UNIVERSITÀ DI PISA

LARGE SCALE AND MULTI-STRUCTURED DATABASES

< SOCIAL WINE APPLICATION REPORT >

Github repository (Private) **: https://github.com/DomenicoArmillotta/SocialWine**

Project made by**:**

- **Bashar Hayani -** badge #: 000000 - email: b.hayani@studenti.unipi.it
- **Domenico Armillotta -** badge #: 643020 - email: d.armillotta@studenti.unipi.it
- **Leonardo Bellizzi -** badge #: 643019 - email: l.bellizzi@studenti.unipi.it

OVERVIEW

INTRODUCTION AND REQUIREMENTS 3
- FUNCTIONAL REQUIREMENTS 3
- NON FUNCTIONAL REQUIREMENTS 4

DATASET 5
- DOCUMENT PREPROCESSING 5
- DATA TRANSFER TO THE TWO DATABASES 5

ACTORS AND USE CASE 6

UML CLASS DIAGRAM 7
- JAVADOC 8

DATA MODELLING 9
- MONGO DB 9
  - INTRODUCTION 9
  - DOCUMENTS ORGANIZATION AND STRUCTURE 9
  - ADVANCED QUERIES IMPLEMENTATIONS 10
  - IMPLEMENTATION DETAILS 10
  - IMPACT OF INDEXES IN MONGODB QUERIES 11
  - MONGO OPERATIONS ANALYSIS 13
  - MONGODB REPLICAS 14
  - SHARDING 14
- NEO4J 15
  - INTRODUCTION 15
  - GRAPH DESIGN 18
  - QUERIES IMPLEMENTATIONS 19
  - IMPLEMENTATIONS DETAILS 21
- DATA AMONG DATABASES 22
- DISTRIBUTED DATABASE CAP THEOREM 23

ARCHITECTURAL DESIGN 24

OTHER IMPLEMENTATION DETAILS 25
- SCRAPER 25
- MENU 25

INTRODUCTION AND REQUIREMENTS

The application that has been developed is called “Social Wine”. The goal is to create a social network focused on wine, once registered the user can explore the circulating wines, leave a comment for each wine, like the comments of other users and follow his friends so as not to miss the future reviews.
Among the users there is some admin who can make statistics on the social network and also has the task of moderator, in fact he can delete comments and wines and ban users. Moreover an Admin has the possibility to elect another social wine’s user as Admin.
The social network also has recommendation functions both to recommend friendships and to recommend wine to each user based on their interactions, this mechanism can be used for advertising on the social network.
Another important feature of the project is a scraper that works continuously in the background and updates the database, to ensure that the data is always up to date with most recent reviews that are in Winemag.

FUNCTIONAL REQUIREMENTS

Features offered to the Unregistered User

- Registration
  To access the app, the user must log in or register as a new user, otherwise they will not be able to use the app

Features offered to the Registered User

- Login/Logout
- Search specific Wine
- Browse suggested Wine
- Browse All Wines
- Browse the comment of friends and put/delete Like (Homepage)
- Write a comment on a wine
- Delete his comment
- Like a comment
- Unlike a comment
- Browse trending comment on the social and put like (Homepage)
- See your own profile
- Delete your own account
- Browse the following people and unfollow
- See a friend's profile
- Search specific user and follow/see profile

- Browse suggested user and follow or see profile.Features offered to the Admin in addition to user operations:
- Browse all
- Users and ban user
- Add a wine
- Browse all wines and delete a wine
- Browse all comments and delete
- Make statistics
- Elect another user as Admin

NON FUNCTIONAL REQUIREMENTS

- Data Consistency: All the users must see the last version of the data and the update operations must be performed in the same order in which they are issued.
- Performance: A low response time is necessary in order to avoid too long waits for the user.
- Reliability : The application must never crash, all critical points must be managed with exceptions
- Usability: The website's interface has to be user-friendly and easy to use.

DATASET

The data were taken from "https://www.kaggle.com/zynicide/wine-reviews". Dataset is composed by the following attributes:

- Wine attributes: country, designation, price, province, region_1, region_2, title (wineName), variety, winery;
- User attributes: taster_twitter_handle, taster_name;
- Review attributes: description, score

DOCUMENT PREPROCESSING

The original json counts about 130 thousand reviews but for application it has been decided to truncate it, because some reviews or some wines attributes had null values.
With the constant use of scraper, over a period of time, the collection will have increasing reviews but always with the criteria that reviews having null values will not be inserted.
If in the original json, there are two wines with the same name and different features like price, country…, in the transformed json will be inserted only one document and the attributes will be taken from last updated wines.
Two more attributes for the user were added like: user_country and email. For users coming from json and scraper they will be "None".

