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 #: - email: d.armillotta@studenti.unipi.it
- **Leonardo Bellizzi -** badge #: 643019 email: l.bellizzi@studenti.unipi.it

Summary

INTRODUCTION AND MAIN REQUIREMENTS **3**

FUNCTIONAL REQUIREMENTS 3

NON FUNCTIONAL REQUIREMENTS 4

WORKING ASSUMPTIONS 4

SPECIFICATION **5**

UML CLASS DIAGRAM 5

CLASSES DEFINITIONS 6

ACTOR 10

USE CASE DIAGRAM 10

ARCHITECTURAL DESIGN **11**

APPLICATION CORE 11

SERVER POPULATION 12

MONGO DB DESIGN AND IMPLEMENTATION **13**

DOCUMENTS ORGANIZATION AND STRUCTURE 13

QUERIES IMPLEMENTATIONS 15

IMPLEMENTATION DETAILS 15

IMPACT OF MONGO DB INDEXES 16

SHARD CONFIGURATION 18

NEO4J DESIGN AND IMPLEMENTATIONS **19**

GRAPH DESIGN 19

QUERIES IMPLEMENTATIONS 20

IMPLEMENTATION DETAILS 21

LEVEL DB DESIGN AND IMPLEMENTATIONS **21**

IMPLEMENTATION DETAILS 21

OTHER IMPLEMENTATIONS DETAILS **21**

SCRAPER 21

MENU 22

LOGIN 22

INTRODUCTION AND MAIN REQUIREMENTS

The application that has been developed is called “Social Wine”. The main objective of this application is to offer to the users a Social Network based on wines’ reviews.
Each review is composed by points, title, description, taster_name, taster_twitter_handle, price, designation, variety, region_1, region_2, province, country and winery.
Each user that could log inside the application could do different things inside the Social like, interact with other users of the network.
Moreover there is an admin figure that could do several statistics on the social network, will see in detail after, and add users, ban users, add winery, delete winery and populate the network using MongoDB like a Backup Database.
Another important feature is the dynamic scraping mechanism. The reviews that are used have been retrieved in a “static way” from Kaggle but a scraper has been added to add to the existing dataset every time new reviews.

FUNCTIONAL REQUIREMENTS

In the following, there is a bullet list that explain which are the requirements of the application:

- The users of the application must be divided into *two* categories: the *Standard User* and the *Admin*. These two figures are separated by username and password, in the application has been emulated an username/password model.

For the **Admin** the **username** is **admin** while **password** is **root**, for the **Standard User** the username corresponds to the user **name** that is **present** inside **MongoDB collection** while **password** is **abcd**.

- The following operations and statistics must be offered to the **Admin**:
- Populate social based on review collection of MongoDB;
- Top-10 countries that have most wineries in descending order;
- Display top-20 wines’ varieties according to their mean price;
- Top-5 users with the highest average of the review scores;
- Create 10 follow relation between selected user and 10 random people;
- Create 10 like relation between selected user and 10 random post;
- Add an user;
- Ban an user;
- Add a winery;
- Drop a winery;
- The following operations must be offered to the Standard User:
- Follow a friend;
- Unfollow a friend;
- Put like on a post;
- Remove like on a post;
- Add an own post on the social;
- See suggested friends’ list for him;
- Discover the top 5 trending post in that moment;
- See all user that he follows;
- See all the reviews given by a specific user of the network.

