



University of Pisa  
MSc in Artificial Intelligence and Data Engineering  
Data Mining and Machine Learning

# AnimeBook

Andrea Di Marco  
Stefano Dugo

Academic Year 2021/2022

# Contents

<b>1</b>	<b>Design</b>	<b>2</b>
1.1	The Application . . . . .	2
1.2	Requirements . . . . .	2
1.2.1	The Actors . . . . .	2
1.2.2	Functional Requirements . . . . .	2
1.2.3	Non-Functional Requirements . . . . .	3
1.3	Use Cases Diagram . . . . .	3
1.4	Classes Analysis . . . . .	4
1.5	Software architecture . . . . .	4
<b>2</b>	<b>Machine Learning</b>	<b>6</b>
2.1	Introduction . . . . .	6
2.2	Dataset . . . . .	6
2.3	Pre-Processing . . . . .	6
2.3.1	Feature Selection . . . . .	6
2.3.2	Data Transformation . . . . .	7
2.3.3	Clustering Process . . . . .	8
2.3.4	Recommendation System . . . . .	16
2.3.5	Conclusions . . . . .	17
<b>3</b>	<b>Application Code</b>	<b>19</b>
3.1	Class organization . . . . .	19
3.2	Layout . . . . .	19
3.3	Controller . . . . .	20
3.4	DBManager . . . . .	23
3.5	Entities . . . . .	25
3.6	Datamining . . . . .	26
3.7	Utils . . . . .	26
3.8	Python Modules: . . . . .	26
<b>4</b>	<b>User Manual</b>	<b>28</b>
4.1	Login . . . . .	28
4.2	Signing Up . . . . .	28
4.3	Your Anime List . . . . .	31
4.4	Search Animes . . . . .	34
4.5	Editing favourite genres . . . . .	40

# 1 Design

## 1.1 The Application

*AnimeBook* is a personal anime list application service, which stores information about anime and users.

Every user can register an account and log into it; after the registration process, the user can search for an anime and add it to his/her personal anime list which will be created after the addition of the first element.

After the creation of this list, every user can browse it, selecting an anime and assign a vote or remove it. Our application, also offer the possibility to select anime from recommendation list generated by his/her preferences contained in the personal anime list.

## 1.2 Requirements

### 1.2.1 The Actors

The application requires the presence of *two* main actors:

- **Unregistered User:** This user have to register a new account or login to an existing one, in order to use the application's features.
- **User:** This is the main user of the application: he/she can browse all the anime contained in the database and add them to his/her list. Access to his/her personal list and rate or remove anime featured in it. Every user can also browse a list of recommended anime and add them to his/her list.

### 1.2.2 Functional Requirements

- The application must provide a registration form in order to allow a *Unregistered User* to create a personal account for the application.
- The application must allow an *Unregistered User* to select its favourite genres during registration.
- The application must provide a login form in order to allow *Unregistered Users* to log into their personal account.
- The application must deny any functionalities aside from registration to *Unregistered Users*.
- The application have to allow *Users* to browse the application's anime database and view their details upon selecting one and/or add them to their personal anime list.
- The application must allow *Users* to create a personal anime list in which they can save a list of their favourite shows.

- The application must allow *Users* to browse their personal anime list, select an anime and rate it based on his/her preferences.
- The application must give the chance to show *Users* a list of recommended anime based on their anime preferences (gathered by the analysis of their personal anime list).
- The application must give the chance to the *Users* to edit their favourite genres.

### 1.2.3 Non-Functional Requirements

- The application must embed one or more machine learning algorithms.
- Usability: The application must adopt a user friendly interface, easy to access and to interact with.

## 1.3 Use Cases Diagram

This is the use case diagram adopted for *Animebook*.

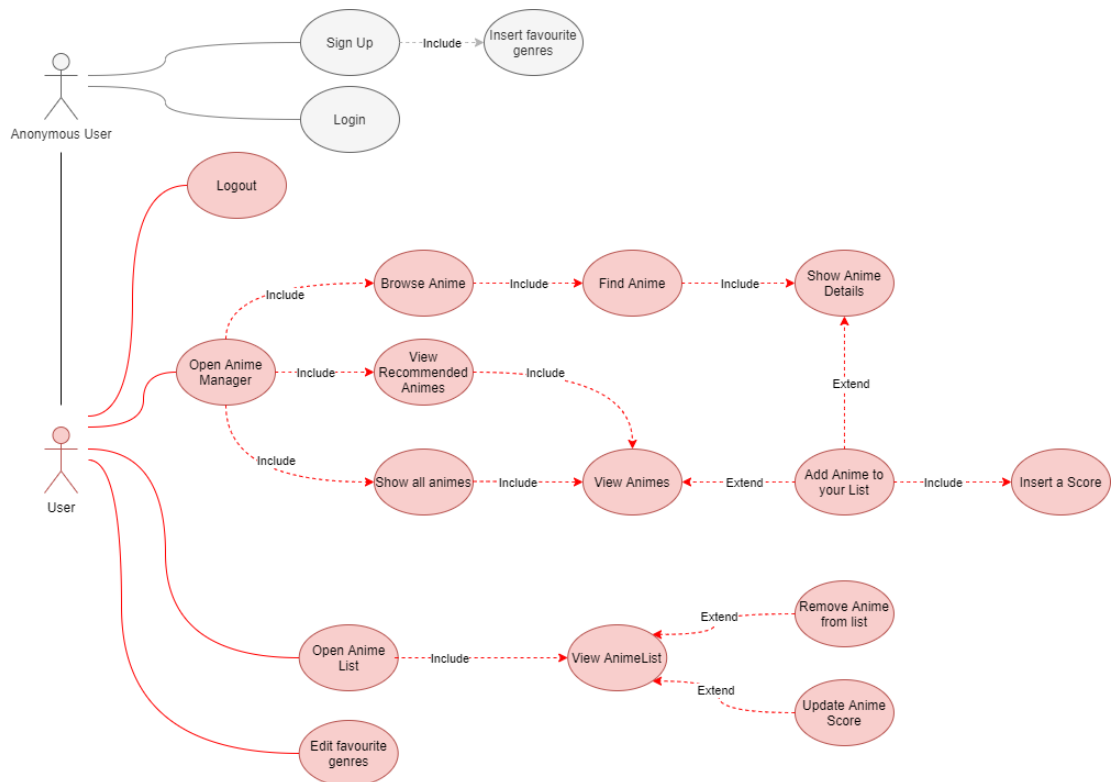


Figure 1: Use Case Diagram

## 1.4 Classes Analysis

In the application we have users, which have their personal animelist, which can contain an undefined number of animes, and each anime has a list of genres which define them; also an user has a list of favourite genres, based on his selected preference and on the anime the user added to its list.

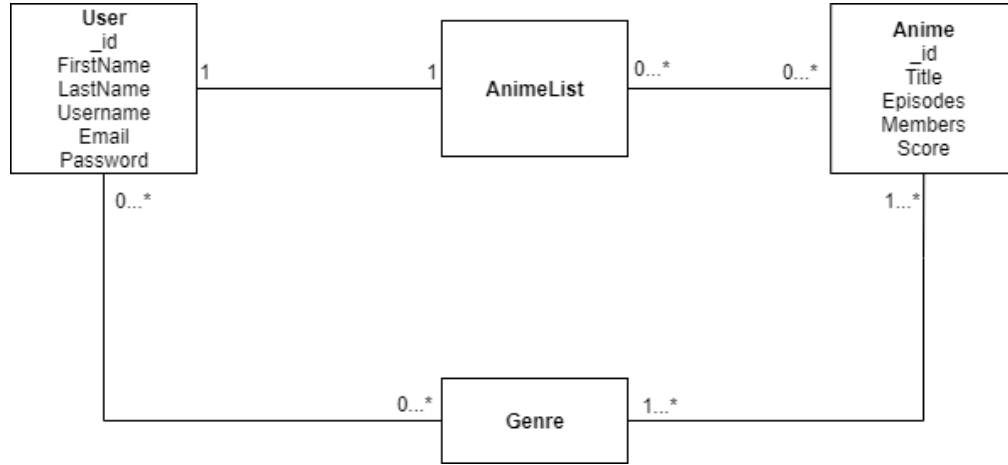


Figure 2: Use Case Diagram

## 1.5 Software architecture

This application is written in Java. We used JavaFX to implement a graphical user interface. The database is managed through MongoDB. We decided to use it since we thought that was better suited to handle data: in particular the usage of embedded document structures was the best choice to handle substructures as the anime list or the genre list.

For the datamining task we used Weka java libraries for developing the application, but we also used Python for testing and evaluating clustering algorithms' performances.

