
```

Listing 3.5: SuggestUsersByCommonFollowings

- Suggest other users likes in common:

```
1 MATCH (u:User {id: $userId})-[r:LIKE]->(media:Manga)<-[:LIKE]-(suggested:User)
2 WHERE u <> suggested AND r.date >= $date
3 WITH suggested, COUNT(DISTINCT media) AS commonLikes
4 WHERE commonLikes > $min
5 RETURN suggested AS user, commonLikes
6 ORDER BY commonLikes DESC
7 LIMIT $n
```

Listing 3.6: SuggestUsersByCommonLikes

ANIME/MANGA:

- Suggest anime and manga based on following in common:

```
1 MATCH (u:User {id: $userId})-[:FOLLOWS]->(f:User)-[r:LIKE]->(a:Anime)
2 WHERE NOT (u)-[:LIKE]->(a) AND r.date >= $startDate
3 WITH a, COUNT(DISTINCT f) AS num_likes
4 RETURN a AS anime
5 ORDER BY num_likes DESC
6 LIMIT $n
```

Listing 3.7: GetSuggestedByFollowings

- Suggest anime and manga based on likes in common between followers:

```
1 MATCH (u:User {id: $userId})-[r1:LIKE]->(a:Anime)<-[:LIKE]-(f:User)
2 WHERE r1.date >= $startDate
3 WITH u, f, COUNT(a) AS common_likes
4 ORDER BY common_likes DESC
5 LIMIT 20
6 MATCH (f)-[:LIKE]->(a2:Anime)
7 WHERE NOT (u)-[:LIKE]->(a2)
8 WITH a2, COUNT(DISTINCT f) AS num_likes
9 RETURN a2 AS anime
10 ORDER BY num_likes DESC
11 LIMIT $n
```

Listing 3.8: GetSuggestedByLikes

- Get the trend of media contents in a specific year based on the number of likes:

```
1 MATCH (a:Anime)<-[:LIKE]-(u:User)
2 WHERE r.date >= $startDate AND r.date < $endDate
3 WITH a, count(r) AS numLikes
4 ORDER BY numLikes DESC
5 RETURN a AS anime, numLikes
6 LIMIT $n
```

Listing 3.9: GetTrendMediaContentByYear

- Get the general trend of media contents based on the number of likes:

```

1 MATCH (u:User)-[r:LIKE]->(a:Anime)
2 WHERE r.date >= $startDate
3 WITH a, COUNT(r) AS numLikes
4 ORDER BY numLikes DESC
5 RETURN a AS anime, numLikes
6 LIMIT $n

```

Listing 3.10: GetMediaContentTrendByLikes

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 3.11: 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 3.12: 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 3.13: 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 3.14: 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 3.15: 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 3.16: 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 is 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 in the data efficiently.

Integrated Development Environment

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

Version Control

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

Web Server

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

Build Automation

Maven is 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 is 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. Each controller class utilized a switch-case structure to determine the action requested and invokes the appropriate handler method accordingly.

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         } catch (Neo4jException e) {
39             throw new DAOException(DAOExceptionType.DATABASE_ERROR, e.getMessage());
40         } catch (Exception e) {
41             throw new DAOException(DAOExceptionType.GENERIC_ERROR, e.getMessage());
42         }
43     }
44 }
```


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.

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.

```
class MangaDAOMongoImplTest { /* Flavio Messina */
    // test 1 : update an existing manga (Before that, I try to find the manga by title)
    // test 2 : update a non-existing manga
    @Test /* Flavio Messina */
    void updateMediaContentTest() throws DAOException {
        MangaDAOMongoImpl mangaDAO = new MangaDAOMongoImpl();
        List<MediaContentDTO> mangaList = mangaDAO.search(List.of(Map.of("title", "Sample Manga")), Map.of("title", "1", "page", 1, "reducedinfo": false).getEntries());
        // test 1
        if (!mangaList.isEmpty()) {
            MediaContentDTO mangaToUpdate = mangaList.getFirst();
            System.out.println("Manga to update: " + mangaToUpdate);
            Manga manga = new Manga();
            manga.setId(mangaToUpdate.getId());
            manga.setTitle("Updated Manga");
            assertDoesNotThrow(() -> mangaDAO.updateMediaContent(manga));
            System.out.println("Manga updated");
        }
        // test 2
        Manga manga = createSampleManga();
        manga.setId("6635632b4276578429f29384");
        manga.setTitle("Non-existing Manga");
        assertThrows(DAOException.class, () -> mangaDAO.updateMediaContent(manga));
        System.out.println("Non-existing Manga not found");
    }

    // test 1 : delete an existing manga (Before that, I try to find the manga by title)
    // test 2 : delete a non-existing manga
    @Test /* Flavio Messina */
    void deleteMediaContentTest() throws DAOException {
        MangaDAOMongoImpl mangaDAO = new MangaDAOMongoImpl();
        List<MediaContentDTO> mangaList = mangaDAO.search(List.of(Map.of("title", "Sample Manga")), Map.of("title", "1", "page", 1, "reducedinfo": false).getEntries());
        // test 1
        if (!mangaList.isEmpty()) {
            MediaContentDTO mangaToDelete = mangaList.getFirst();
            System.out.println("Manga to delete: " + mangaToDelete);
            assertDoesNotThrow(() -> mangaDAO.deleteMediaContent(mangaToDelete.getId()));
            System.out.println("Manga deleted");
        }
        // test 2
        assertThrows(DAOException.class, () -> mangaDAO.deleteMediaContent("mangaid: 6635632b4276578429f29384"));
        System.out.println("Non-existent manga not deleted");
    }
}
```

Figure 5.1: MangaDAOMongoImpl Class Test Example

```

class UserServiceImplTest {
    @Test
    void followTest() {
        try {
            UserService userService = new UserServiceImpl();
            UserSummaryDTO userSummaryDTO = userService.searchFirstUsers(
                username: "exampleUser", no: 1, loggedUser: null).getFirst();
            UserSummaryDTO userSummaryDTO2 = userService.searchFirstUsers(
                username: "exampleUser2", no: 1, loggedUser: null).getFirst();
            System.out.println(userSummaryDTO);
            System.out.println(userSummaryDTO2);
            userService.follow(userSummaryDTO.getId(), userSummaryDTO2.getId());
            Thread.sleep(2*1000);
        } catch (BusinessException e) {
            System.err.println(e.getMessage() + " " + e.getType());
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }

    @Test
    void unfollowTest() {
        try {
            UserService userService = new UserServiceImpl();
            UserSummaryDTO userSummaryDTO = userService.searchFirstUsers(
                username: "exampleUser", no: 1, loggedUser: null).getFirst();
            UserSummaryDTO userSummaryDTO2 = userService.searchFirstUsers(
                username: "exampleUser2", no: 1, loggedUser: null).getFirst();
            System.out.println(userSummaryDTO);
            System.out.println(userSummaryDTO2);
            userService.unfollow(userSummaryDTO.getId(), userSummaryDTO2.getId());
            Thread.sleep(2*1000);
        } catch (BusinessException e) {
            System.err.println(e.getMessage() + " " + e.getType());
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }
}

```

Figure 5.2: UserServiceImpl Class Test Example

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: Admin Test case

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 updaated with the new one.		
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					

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