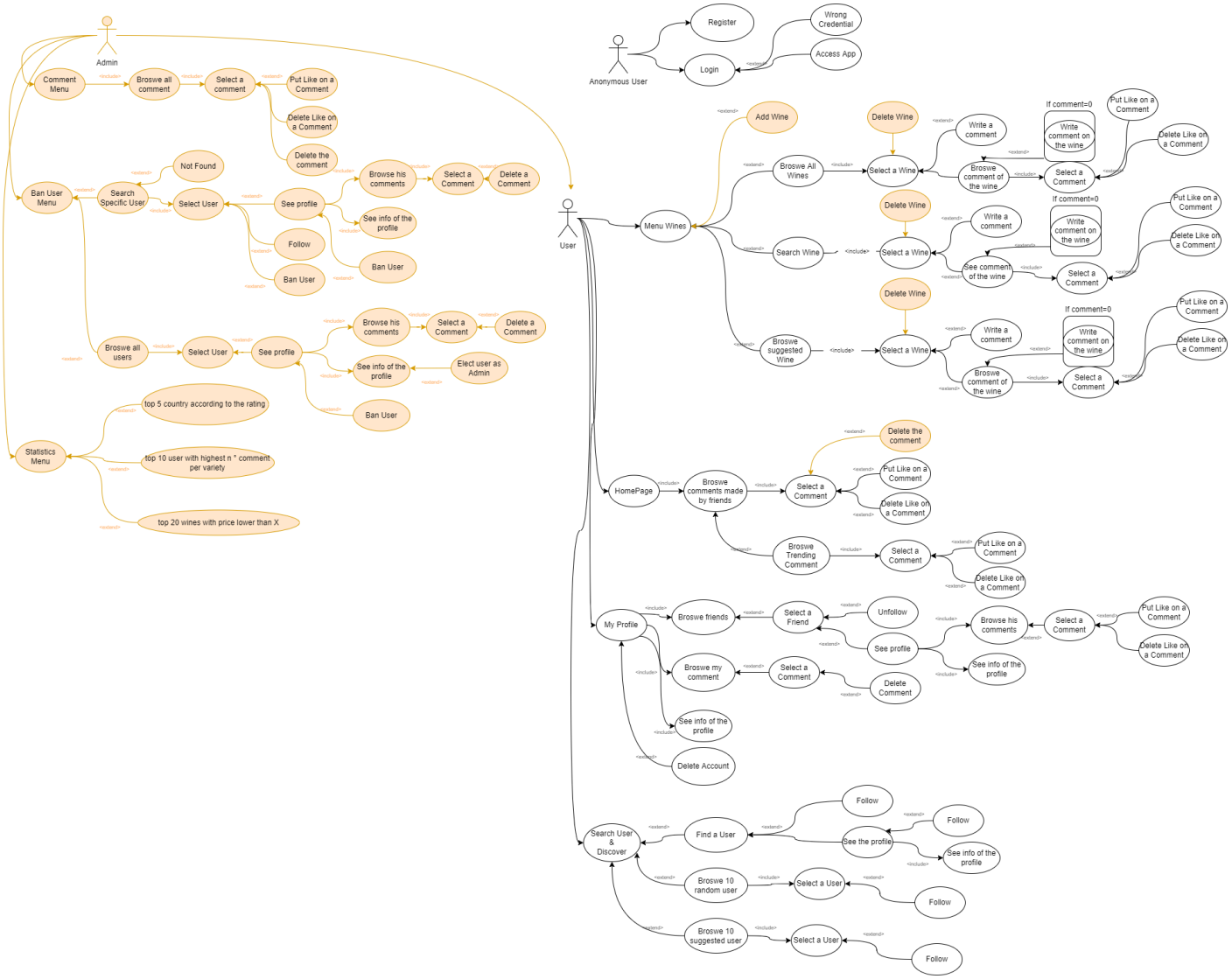
DATA TRANSFER TO THE TWO DATABASES

To transfer data from json to MongoDB was used *Populating_wine_document* class and then to populate Neo4J from MongoDB *Populating_function_social* was used.
Scraper combines addPostComplete method (Crud_graph) and createWine method (Mongo_crud) to insert data simultaneously in both databases.