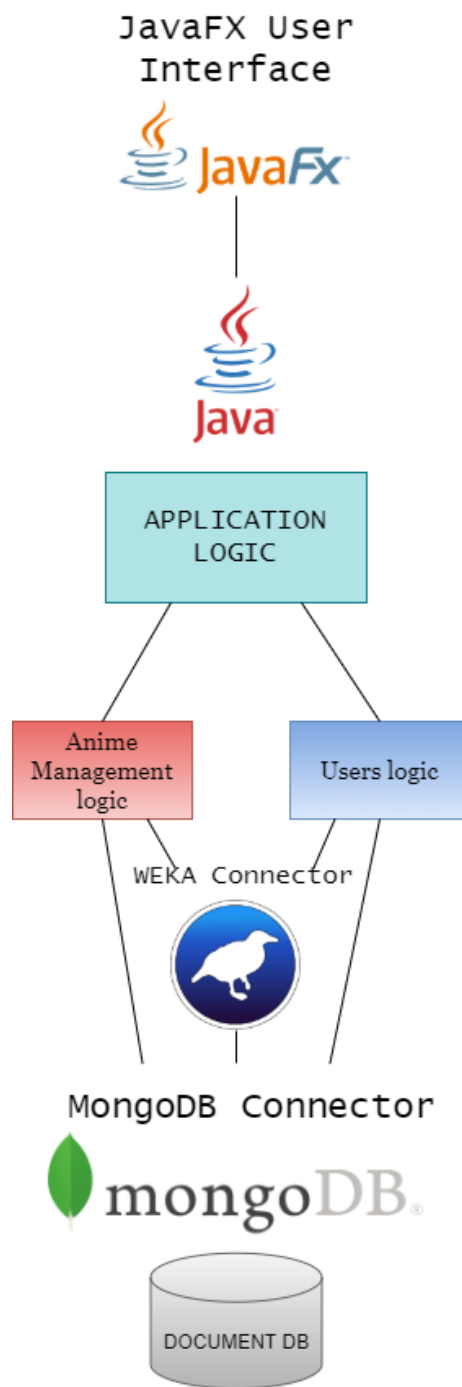


Figure 3: Software architecture of the application

## 2 Machine Learning

### 2.1 Introduction

### 2.2 Dataset

We decided to use an Anime database offered by the Kaggle platform (<https://www.kaggle.com/kerneler/starter-anime-dataset-with-reviews-6fdcf071-6/data>).

This dataset contains information about Anime (16k), Reviews (130k) and Profiles (47k) crawled from <https://myanimelist.net/> at 05/01/20.

The dataset contains 3 files:

- *animes.csv* contains list of anime, with title, title synonyms, genre, duration, rank, popularity, score, airing date, episodes and many other important data about individual anime providing sufficient information about trends in time about important aspects of anime. Rank is in float format in csv, but it contains only integer value. This is due to NaN values and their representation in pandas.
- *profiles.csv* contains information about users who watch anime, namely username, birth date, gender, and favorite animes list.
- *reviews.csv* contains information about reviews users per animes, with text review and scores.

### 2.3 Pre-Processing

#### 2.3.1 Feature Selection

Since our database presented many kind attributes, we decided to remove those that were not useful for our analysis. In particular, we decided to remove entirely the *reviews.csv* dataset and create a new embedded document "*animeList*" which contained the *anime title* and the *overall score* assigned by the user. This new entity was added to every user in the *profiles.csv* dataset.

Since our objective was to build a recommendation system, we also decided to remove unnecessary attributes from the user dataset, in particular:

- *gender*, since our clustering approach wasn't aimed for a gender-based classification;
- *birthday*, since our clustering approach wasn't aimed for an age-based classification;
- *link*, since we wanted to guarantee user's privacy;
- *profile*, since the clustering won't make use of the user's username;
- *userid*, since the clustering won't make use of the user's id;
- *animelist* since the clustering won't make direct use of the user's animelist.

### 2.3.2 Data Transformation

In order to create an effective suggestion system based on tags, as stated above, we decided to not use directly an user animelist, but to create a new embedded document containing all users' genre preferences. This document, called "genres", has been created summing all Anime genres taken from a user animelist selecting every element that is considered as "liked" (i.e. received a score equal or higher than 6) by that specific user; for example, if a user has an Anime in his/her animelist with the genres "Action" and "Adventure" and its rating is equal or higher than 6, the genre sub-document will have both fields "Action" and "Adventure" increased by one.

At the end of the process, an user genres' list should appear as shown in Fig.4

```
  _id: ObjectId("61c8471f3853973393160b72")
  profile: "Kitlan"
  userid: 38731
  > animelist: Array
  ~ genres: Object
    Action: 28
    Adventure: 16
    Comedy: 16
    Demons: 0
    Drama: 8
    Fantasy: 4
    Game: 0
    Historical: 0
    Horror: 4
    Magic: 2
    Mecha: 14
    Military: 5
    Music: 1
    Mystery: 1
    Parody: 1
    Police: 1
    Psychological: 1
    Romance: 3
    School: 2
    Sci-Fi: 53
    Seinen: 4
    Shoujo: 1
    Shounen: 12
    Slice of Life: 1
    Space: 9
    Sports: 1
    Super Power: 5
    Supernatural: 3
    Thriller: 0
```

Figure 4: User Genres' List

We also decided to remove duplicates from every dataset, removing a total of:

- 61593 duplicates from the reviews dataset
- 2943 duplicates from the anime dataset



- 33820 duplicates from the user dataset

After the removal of every duplicate, the dataset resulted still very unbalanced, since more than 30k users had less than 3 anime in their animelist. So, to further balance the dataset, we decided to exclude from our analysis every user having less than 35 anime in his/her animelist, since they were considered statistically irrelevant to determine the approval trends.

The resulting dataset was composed by 9157 users presenting a more balanced dataset, as show in Fig.5

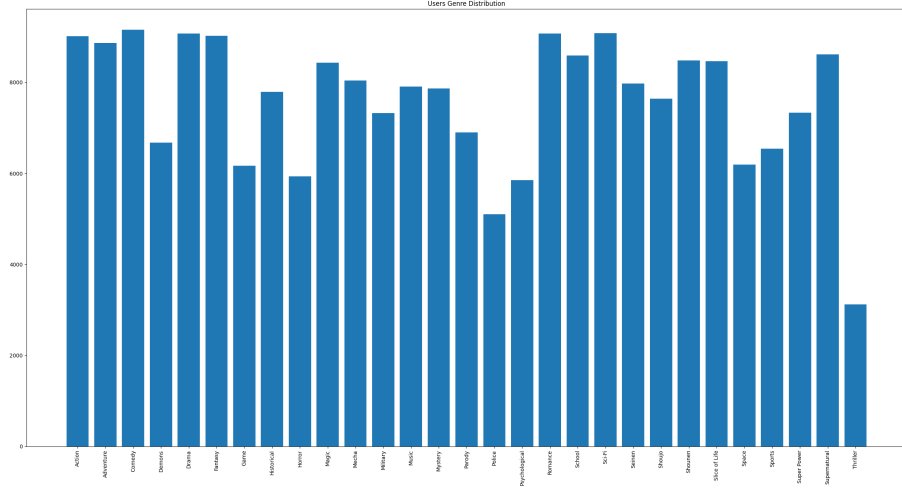


Figure 5: Genres Distribution. [y axis represents the number of instances, x axis represents the different genres]

### 2.3.3 Clustering Process

In order to create an efficient suggestion system, we opted to group the users according to their tastes, so the use of a clustering approach was the best choice, since in our problem no ground truth was been defined.

To find the best clustering algorithm, we decided to test both the KMeans algorithm and an agglomerative hierarchical algorithm offered by the python library scikit-learn.

**KMeans:** This particular algorithm offers good performances and it's one of the easiest to implement, thus we decided to test it.

One of the most important tasks to face when using the kmeans algorithm, is to find the correct value of K (i.e. the number of clusters that we want to generate).

In order to do this, we decided to use the elbow method, a widely used heuristics that consist in running the kmeans n number of times, (where n is the number

of runs of the kmeans using different values of  $k$ ), and plot the cost function for each value of  $k$  used.

The variation of the cost function is used to analyze if the dataset presents some natural clusters or not: in fact, following the natural disposition of the points inside the dataset, will lead to a significant decrease in the cost function, while dividing further the dataset will give a really lower decrease in cost. So, our objective is to find a "turning point" at a certain value of  $k$ .

Unfortunately, since our dataset was composed by scraped data from random real users, we found that our plot didn't present any "turning point", thus no natural clusters. (Fig.6)

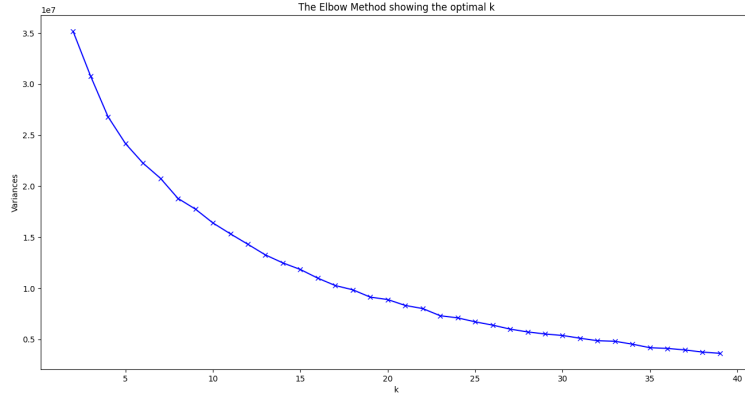


Figure 6: Elbow Method

<b>K Value</b>	<b>Cost Function</b>
2	35175751.13996
3	30795904.19606
4	26807830.05558
5	24185906.04886
6	22277075.10175
7	20548428.55445
8	18811575.93885
9	17746074.44409
10	16393423.71402
11	15436239.19552
12	14318343.46089
13	13318489.36956
14	12603950.26063
15	11761726.94342

16	10925396.83088
17	10343722.29353
18	9826416.33914
19	9146313.34188
20	8818166.05637
21	8197752.62405
22	8012560.77606
23	7545224.02746
24	7111904.73162
25	6688670.24340
26	6466368.24480
27	6152064.41261
28	5833035.10309
29	5559784.89486
30	5384011.64850
31	5072938.85364
32	4858715.00000
33	4789318.62900
34	4550949.76500
35	4251032.10683
36	4075960.00657
37	4057872.54785
38	3722896.82200
39	3688179.40689

Since the elbow method was clearly not enough to choose the best value of K, we decided to use the Silhouette analysis. This particular method, is used to study the separation distance between the resulting clusters. The silhouette plot displays a measure of how close each point in one cluster is to points in the neighboring clusters and thus provides a way to assess parameters like number of clusters visually. This measure has a range of  $[-1, 1]$ . Silhouette coefficients (as these values are referred to as) near +1 indicate that the sample is far away from the neighboring clusters. A value of 0 indicates that the sample is on or very close to the decision boundary between two neighboring clusters and negative values indicate that those samples might have been assigned to the wrong cluster.

The resulting plot of our silhouette score is shown in the figure below (Fig.7)

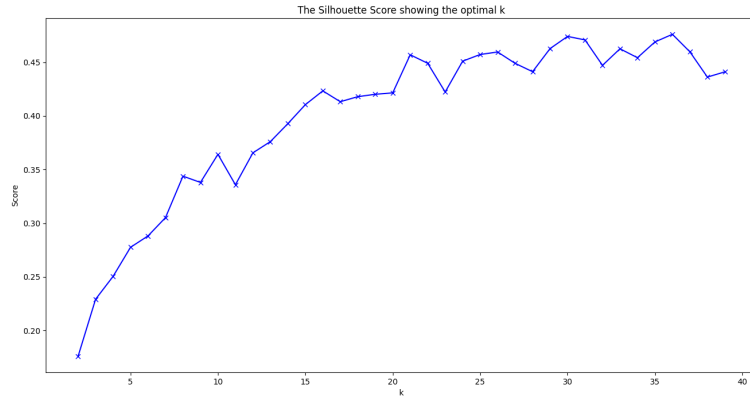


Figure 7: Silhouette Score Plot

K Value	Silhouette Score
2	0.1760
3	0.2218
4	0.2504
5	0.2767
6	0.2919
7	0.3041
8	0.3190
9	0.3445
10	0.3637
11	0.3831
12	0.4002
13	0.3711
14	0.3781
15	0.4094
16	0.4209
17	0.4072
18	0.4363
19	0.4594
20	0.4210
21	0.4226
22	0.4484
23	0.4491
24	0.4582
25	0.4393
26	0.4449

27	0.4672
28	0.4547
29	0.4712
30	0.4678
31	0.4710
32	0.4629
33	0.4600
34	0.4580
35	0.4592
36	0.4353
37	0.4423
38	0.4556
39	0.4605

From the analysis of these two tests, we decided to use  $k = 21$  since it was what we considered the best compromise as number of clusters, cost function and silhouette score.

Using this parameter we got this cluster distribution, that shows an acceptably even distribution of elements inside every cluster:

Cluster	Number of Elements
1	291.0
2	313.0
3	704.0
4	770.0
5	312.0
6	836.0
7	259.0
8	266.0
9	402.0
10	235.0
11	351.0
12	196.0
13	171.0
14	546.0
15	294.0
16	159.0
17	542.0
18	283.0
19	1524.0
20	398.0

21	305.0
----	-------

**Agglomerative Clustering:** Hierarchical clustering is a general family of clustering algorithms that build nested clusters by merging or splitting them successively. This hierarchy of clusters is represented as a dendrogram. The root of the tree is the unique cluster that gathers all the samples, the leaves being the clusters with only one sample.

We decided to use the Agglomerative Clustering offered by the Python library scikit-learn which offers a bottom up hierarchical clustering approach.

In order to select the best value of  $k$ , we had to generate the dendrogram of our dataset which showed a good level of separation with a cut that generates 20 clusters (Fig.8).

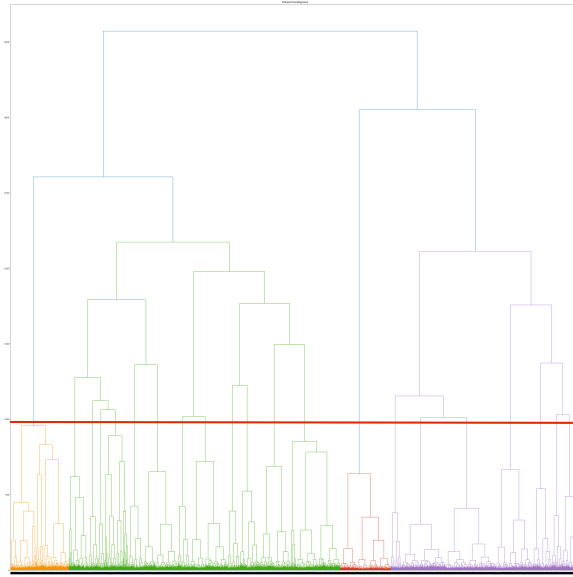


Figure 8: Dendrogram

After checking the dendrogram, we decided to compute the silhouette score of this clustering approach, in order to find if our decision could create some good clusters (Fig.9).

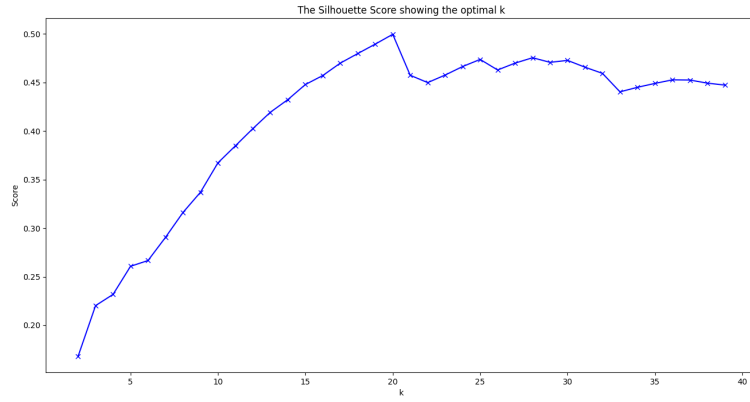


Figure 9: Silhouette Score Plot

K Value	Silhouette Score
2	0.1680
3	0.2202
4	0.2317
5	0.2607
6	0.2666
7	0.2905
8	0.3161
9	0.3369
10	0.3672
11	0.3847
12	0.4025
13	0.4193
14	0.4323
15	0.4478
16	0.4571
17	0.4698
18	0.4797
19	0.4894
20	0.4995
21	0.4573
22	0.4498
23	0.4576
24	0.4664
25	0.4737
26	0.4628

27	0.4699
28	0.4754
29	0.4707
30	0.4727
31	0.4656
32	0.4593
33	0.4403
34	0.4449
35	0.4490
36	0.4526
37	0.4524
38	0.4492
39	0.4472

After checking these scores, we confirmed our initial hypothesis, so we decided to use  $k=20$ . This choice produced in this case, a mostly balanced distribution of elements inside every cluster:

Cluster	Number of Elements
1	1524.0
2	935.0
3	819.0
4	385.0
5	790.0
6	305.0
7	315.0
8	613.0
9	551.0
10	596.0
11	494.0
12	285.0
13	235.0
14	328.0
15	196.0
16	171.0
17	159.0
18	104.0
19	158.0
20	194.0



**Algorithm Choice:** From the analysis of these two algorithms, we decided that the best one was the kmeans since, using similar values of k ( $K=20$ ,  $K=21$ ), the silhouette score was very similar ( $\sim 0,45$ ), the execution time were mostly the same ( $\sim 1$  second) and the cluster distribution of the elements in the clusters was overall better in the kmeans algorithm.