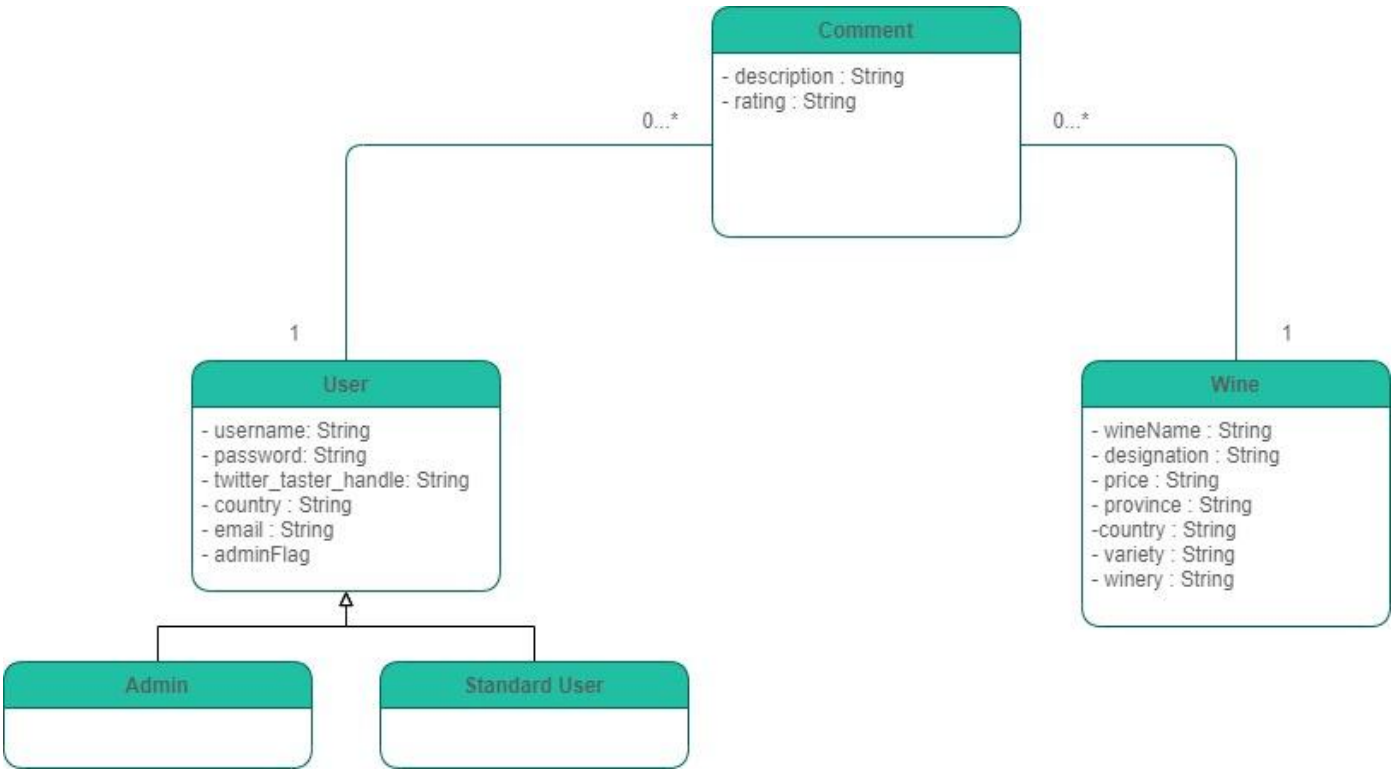
ACTORS AND USE CASE

Based on the software functional requirements, the application is meant to be used by three actors:

- **Anonymous Users**: who are only allowed to log in or register within the application via a username/password system.
- **Standard Users:** can perform all normal functions and use the social network. It has several sections and the general can manage their own profile, view the comments of friends, follow new users, search for friends, have wine suggestions or users, stop following friends, delete the account, delete / insert a comment for a wine, like / unlike comments, view friends' profiles
- **Admin**: in addition to the user functions, it can also ban a user, delete reviews, add wines, make statistics on the social network. (Orange function)



UML CLASS DIAGRAM



Classes definitions

Class	Description
User	A standard user (registered one) who can only perform basic operations
Admin	Most powerful user, he can also ban user, add wines,delete comment,make statistics
Comment	Comment posted by a user
Wine	Wine added by admin

Class Attribute for Admin and User

Attribute	Type	Description
Username	String	Username of the user/admin
Password	String	Password of the user/admin
Twitter Name	String	Twitter name of the user/admin
Country	String	Countryof the user/admin
Email	String	Email of the user/admin
Admin Flag	Int	Define the role: 0=User 1=Admin

Class Attribute for Comment

Attribute	Type	Description
Description	String	Body of the comment
Rating	Int	Rating of the comment

Class Attribute for Wine

Attribute	Type	Description
wineName	String	Wine name of the wine
Designation	String	Designation of the wine
Price	Int	Price of the wine
Province	String	Province of the wine
Country	String	Country of the wine
Winery	String	Winery of the wine
Variety	String	Variety of the wine

JAVADOC

For a complete overview on the code it is recommended to read Javadoc that is in the “Javadoc” folder inside the project.

DATA MODELLING

MONGO DB

INTRODUCTION

MongoDB has been designed to implement the one-to-many relationship between wine and reviews. Is used to display information regarding users, wines and reviews but login credentials are stored only on Neo4J.
Mainly is faster in reading w.r.t Neo4J for wines and reviews reading. Moreover it hosts advanced queries that could be used by the Admins to do some useful statistics on the Social Network to observe the trend.

DOCUMENTS ORGANIZATION AND STRUCTURE

In MongoDB are stored the following collections:

wines

Storage size:	Documents:	Avg. document size:	Indexes:	Total index size:
24.14 MB	79 K	554.00 B	2	17.71 MB

Wines is composed by a main document that identifies attributes of wines. And for each document there is an array of nested documents. One nested document is composed of reviews for that wine and the user attributes.


```
{
    "_id": {
        "$oid": "61e922ded644e16514dbf737"
    },
    "wineName": "Viña Alicia 2014 Tiara White (Luján de Cuyo)",
    "variety": "White Blend",
    "country": "Argentina",
    "province": "Mendoza Province",
    "price": 30,
    "winery": "Viña Alicia",
    "designation": "Tiara",
    "reviews": [{
        "rating": 84,
        "description": "Muddled, oily aromas are funky for certain. This blend of Riesling, Albariño and Savagnin
        "taster_twitter_handle": "@wineschach",
        "taster_name": "Michael Schachner",
        "user_country": "None",
        "email": "None"
    }]
}
```


Each document is a distinct wine. Cannot exist in wines collection two documents for the same wine, if more users reviews the same wines, their reviews will be stored inside reviews array as nested document with their attributes. If for some reasons there are two identical wines with different attributes will be inserted the last one with it attributes so there will be an attributes update.

ADVANCED QUERIES IMPLEMENTATIONS

In this table the crud operations and the basic functions have been omitted

Operation	Mongo Db query
Shows the TOP-3 countries that own higher average wines' ratings	db.wines.aggregate([{ $unwind:"$reviews"}, { $group: {_id:"$country",Average:{$avg:"$reviews.rating"}}}, {$limit:3}])
Shows TOP-10 username, with wines varieties and relative prices, that made highest number of reviews per variety	db.wines.aggregate([{ $unwind:"$reviews"}, { $group: {_id: { taster_name : "$reviews.taster_name", variety : "$variety"}, count:{$sum:1}}}, {$limit:10}])

Shows Top-50 wines, wines' prices, highest average rating, below a price treshold inserted by admin by keyboard	db.wines.aggregate([{ "$match": { "price": { "$lt": 14 } }}, { $unwind : "$reviews" }, { "$group": {"_id": { "wineName": "$wineName","price": "$price"},Average:{$avg:"$reviews.rating"}}}, {$limit:50}, {$sort:{"Average":-1}}])

IMPLEMENTATION DETAILS

- All advanced functions use mongodb's aggregation pipeline;
- All queries on mongo exploit mathematical operations, such as the calculation of the average, on graph db would not have been efficient;
- The advanced functions were used for admin analysis operations.