We decided to opt with the kmeans algorithm since it could perform better in case of bigger datasets, (since the dendrogram creation is very computationally heavy), and is easier to implement.

### 2.3.4 Recommendation System

A recommendation system is an algorithm that gives recommendations for items that is most interesting to a user. Recommendations is related to many kinds of real applications, such as what commodities are purchased, what songs is listened, or what latest news is read. In order to create an effective recommendation system, three approaches can be used:

- Collaborative filtering: uses the collaborative power of the available assessment by users to make recommendations. It assumes that different users can rate an anime similarly to other users or have similar behavior.
- Content-based filtering uses the features of an anime to make recommendations. In our case, an anime feature is represented by its genres. This means that anime with similar features will be included in the same suggestion.
- Hybrid: combines both methods described above.

Since this last approach combines the best features of the first two, we decided to use this one. In particular, we decided to use the user's preferences (collaborative approach) and combine these with the anime genres (content-based filtering) used in form of a genre list, used to identify the users' taste.

The recommendation system process works like this:

- The clustering model is built using the KMeans algorithm.
- A new user is associated to the existing cluster having the nearest centroid (to compute the distance, both cosine similarity and euclidean distance has been tested).
- All users belonging to the same cluster as the selected user are grouped and all their anime list are selected.
- If the anime presents a score higher than 6, meaning that that user has liked that particular anime, the anime is selected and added to a list.
- the list groups the most popular anime in terms of number of presences in every user list.

- the top ten anime retrieved using this method are suggested to the selected user as recommended anime.

In order to test the performances of this system, we performed a small number of experiments: we asked to a group of friends with different tastes to create a new account and to list their preferences in the initial form presented at the registration phase and to evaluate the recommendation provided by the application (note that a newly registered user has a value of 10 to every selected genre).

The distance metrics considered to associate a new user to a cluster are the euclidean distance and the cosine similarity. We tested both in order to find the best one.

The experiments' results are showed in the table below:

User	Genres Liked	Anime of Interest (Euclidean)	Anime of Interest (Cosine)
alex312	Comedy, Demons, Drama	7/10	7/10
ch1@ra	Adventure, Historical, Romance, Thriller	5/10	7/10
carl892	Action, Adventure	7/10	8/10
D@niel74	Mystery, Police, Psychological, Seinen, Thriller	3/10	3/10
LuCiA97	School, Shoujo, Shounen, Slice of Life	5/10	9/10
NickH014	Action, Adventure, Comedy, Demons, Super Power, Supernatural	8/10	8/10
Paul87	Action, Mecha, Military, Sci-Fi, Space	7/10	7/10
ro54	Game, Historical, Magic, Music, Parody	5/10	5/10
Stefy31	Demons, Fantasy, Game, Horror, Magic, Mystery, Seinen, Super Power, Supernatural	8/10	6/10
Tommy99	Adventure, Comedy, Fantasy, Game, Historical, Magic, Mecha, Music, Mystery, Sci-Fi, Slice of Life, Space, Super Power, Supernatural	10/10	10/10

Analyzing the results obtained we found that both euclidean distance and cosine similarity offered good performances in terms of cluster assignment, but the cosine similarity offered in some cases a slightly better assignment than the euclidean distance.

At the end we decided to opt for the cosine similarity approach since we thought that this metric would offer better performances with less characterized users, since the cosine similarity performs better in case of sparse vectors.

### 2.3.5 Conclusions

In the end, we can say that we are satisfied with the results obtained by our suggestion system, since in most case every user received at least the 50% of interesting suggestions.

As we expected, the suggestion system performs better in two cases:

- if the user has many genre preferences, since it's easier for the program to suggest interesting anime.
- if the user has related genre preferences (e.g. user Paul) since people with common interests are more likely to be clustered together.

While the system presents worse performances in case of dishomogeneous genres' preferences (e.g. user Daniel).

Finally we think the system could be further improved using different kind of suggestion systems (k-nearest neighbors to find users close to the one who's requesting the suggestions) or using different clustering approaches.

Another chance to improve further the system could be to use a bigger dataset with even more characterized users that could help improving the clustering of the data.

Aside from these consideration though, we can say that we are satisfied with the results obtained by our suggestion system, since in most case every user received at least the 50% of interesting suggestions, meaning that the system can provide good results.

## 3 Application Code

### 3.1 Class organization

The application is written in Java. The code is available on the github server <https://github.com/Stepnod98/data-mining-project-animebook.git>, in the folder animebookdm. The class are divided in the following modules, based on their functionality in the application:

- **Layout:** contains the classes which implement the GUI.
- **Controller:** contains the classes which manage the operations which the user inputs.
- **DBManager:** contains the class which manages the operations of the document database.
- **Entities:** contains the classes which describe the entities of the application.
- **Datamining:** contains the classes which implements the clustering and recommendation algorithm. .
- **Utils:** contains some utility class.

### 3.2 Layout

- **AnimeLayout:** implements the layout for the area of the application dedicated to search and viewing available animes. Main methods:
  - public void updateBrowseResults(List<String> result): updates the result showed when browsing an anime.
  - public void showAnimeResults(List<String> list): creates a table to display the animes
  - public void showAnimeFindResults(String title, int episodes, int members, int score): creates a box in which information about the anime found with the FIND button are displayed
  - public void clearLayout(): clears the tables.
- **AnimeListLayout:** layout of the area of the application dedicated to viusalizing and managing your anime list. Methods: Constructor, get methods.
- **AppLayout:** layout of the menu. Methods: Constructor, get methods.

- **GenreSelection:** layout of the area from which an user can select his favourite genres. Main methods:

-private void addSelection(): initializes the genre selection layout with the checkboxes for each available genre.

- **LoginLayout:** layout of the login area. Methods: Constructor, get methods, printError(for showing errors in the login phase, such as wrong username or password)
- **SignUpLayout:** layout of the sign up area. Methods: Constructor, get methods.

### 3.3 Controller

- **AnimeManager:** controller for the operation of finding, browsing and visualizing information about animes. Main methods:

-public void viewAnimes(): takes the list of all the available animes and outputs them through the GUI

-public void findAnime(String inTitle): outputs informations about the anime with the given title

-public void viewRecommendedAnimes(): starts the recommendation tasks and outputs its results

-public static void addAnime(String title, int score): given the anime title and the score the user gave in input, adds it to the anime list of the user (calling the database manager)

-public void setEvents(): sets the events of the buttons and textfield

-private void setBrowseEvents(): sets the events of the browsing task

-private void browseTasks(): updates browsing result in case of changes of the text or of the focus

-public void setTableEvents(): sets the events on the table elements

-private void setAnimeBox(String title): open a box with the information of an anime

-private boolean isNumeric(String str): check if the text in the score textfield is a number or not

- **AnimeListManager:** controller for the anime list section. Main methods:

-public static void updateScore(String title, int newScore): updates the score for the anime with title title, in the corresponding row of the anime list, with the score newScore.

-public static void removeAnime(String title, int score): removes the anime with title title and score score from the animelist.

-public void setEvents(): sets table events.

- **AppLayoutManager:** set the events of the menu.
- **GenreSelectionManager:** used for selecting the favourite genres of an user. Main methods: -public static void confirmRegistration(): inserts the profile in the database with the specified genres.

-public static void confirmEdit(): edits the genres selected for the user's profile.

-public static void setEvents(): sets the events of the buttons.

- **GUIManager:** manages the flow of the application setting up the GUI. Main methods:

-public static void openAppManager(): opens the main menu.

-public static void openAnimeManager(): opens the anime management and search section.

-public static void openAnimeList(): opens the anime list of the user.

-public static void openLoginManager(): opens the login section.

-public static void openSignUpManager(): opens the registration form.

-public static void openGenreSelection(): opens the genre selection form.

-public static void openGenreSelectionEditor(): opens the genre editing form.

-public static void addNode(Node node), public static void removeNode(Node node): for managing node elements on the GUI.

-public static void clearAnimeBoxes(): cleans the screen from previous anime information boxes.

-public static Group setUI(): initial setting of the GUI.

-public static String getCurrentUser(): returns the username of the current user.

-public static User getCurrent(): returns the User instance relative to the current user.

-public static void setCurrentUser(User user): sets the current user of the application.

-public static void main(String args[]): starts the application.

- **GUIStarter:** starts the GUIManager.

- **LoginManager:** manages login operations and controls over the credentials. Main methods:

-public static void login(): checks the credential and logs in the user if they are correct.

-public static void signup(): calls the GUIManager to open the registration form.

-private static String encrypt(String pass): encrypts with SHA1 the password inserted from the user.

-public static void setEvents(): sets button events.

- **SignUpManager:** manages the sign up operations, checking on the credentials given for signing a new user. Main methods:

-private void setEvents(): sets button events.

-public void checkCredentials(): checks if the credentials inserted from the unregistered user are valid.

-private void addUser(): registers the user with the credentials inserted.

-private static String encrypt(String pass): encrypts with SHA1 the password inserted from the user.

-private boolean checkUsername(): checks if the username inserted is valid.

-private boolean checkEmail(): checks if the username inserted is valid.

### 3.4 DBManager

- **MongoDBManager:** contains all the operations for managing the application's database. Main methods:
  - public static void connectionStarter(String connection, String database): starts the connection to MongoDB.

-public static User findUser(String username): finds a user, given his/her username.

-public static User findUserbyUserID(int uid): finds a user, given his/her user id.

-public static boolean removeAnimeListElement(String userName, String animeTitle): deletes an animelist element from the corresponding user.

-public static boolean updateAnimeScore(String userName, String animeTitle, int score): updates an anime score from a user animelist.

-public static boolean checkUser(String username, String pwd): checks user's credentials.

-public static boolean checkEmail(String email): checks if the email is already present.

-public static List<AnimeListElem> getAnimeList(): returns the animelist of the current user.

-public static List<AnimeListElem> getUserAnimeList(String username): returns the anime list of the specified user.

-public static List getAnimes(): returns a list with all available animes.

-public static boolean storeAnimeListElement(String userName, String animeTitle, int score): stores a new animelist element to the correspond-



ing user.

-public static boolean duplicatesChecker(User target, String animeTitle): checks if an anime is already present inside a user animelist.

-public static List findAnimeList(String inTitle): returns a list with anime titles containing the expression specified in input.

-public static int getAnimeEps(String inTitle): returns the number of episodes of the specified anime.

-public static int getAnimeUserScore(String inTitle): returns the users' score of the specified anime.

-public static int getAnimeMembers(String inTitle):: returns the number of people who inserted in their list the specified anime.

-public static void removeFromMal(String title): removes the specified anime, if present, from the user's list.

-public static void updateScore(String animeTitle, int score): updates the score of the anime with the title specified with the score score.

-public static void addUser(User user): adds the user user to the database

-public static void insertProfile(User user): inserts the profile of the user in the profiles section

-public static void updateGenres(List<Genre> genres): updates the user's genres

-public static boolean checkAnime(String inTitle): checks if an anime with the specified title exists

-public static boolean checkAnimeinList(String inTitle): check if the anime with the specified title is in the user's list

-public static List<Genre> getGenres(String title): returns the genres of an anime

-public static List<Genre> getUserGenres(): returns the user's favourite genres

-public static void connectionCloser(): close MongoDB connection

### 3.5 Entities

- **Anime:** for displaying info about an anime found by an user. Main methods:
  - Constructors: public Anime(String title, List genres, int episodes, int members, double score); public Anime(Document doc);
  - Get methods (getTitle, getMembers, getEpisodes, getScore).
- **AnimeListElem:** defines the elements stored in an anime list.
- **Genre:** defines the genres, with the name and the frequency count for each genre.
- **User:** contains information about users, their username, their animelist and their favourite genres. Main methods:
  - public double[] getGenresFreq(): returns the sorted counters for each genre of the user's favourite genres

-public ArrayList<AnimeListElem> getAnimeListfromDoc(ArrayList<Document> animeListDoc): returns the anime list of the user from a document object

-public ArrayList<Document> removeAnimeListElement(String animeTitle): removes an element from a user's anime list

-public ArrayList<Document> addNewAnimeListElement(AnimeListElem e): adds a new element to the user's anime list

-public ArrayList<Document> setAnimeScore(String animeTitle, int score): sets the score of the specified anime with the specified value

-public ArrayList<Document> createAnimeList(ArrayList<AnimeListElem> list): creates an animelist from the list specified

-public void addAnimeGenres(List<Genre> animeGenres): add the genres of the anime inserted to the genre list

-public void removeAnimeGenres(List<Genre> animeGenres): removes the genres of the anime inserted from the genre list

### 3.6 Datamining

- **Clustering:** class in charge of the clustering task. Main methods:
  - public static int[] kmeansAssignment (double[] elementCoords) throws Exception: loads the CSV Dataset, removes unnecessary attributes, creates a new instance of clusterer, preserves the order of every instance, builds the clusterer, creates hashmap with all centroids coordinates and retrieves cluster elements.
- **Recommender:** class used to exploit the recommendation task. Main methods:
  - public static List<String> getMostPopularAnime(int userList[]) given the ids of the users belonging to a certain cluster, counts most popular anime belonging to a cluster and returns them.
  - public static List<String> animeCounter(ArrayList<User> clusterUserList): given all the users belonging to a certain cluster, returns the most popular animes among their lists.

### 3.7 Utils

- **AnimeListRow:** defines the row of the user's anime list table, consisting on the anime title, the score given by the user, the textfield for updating the score and the UPDATE and REMOVE buttons.
- **AnimeRow:** defines the row of the anime management table, consisting on the anime title, the score textfield and the ADD and VIEW buttons.
- **Utility:** methods and parameters used by the clustering process. Main methods:
  - public static double cosineSimilarity(double[] vectorA, double[] vectorB): computes the cosine similarity given two vectors of doubles
  - public static int assignCluster(double[] coordinates, HashMap<Integer, double[]> centroids): assigns the object to a cluster and returns the cluster id
  - public static int[] findClusterUsers(int[] clusterElements, double size, int clusterNumber): gets all elements from that cluster, defined by clusterNumber, clusterElements and size.

### 3.8 Python Modules:

These modules have been used to evaluate which clustering algorithm could be better suited for our project:

- Hierarchical Clustering.py: Includes all method used to build the dendrogram and the silhouette graph for the Agglomerative Clustering.
- KMeans Clustering.py: Includes all method used to plot silhouette and elbow graphs for the KMeans Clustering Algorithm.

## 4 User Manual

### 4.1 Login

The user, if registered, has to input their credentials to log to the application and use it; if the credentials are wrong an error message will show up

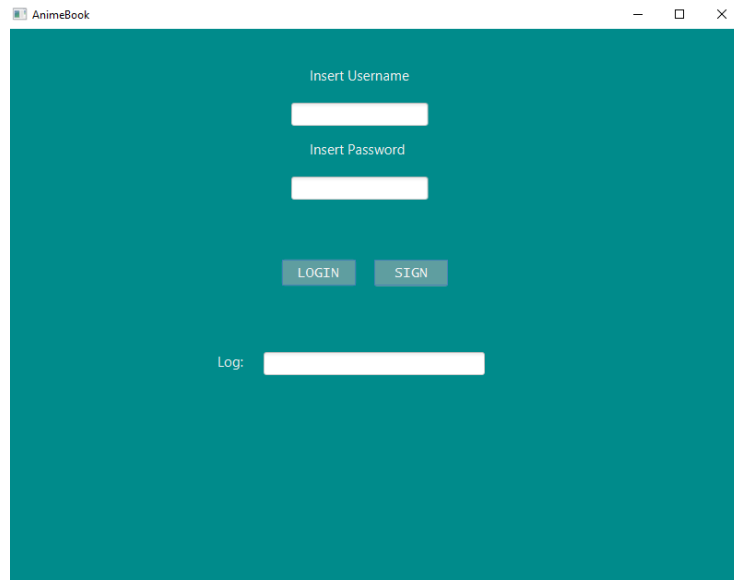
A screenshot of a web application window titled "AnimeBook". The window has a teal background. In the center, there is a login form. It consists of two white input fields: the top one is labeled "Insert Username" and the bottom one is labeled "Insert Password". Below these fields are two buttons: "LOGIN" and "SIGN". At the bottom of the form, there is a label "Log:" followed by a white input field.

Figure 10: Login interface

### 4.2 Signing Up

An unregistered user can create an account if click on the sign up button; then he/she has to give their credentials; an error message will show up if there is some error (for instance username already used, password not valid or different from the confirm password field).

A screenshot of a web application window titled "AnimeBook". The window has a teal background and contains a sign-up form. The form has the following fields and labels:

- First Name:
- Last Name:
- Insert Email:
- Insert Password:
- Confirm Password:
- Insert Username:
- Log:

There is a "SIGN UP" button below the "Insert Username" field and a "BACK" button in the bottom right corner.

Figure 11: Sign up interface

A screenshot of the same "AnimeBook" web application window, but now showing a password error. The form fields are filled with the following data:

- First Name: Andrea
- Last Name: Di Marco
- Insert Email: adimarco@ab.com
- Insert Password: ...
- Confirm Password: ...
- Insert Username: adimarco
- Log:

The "SIGN UP" button is still present. A red error message "The password is too short!" is displayed next to the "Log:" field. The "BACK" button remains in the bottom right corner.

Figure 12: Sign up error: password too short

The screenshot shows a web browser window titled "AnimeBook". The sign-up form has the following fields and values:

- First Name: Andrea
- Last Name: Di Marco
- Insert Username: adimarco
- Insert Email: adimarco@ab.com
- Insert Password: (masked with dots)
- Confirm Password: (masked with dots)

A red error message "The passwords are different!" is displayed next to the Confirm Password field. A "SIGN UP" button is at the bottom left, and a "BACK" button is at the bottom right.