IMPACT OF INDEXES IN MONGODB QUERIES

This section will analyze the three MongoDB queries both with and without the use of indexes. This is done in order to actually understand what the impact of their use is and whether or not they manage to speed up the reading operations in this specific application.

The following table shows the indexes created on the wines collection:

	1	2	3	4	5	6
Index Field	_id	PriceCountry	wineCountry	price	country	wineName
Index Type	Default	Compound	Compound	Simple Index	Simple Index	SimpleIndex

The compound index WineName and Country, and the index on wineName, somehow, is mandatory because there is a CRUD operation that is useful for searching wine names even if the user inserts a partial name or only a few characters. To do this search an index on WineName is required.

Now each query will be analyzed with and without the aforementioned indexes

First query

The following table summarizes the execution performance of the previous query before and after creating indexes on the review collection.

Index	Document Returned	Index Keys Examined	Documents Examined	Execution Time(ms)
True	78680	0	786080	204
False	78680	0	786080	187

Second query

Index	Document Returned	Index Keys Examined	Documents Examined	Execution Time(ms)
True	78680	0	78680	225
False	78680	0	78680	256

Third query

Index	Document Returned	Index Keys Examined	Documents Examined	Execution Time(ms)
True	75598	75598	75598	290
False	75598	0	75598	313

MONGO OPERATIONS ANALYSIS

From the analysis of the use case were identified the following queries that involve Mongo DB database, along with their expected frequency and cost:

For write operations:

Operation	Expected Frequency	Computational Cost
Add/remove wine	Average	High (7 attributes write)
Add/delete one comment	High	High (6 attributes write)
Delete all comment for a given user	Low	Average (document(s) remove)

For reading operations:

Operation	Expected Frequency	Computational Cost
Find specific wine	High	Low (1 read)
Find all users	Low	Low (1 read)
Find all comments	Low	High (1 aggregation)
Find all wines	Low	Low (1 read)
Find wine by substring	High	Average (Filtering operation)
Shows the TOP-3 countries that own higher average wines' ratings	Average	High (aggregation)
Shows TOP-10 username, with wines varieties and relative prices, that made highest number of reviews per variety	Average	Very high (complex operation)
Shows TOP-30 wines and usernames that bought them, below a price threshold inserted by admin	Average	Very high (complex aggregation)

MONGODB REPLICAS

Replicas of the same data were used, in order to ensure availability and consistency. Nevertheless we have to point out that replicas alone don't ensure consistency, because it's possible to have one or more replicas that are not updated, this doesn't happen in our case because in order to commit a write operation all replicas must be updated. This is because the application is mainly read-heavy and so we need fast reads operation and, in order to achieve this, we can accept slow write operations. Moreover, this configuration ensures the Consistency requirement.

For writing operations we adopted:

- **W1**: Wait for acknowledgement from a single member.

For reading operations we adopted:

- **NEAREST**: Read from any member node from the set of nodes which respond the fastest.

SHARDING

Sharding: It has been decided to split the dataset into three shards and use three replica sets for each shard which ensures high availability for the system and to make the system persistent against any failures.

To divide our data in shards, it has been decided to choose a partitioning algorithm based on hash strategy.

The Object_id is the field that we have chosen as our hashed shard key, because the shard key should have a good cardinality and with fields that change monotonically like ObjectId , as per best practices read on the mongoDb website.

NEO4J

INTRODUCTION

Neo4j is used for login management and social aspects of the application, including the friend and wine recommendation system.

Among the classic social aspects we have the management of:

1. Friendships between users
2. Like between user and post
3. Relationship between the creator and the comment
4. Relationship between the Wine and the comment

1. Thanks to this relationship it is possible to perform several functions:

- Going to the homepage you will see all the posts of friends, so as not to miss the future comments they will make on the wines.
- In the section dedicated to your profile you can view all the users followed and, if necessary, stop following them
- It is possible to view a friend's profile
- By seeing your own or a friend's profile, the number of followers will be displayed
- Used for the friend recommendation system

2. Thanks to this relationship it is possible to perform several functions:

- Every time a comment is displayed, you will see the number of likes that received that comment
- Trending posts are the most liked comments, so they will be recommended to more people on the homepage
- Every time you see a comment you can see if the like has been left or not based on the "X" or "V" symbols
- Used for the wine recommendation system
- Used for the friend recommendation system

3. Thanks to this relationship it is possible to perform several functions:
- When I go to see a user's profile there will be all the posts made by him
- When I go to see my profile, there will be all the posts I have made, and if I want I can also delete it
- When I see a comment, as in all social networks, I have to be able to see the author of the comment
- Used for the wine recommendation system