Figure 13: Sign up error: the inserted passwords are different

An user, after signing up, can input their favourite genres clicking in the apposite check boxes

The screenshot shows a web browser window titled "AnimeBook". The genre selection interface has the title "Select your favorites genres" and a grid of 21 checkboxes for different anime genres:

- Action
- Adventure
- Comedy
- Dementia
- Demons
- Drama
- Fantasy
- Game
- Historical
- Horror
- Magic
- Mecha
- Military
- Music
- Mystery
- Parody
- Police
- Psychological
- Romance
- School
- Sci-Fi
- Seinen
- Shoujo
- Shounen
- Slice of Life
- Space
- Sports
- Super Power
- Supernatural
- Thriller

At the bottom, there are "CONFIRM" and "BACK" buttons.

Figure 14: Genre selection interface

A registered and logged user can have access to the application through this main menu; the options are: "YOUR ANIME LIST", which will open the user's anime list, "SEARCH ANIMES", for searching the animes that a user has seen and wants to add to its list, "EDIT GENRES" to edit their favourite genres, and "LOGOUT" to exit from the application

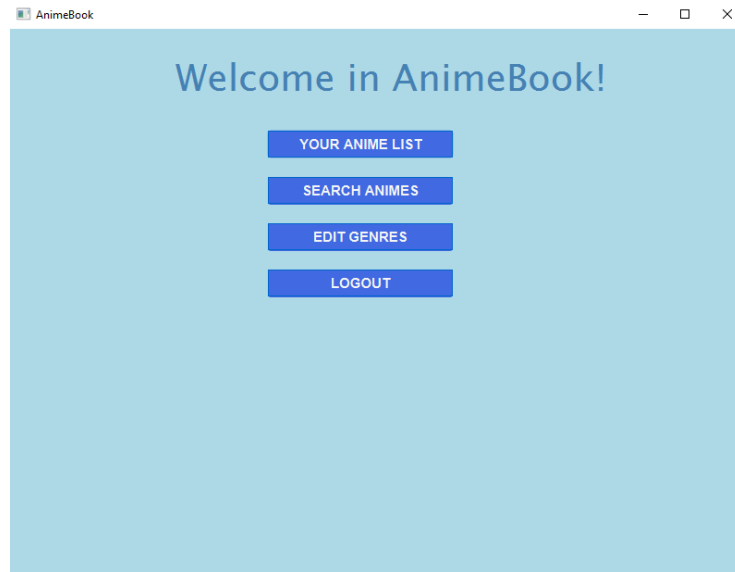


Figure 15: Main menù

### 4.3 Your Anime List

A user can view its anime list and managing the animes he saved in it; if the user wants he/she can remove a specific anime from the list, clicking the REMOVE button in the corresponding row, or update the score of one specific anime clicking the UPDATE button in the corresponding row, after inserting a score in the score textfield



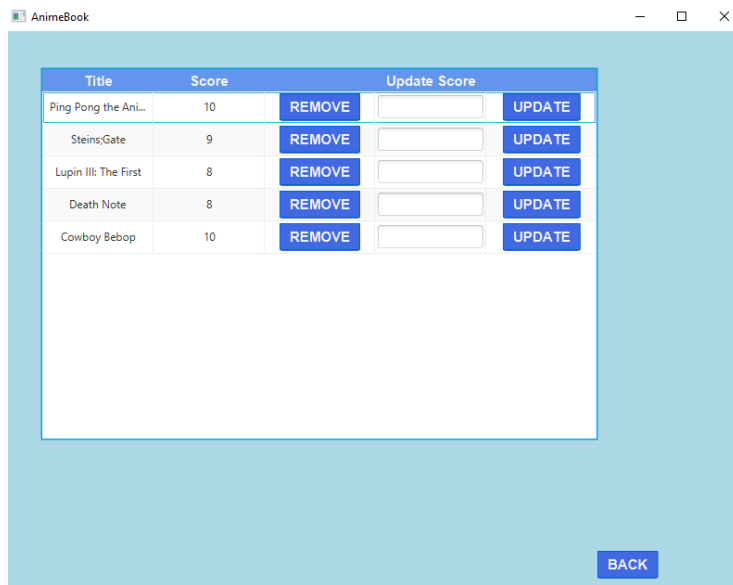


Figure 16: Anime List layout

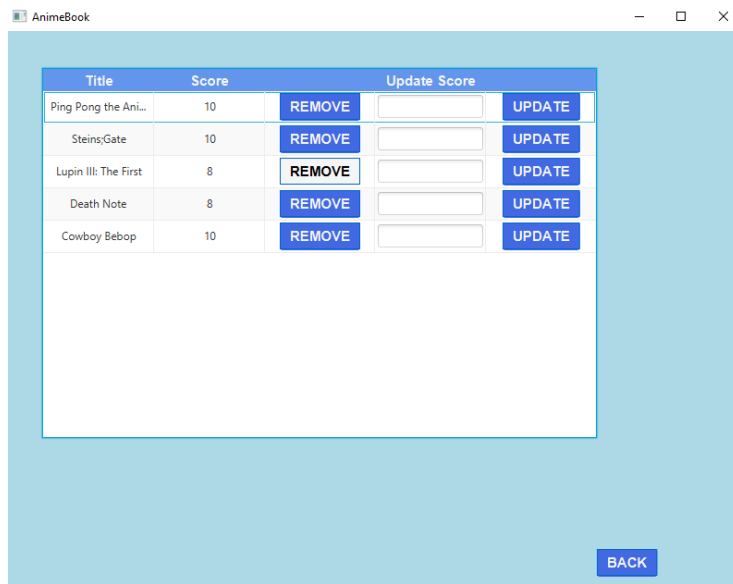


Figure 17: Selecting anime to remove

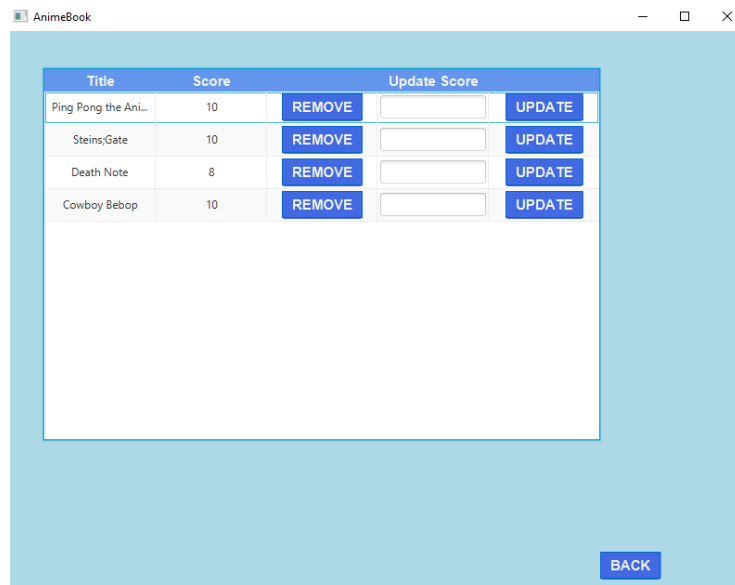


Figure 18: Anime selected removed

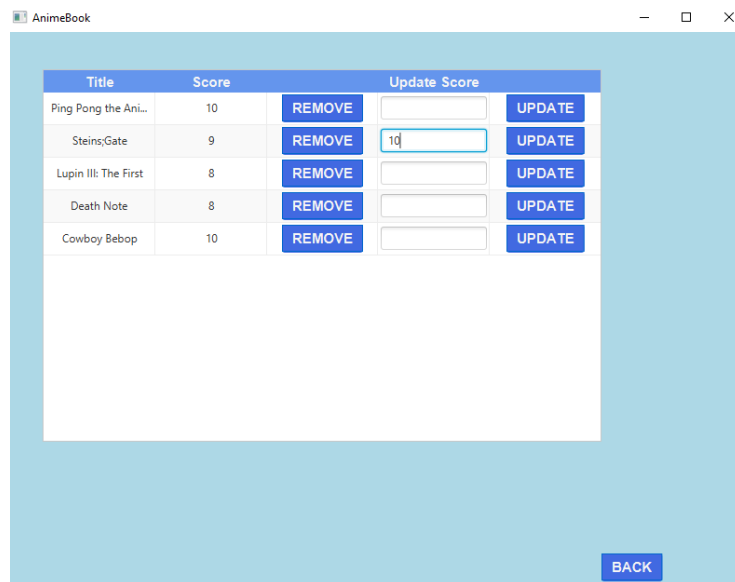


Figure 19: Setting the new score of an anime in the list

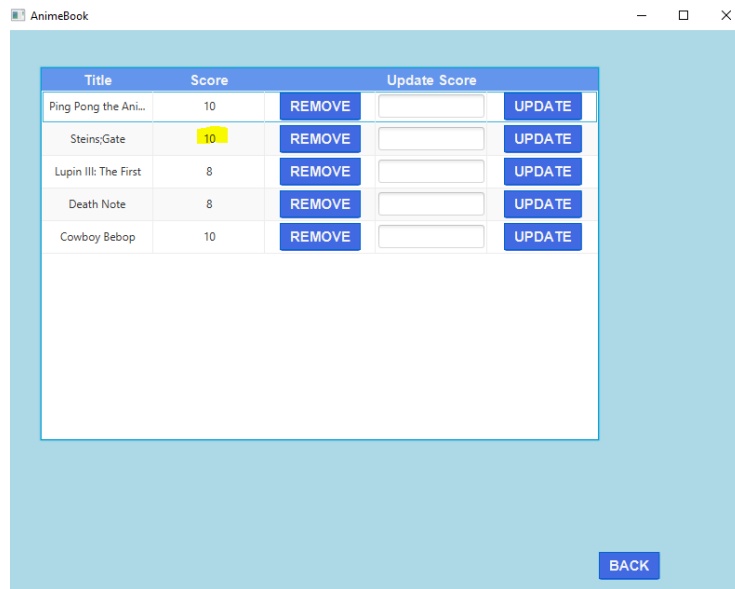


Figure 20: Score updated

#### 4.4 Search Animes

A user can click on the VIEW ALL ANIMES button to view all the available animes; then he/she can view the information about one specific anime or adding it directly to its list after inserting a score

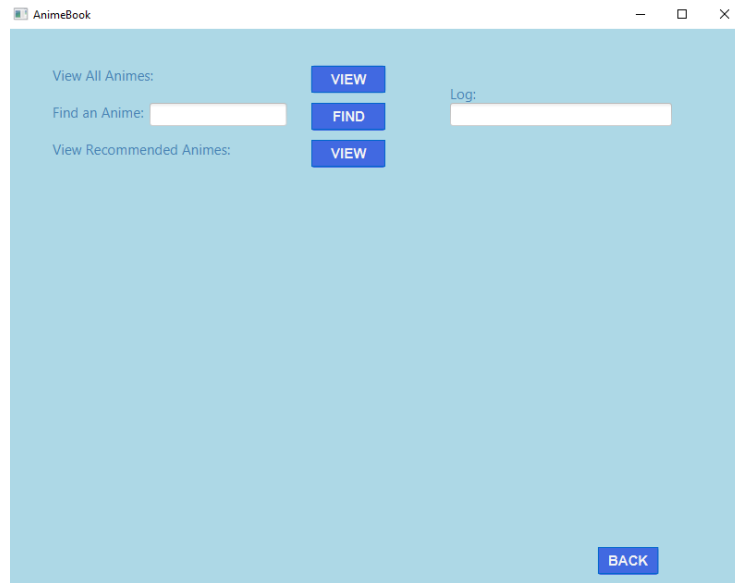


Figure 21: Anime search area interface



Figure 22: Table with all the animes available;

AnimeBook

View All Animes:

Find an Anime:

View Recommended Animes:

Log:

TITLE	YOUR SCORE			
Peter Pan no Bouken Speci...	<input type="text"/>	<input type="button" value="ADD"/>	<input type="button" value="VIEW"/>	
Nana	<input type="text"/>	<input type="button" value="ADD"/>	<input type="button" value="VIEW"/>	
Mobile Suit Gundam II: Sol...	<input type="text"/>	<input type="button" value="ADD"/>	<input type="button" value="VIEW"/>	
Wagaya no Oinari-sama. S...	<input type="text"/>	<input type="button" value="ADD"/>	<input type="button" value="VIEW"/>	
Mobile Suit Zeta Gundam	<input type="text"/>	<input type="button" value="ADD"/>	<input type="button" value="VIEW"/>	
Megalo Box	<input type="text" value="10"/>	<input type="button" value="ADD"/>	<input type="button" value="VIEW"/>	
Ousama no Takaramono	<input type="text"/>	<input type="button" value="ADD"/>	<input type="button" value="VIEW"/>	
Naruto Narutimate Hero 3...	<input type="text"/>	<input type="button" value="ADD"/>	<input type="button" value="VIEW"/>	
Null Peta	<input type="text"/>	<input type="button" value="ADD"/>	<input type="button" value="VIEW"/>	

Figure 23: Inserting a score for an anime in the table

AnimeBook

View All Animes:

Find an Anime:

View Recommended Animes:

Log:

TITLE	YOUR SCORE			
Peter Pan no Bouken Speci...	<input type="text"/>	<input type="button" value="ADD"/>	<input type="button" value="VIEW"/>	
Nana	<input type="text"/>	<input type="button" value="ADD"/>	<input type="button" value="VIEW"/>	
Mobile Suit Gundam II: Sol...	<input type="text"/>	<input type="button" value="ADD"/>	<input type="button" value="VIEW"/>	
Wagaya no Oinari-sama. S...	<input type="text"/>	<input type="button" value="ADD"/>	<input type="button" value="VIEW"/>	
Mobile Suit Zeta Gundam	<input type="text"/>	<input type="button" value="ADD"/>	<input type="button" value="VIEW"/>	
Megalo Box	<input type="text" value="10"/>	<input type="button" value="ADD"/>	<input type="button" value="VIEW"/>	
Ousama no Takaramono	<input type="text"/>	<input type="button" value="ADD"/>	<input type="button" value="VIEW"/>	
Naruto Narutimate Hero 3...	<input type="text"/>	<input type="button" value="ADD"/>	<input type="button" value="VIEW"/>	
Null Peta	<input type="text"/>	<input type="button" value="ADD"/>	<input type="button" value="VIEW"/>	

Figure 24: Adding an anime from the search table

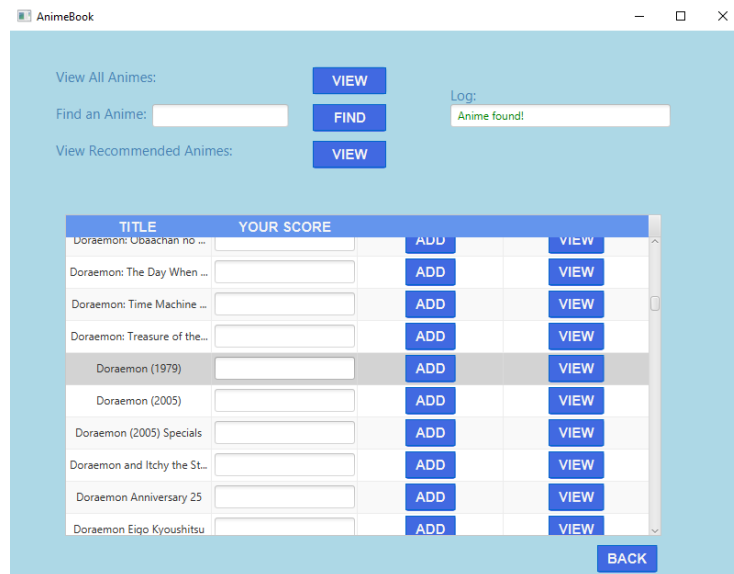


Figure 25: Selecting an anime in the table

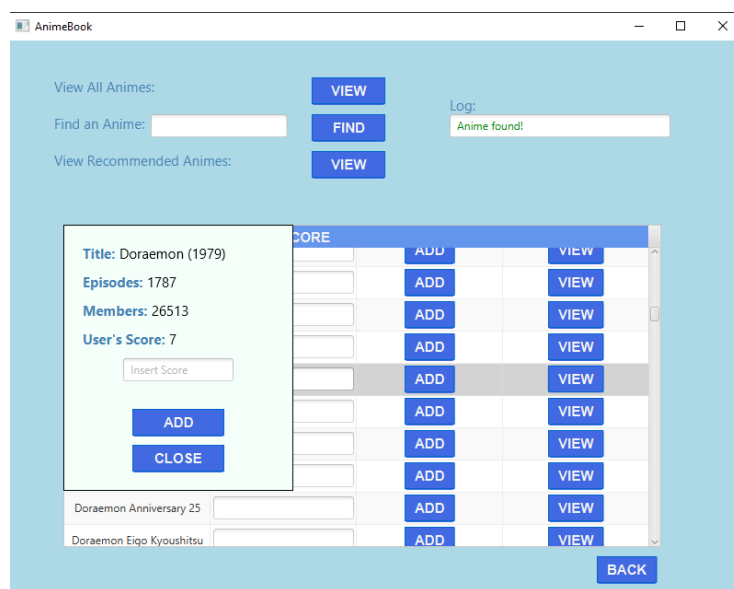


Figure 26: Showing anime information by clicking the VIEW button in the corresponding row in the table