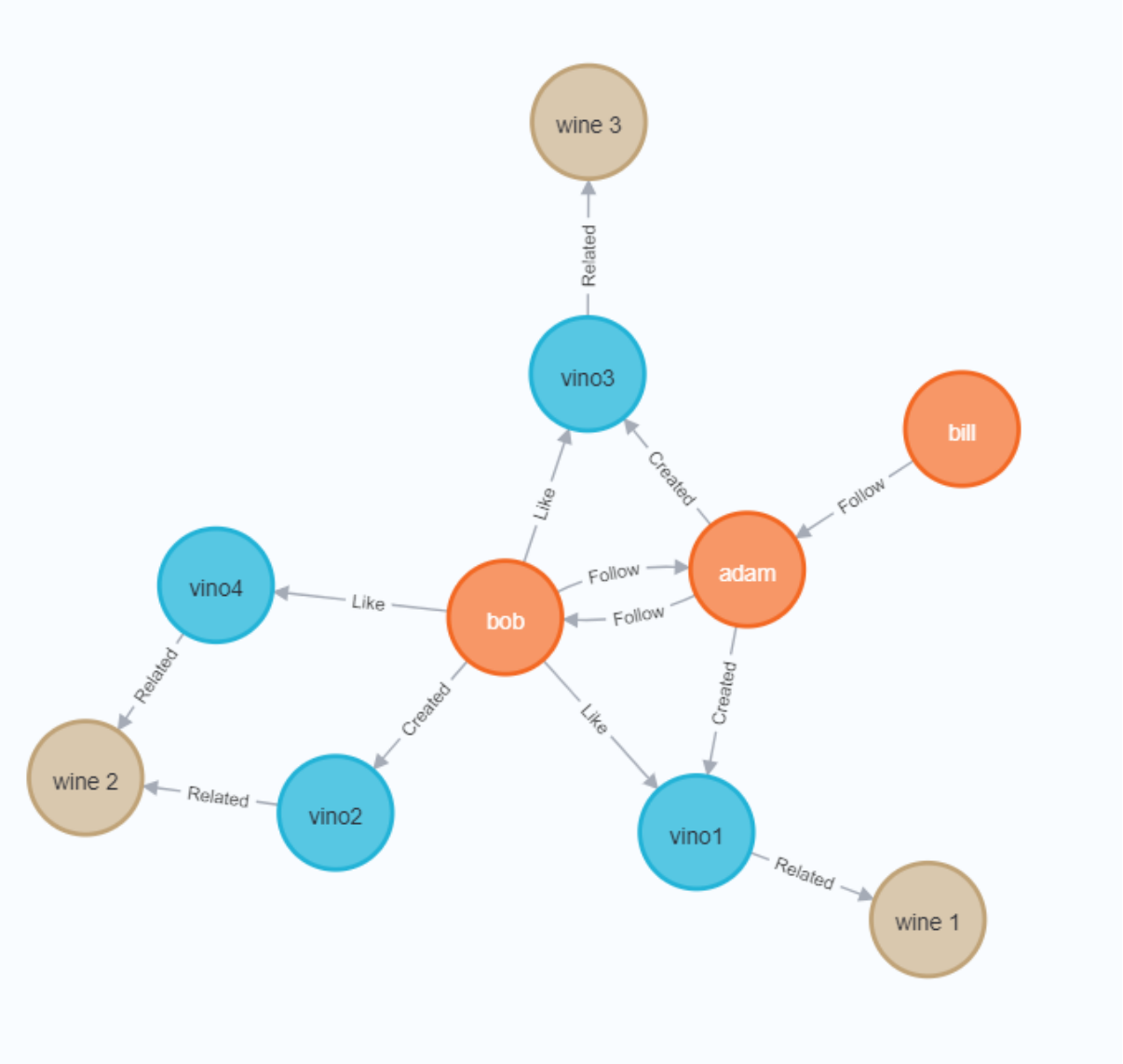
4. Thanks to this relationship it is possible to perform several functions:
- Used for the wine recommendation system

We also have two recommendation systems:

1. Wine recommendation
2. Friendship Recommendation

1. Details:
- Part of the recommended wines derive from the likes of the comments left on certain wines
- Part of the recommended wines derive from the comment left on a wine

2. Details:
- Part of the friendships suggested comes from the likes left in the comments of other users
- Part of suggested friendships come from mutual friends



Example:
Logged user = BOB

Suggested wine = wine1 , wine2 , wine3
wine2 = because he created a comment on the wine
wine1,wine3,wine2= because he liked a comment related to the wine

Suggested User = Bill , John
Bill = because is a friend of a friend
John = because he liked a post created by John

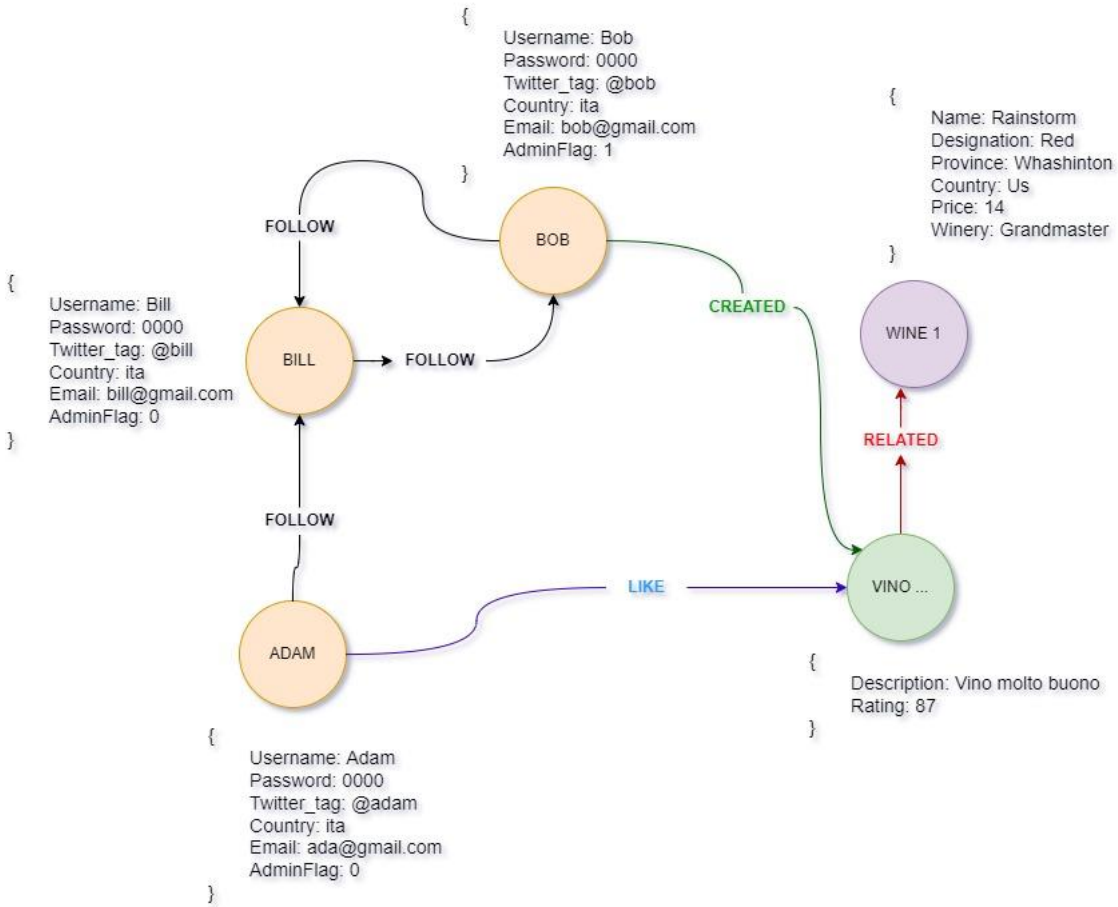
GRAPH DESIGN

Nodes:

- **User**: Nodes, representing the users registered within the application, having as attribute : (username,password,tag twitter, country , email ,adminFlag)
- **Comment**: They represent the comments that users write for the wines. Having as attributes: (description , rating)
- **Wine**: They represent the wines that are present in the social network. Have these attributes: (name,province,winery,price,province,designation)

Relationships:

- User → FOLLOW → User: each user can follow other users, but cannot follow themselves;
- User → LIKE → Post: each user can like other users' posts;
- User → CREATED → Post: this relationship is used to identify who created the post;
- Post → RELATED → Wine: this relationship is used to identify the wine related to the comment



QUERIES IMPLEMENTATIONS

Crud operations and more basic operations have been omitted in this table

Operation	Cypher implementation
Recommend 10 friends to the user based on his friendships with other user and liked comment of other user	"CALL{ \n"+ "MATCH(n:User{username:$username})-[:Like]->(:Post)<-[:Created]-(u:User)\n" + "WHERE NOT EXISTS ((n)-[:Follow]->(u))\n" + "WITH u, rand() AS number\n" + "RETURN u ORDER BY number \n" + "UNION \n"+ "MATCH(n:User{username:$username})-[:Follow]->(:User)<-[:Follow]-(u:User)\n" + "WHERE NOT EXISTS ((n)-[:Follow]->(u))\n" + "WITH u, rand() AS number\n" + "RETURN u ORDER BY number \n" + "}\n"+ "RETURN u.username AS username , u.country as country , u.twitter_taster_handle as twitter_taster_handle LIMIT 10 "
Retrieve the list of 5 trending posts based on number of like	"MATCH (p:Post)-[r:Like]-(u:User)\n" + "RETURN p.titlePost AS titlePost, COUNT(r) AS numLike \n" + "ORDER BY numLike DESC\n" + "LIMIT 5"
It suggests 6 wines based on the comments made and the likes left on the social network	"CALL{\n"+ "MATCH(n:User{username:$username})-[:Created]->(:Post)-[:Related]->(u:Wine)\n" + "WITH u, rand() AS number\n" + "RETURN u ORDER BY number\n" + "UNION\n" + "MATCH(n:User{username:$username})-[:Like]->(:Post)-[:Related]->(u:Wine)\n" + "WITH u, rand() AS number\n" + "RETURN u ORDER BY number\n" + "}\n" + "RETURN u.wineName AS wineName , u.price as price , u.designation as designation , u.winery as winery , u.variety as variety , u.province as province LIMIT 6 \n"
Show followed (used in My Profile)	"MATCH (u1:User{username: $username}) , (u2:User)\n" + "WHERE EXISTS ((u1)-[:Follow]->(u2))\n" +