The user can also directly write an anime he wants to find and browse

through them and view informations about one specific anime; then he/she can add it to its list after inserting a score

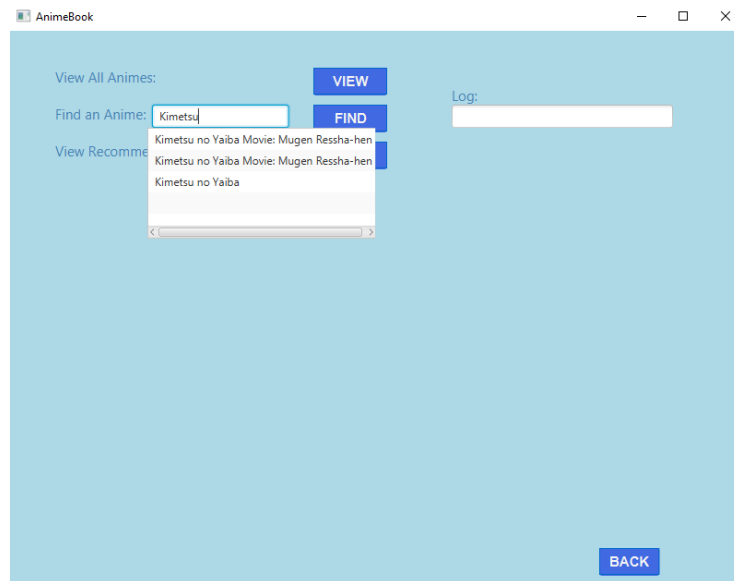


Figure 27: Browsing animes by writing in the textfield

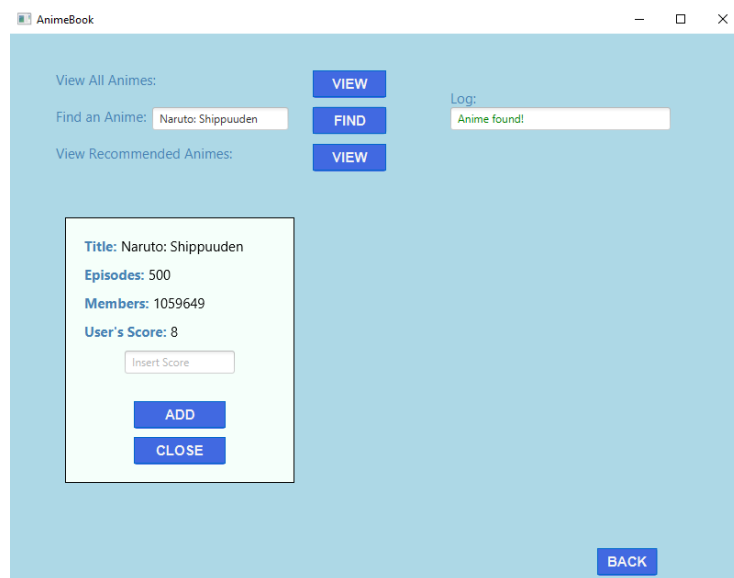


Figure 28: Anime informations displayed after clicking on the FIND button

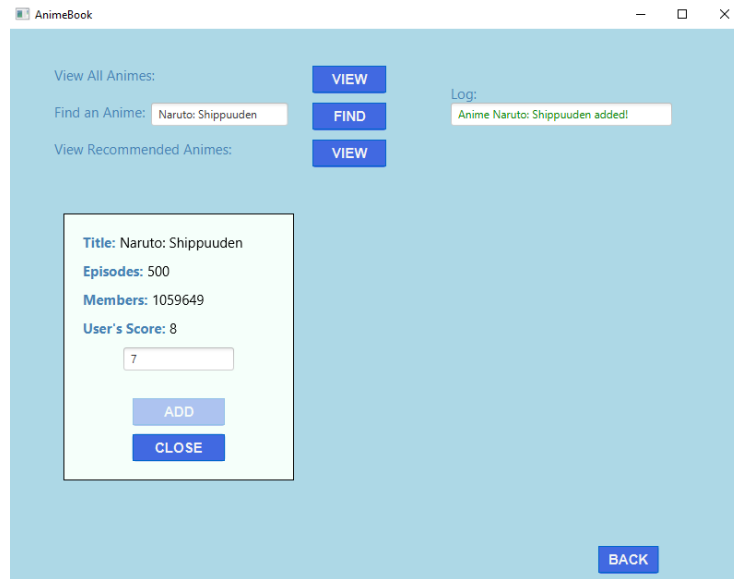


Figure 29: Adding an anime to the list from the box with anime's informations

A user can click the VIEW RECOMMENDED button to see a list of animes which are recommended for the user based on the animes he/she put in his/her list and his/her favourite genres, compared to the ones of other users.

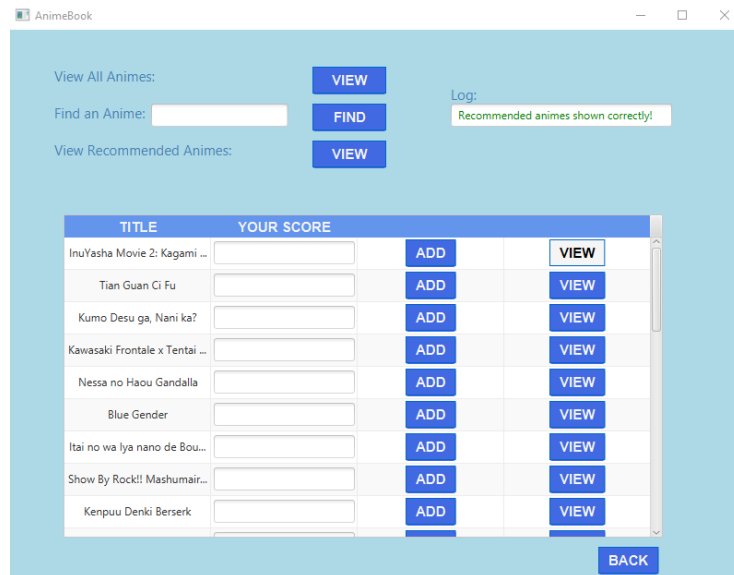
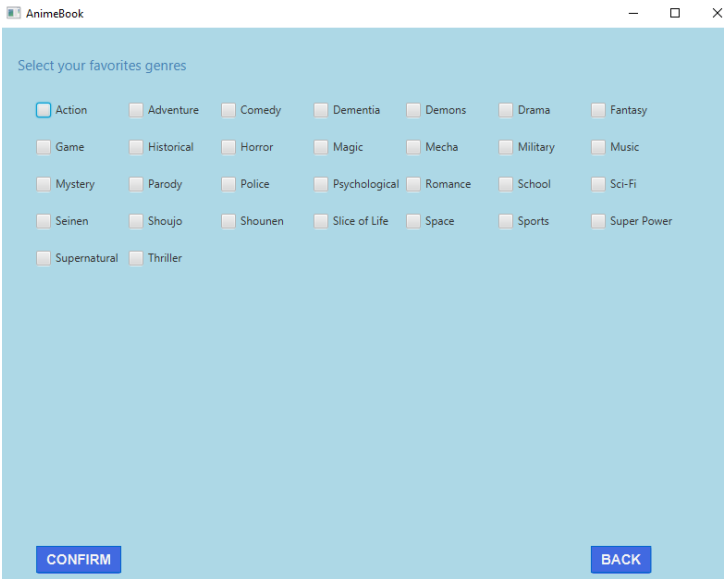


Figure 30: Anime recommendations given by the system



## 4.5 Editing favourite genres

An user can update his favourite genres having access again to the genre selection screen, where he can click on the checkboxes with his favourite genres



The screenshot shows a web application window titled "AnimeBook". Inside, there is a section titled "Select your favorites genres". Below this title, there is a grid of 21 checkboxes, each followed by a genre name. The genres are: Action, Adventure, Comedy, Dementia, Demons, Drama, Fantasy, Game, Historical, Horror, Magic, Mecha, Military, Music, Mystery, Parody, Police, Psychological, Romance, School, Sci-Fi, Seinen, Shoujo, Shounen, Slice of Life, Space, Sports, Super Power, Supernatural, and Thriller. The "Action" checkbox is checked. At the bottom of the interface, there are two buttons: "CONFIRM" and "BACK".

Genre	Selected
Action	Yes
Adventure	No
Comedy	No
Dementia	No
Demons	No
Drama	No
Fantasy	No
Game	No
Historical	No
Horror	No
Magic	No
Mecha	No
Military	No
Music	No
Mystery	No
Parody	No
Police	No
Psychological	No
Romance	No
School	No
Sci-Fi	No
Seinen	No
Shoujo	No
Shounen	No
Slice of Life	No
Space	No
Sports	No
Super Power	No
Supernatural	No
Thriller	No

Figure 31: Genre editing interface