	"RETURN u2.username AS username , u2.country AS country , u2.twitter_taster_handle AS twitter_taster_handle , u2.email AS email",
Show all comment made by Friends (used in the Homepage)	"MATCH(u:User{username:$myUsername}),(u1:User{username: $usernameFriend}) , (p:Post),(w:Wine) \n" + "WHERE EXISTS ((u)-[:Follow]-(u1))\n" + "AND EXISTS ((u1)-[:Created]->(p))\n" + "AND EXISTS ((p)-[:Related]-(w))\n" + "RETURN p.description AS description , p.rating AS rating , w.wineName AS wineName \n"
Show the comment that i made (used in MyProfile section)	"MATCH(u:User{username:$myUsername}),(p:Post),(w:Wine) \n" + "WHERE EXISTS ((u)-[:Created]->(p))\n" + "AND EXISTS ((p)-[:Related]->(w))\n" + "RETURN p.description AS description , p.rating AS rating\n",
Count like of a post (used in the comment visualization)	"MATCH (u:User)-[r:Like]->(p:Post{description: $description})\n" + "RETURN COUNT(r) AS numLike",
Count the number of follow of a User (used for visualization of friend)	"MATCH (u:User{username: $username})<-[r:Follow]-(u2:User)\n" + "RETURN COUNT(r) AS nfollowers"
Used to check if the user has liked the post (Used in viewing comments)	"MATCH (u:User{username: $username}),(p:Post),(w:Wine{wineName:$wineName})\n" + "WHERE EXISTS ((p)-[:Related]->(w))\n"+ "AND EXISTS ((u)-[:Created]->(p))\n"+ "RETURN u.username as username"

Extracts 10 users and creates the follow relationship with the selected user //NOT IN APP	"MATCH (p:User) RETURN p.taster_name as taster_name LIMIT 10"
	While:
	createRelationFollow(selected_taster_name, taster_name);
	createRelationFollow(taster_name, selected_taster_name);

Randomly extracts 10	"MATCH (p:Post) RETURN p.titlePost as titlePost LIMIT 10"

posts and creates the Like relationship with the selected user //NOT IN APP	While: createRelationLike(titlePost, selected_taster_name);

IMPLEMENTATIONS DETAILS

- In the scraper the function has been implemented that while analyzing the reviews automatically creates all the elements in the database and links them exactly;
- Every time a relation is created, it is checked if it is already present in order to avoid unnecessary edges;

DATA AMONG DATABASES

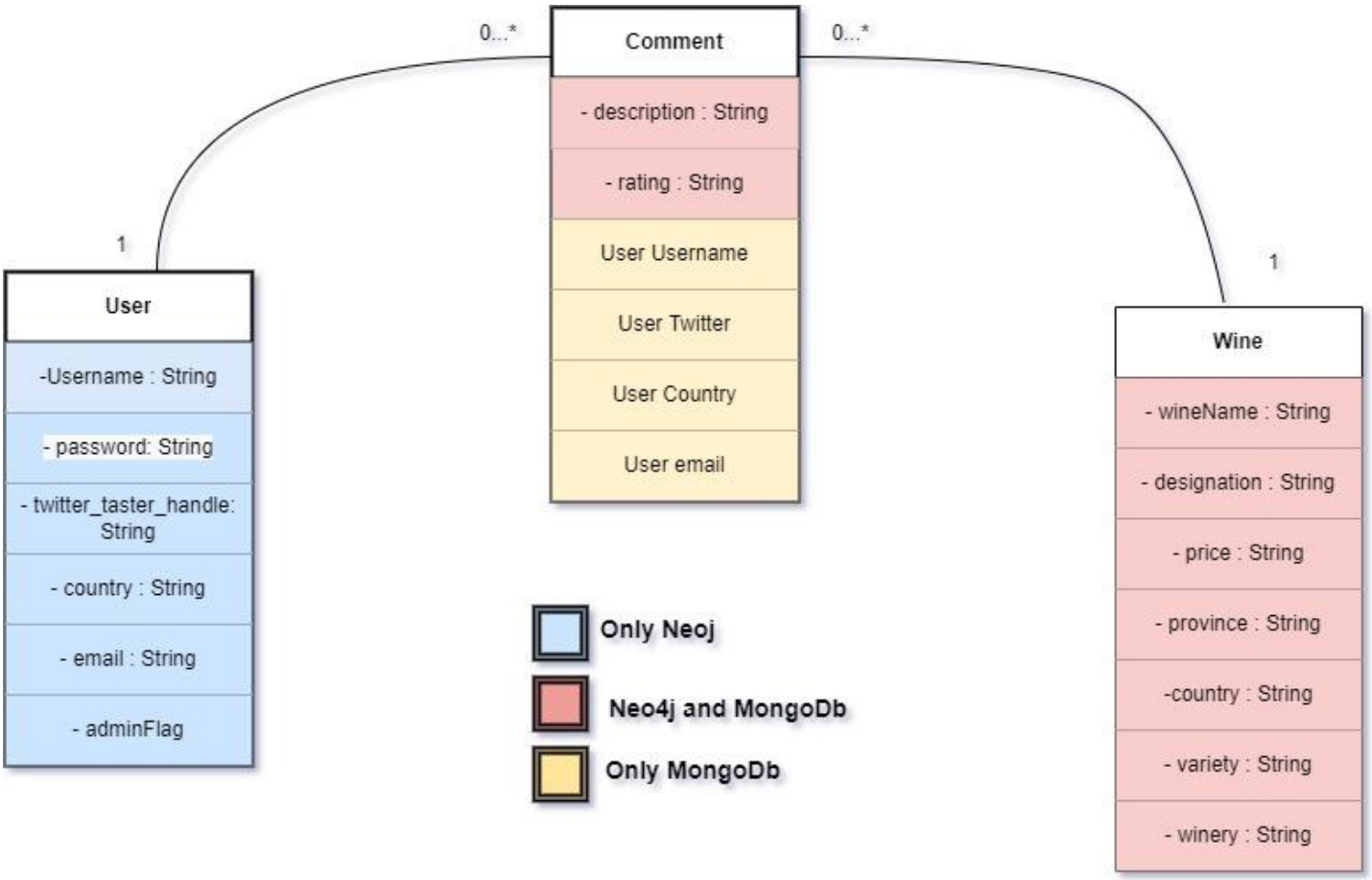
We used the two types of databases to manage different functions of our app.

Comments are saved on both databases, but with slightly different data to suit the needs of the functions

On neo4j we manage the social and recommendation part, in fact each comment can receive a like, and based on these likes we can calculate the trending posts of the moment. Additionally, our referral system uses comment relationships to suggest wines.

On MongoDb we manage the visualization of comments for wine and we make statistics regarding the comments

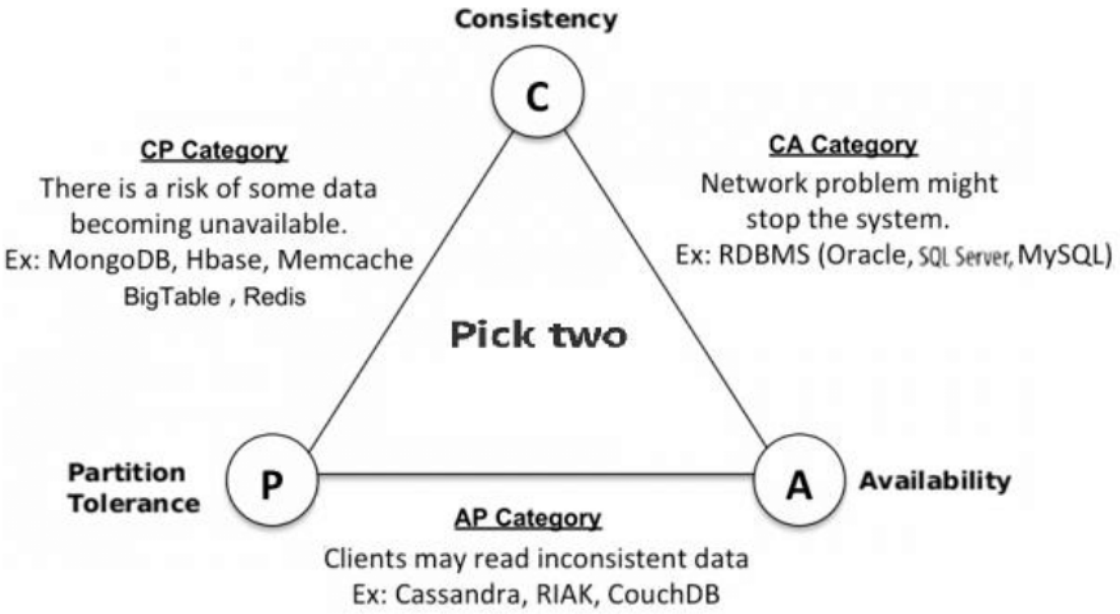
Users are saved only on neo4j to manage accesses and roles (Admin/user).



DISTRIBUTED DATABASE CAP THEOREM

In our application we have decided to ensure Consistency and availability:

- Consistency: each user has the same updated version of the data
- Availability: ensured by having replicas on three different virtual machines on Unipi servers



Mongo DB Replicas provide redundancy and increase data availability. With multiple copies of data on different database servers, replication provides a level of fault tolerance against the loss of a single database server.

In some cases, replication can provide increased read capacity as clients can send read operations to different servers. Maintaining copies of data means increasing data locality and availability for distributed applications.

Thanks to MongoDB Replicas, service is always available because if the primary server goes down the second one will take its place.

For this reason, the system is always available to accept read and write operations from users.

From consistency point of view data all always updated thanks to replicas because all nodes of clusters are updated so users will always see the last version of Social Wine's datas.

ARCHITECTURAL DESIGN

The main language used in developing the application is Java

Client Side:

- The front-end module, which consists of a graphical interface based on Command line interface (CLI), which allows the users to use the application in the simplest way possible.
- A middleware module, needed to interface the client with the server. More precisely, a driver has been implemented to interface with MongoDB and one to interface with Neo4j.

Server Side:

- The server side consists of a cluster of three virtual machines made available to us by the Unipi, which were used to host a sharded MongoDB cluster.

OTHER IMPLEMENTATION DETAILS

SCRAPER

Scraper highlights:

- Scraper is linked to this website ***https://winemag.com/ratings/#***. It takes all the useful information like: winename, variety, country, province, price, winery, designation, rating, description, taster_twiter_handle, taster_name. Like for json translation were inserted other two user attributes like user_country and email. These last attributes don't exist in the website so when they are added, they will be "None";
- From this website the scraper takes the reviews that will be inserted like nested documents for the main document that identify the wine inside the MongoDB WInes collection. If the scraper finds reviews that it had previously scraped will not insert them because of checks that are implemented inside and will not insert reviews that contain null values;
- Scraper, to insert data in MongoDB and Neo4J recall CRUD operations
- Scraper is inserted inside a Thread that works in a pre-defined timing range in such a way that MongoDB collections and Neo4J graph are always updated;
- If, for any reason, the scraper isn't able to works due to some network or server failure (Error 500) the execution continues and the application will work with the reviews already inside MongoDB Review collection;
- For the scraper jsoup is used.

MENU

The application is not provided by GUI, even if it would have been better from aesthetic aspects, because the main focus was on the best design of databases. For this reason the menu is created by using the CLI (Command Line Interface).
Having more permissions, the admin has a slightly different menu that adds some admin functions.