NON FUNCTIONAL REQUIREMENTS

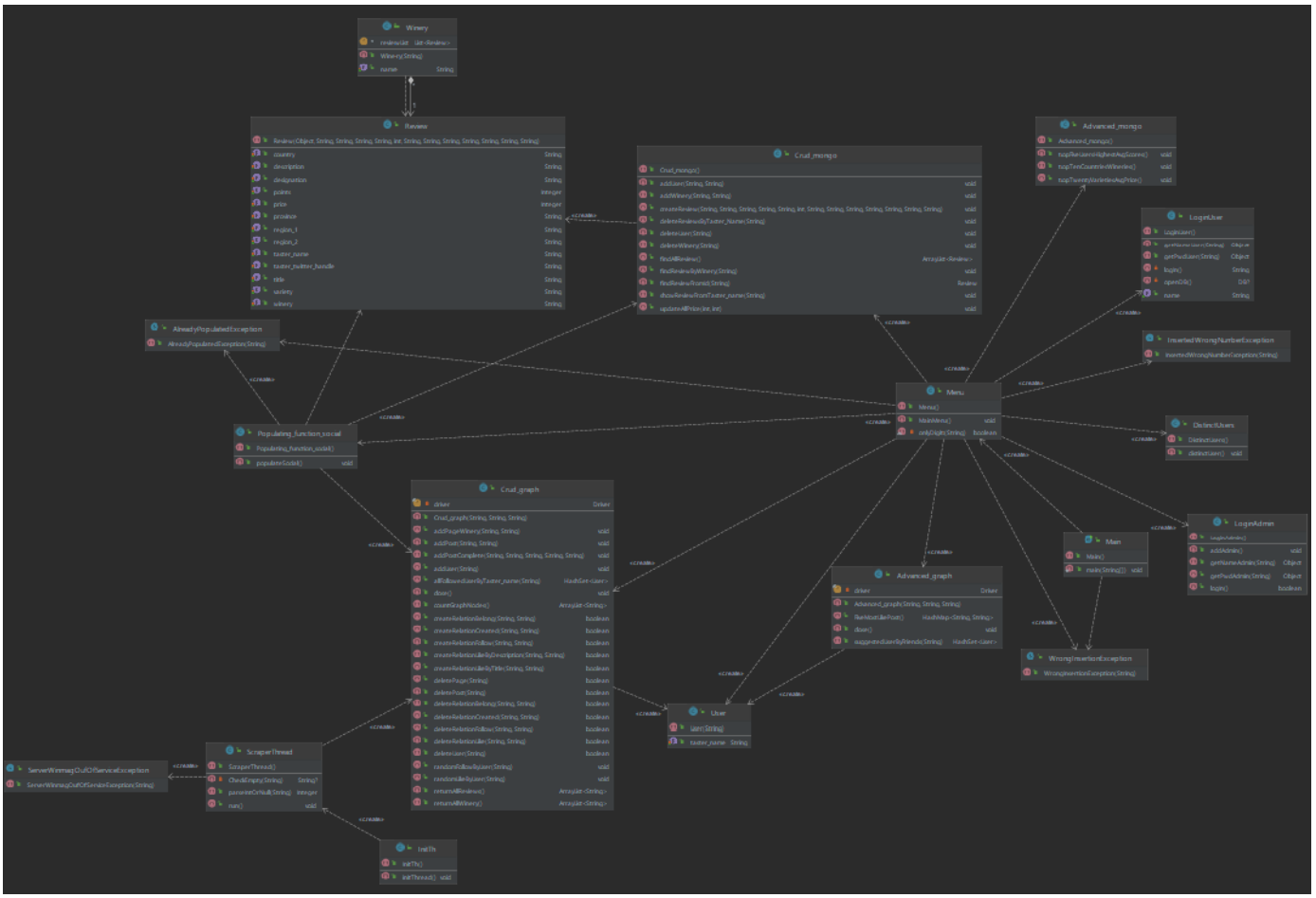
- The application must be simple, intuitive and offer an experience enjoyable for the user.
- The user's username must be unique.
- The code must be readable, documented and easy to maintain.
- The data must be consistent in order to always display the correct information.
- The system requires users to login to access the application.
- Each user can always read and write
- The scraper must work daily at night to read the new data from the win reviews website and insert it in MongoDb database and make it available for the next day.

WORKING ASSUMPTIONS

- The users are already registered inside the social so they have only to login inside the menu and pass checks. Into the system new users can't log, the users are only users that are stored inside the "user_credentials" collection on MongoDB.
- All users have the same password, for simplicity like the Admin figure that has always admin-root like username-password.
- If a user is dropped from the SocialWine, his reviews will stay inside the social and other users could put like on it without following the dropped user.

SPECIFICATION

UML CLASS DIAGRAM



CLASSES DEFINITIONS

Inside the project folder there is a subfolder called “JavaDoc” in which there are all the details about the package, classes and methods definitions. Are all in html format.

Class	Description
Review	The class which contains all review attributes according to the dataset.
Winery	The class contains the attributes for the winery.
Advanced_graph	This class contains Neo4J advanced queries.
Advanced_mongo	This class contains MongoDB advanced queries made with aggregation pipeline.
Crud_graph	Contains all the crud operation that could be done on Neo4J.
Crud_mongo	Contains all the crud operation that could be done on MongoDB.
Populating_function_social	The class contains a method that take all the reviews that are stored in Review collection inside MongoDB and add them inside Neo4J automatically.
Wrong Insertion Exception	Contains exception that is raised when the user doesn’t insert nothing or insert a character when he is asked to press button 1,2,0 to declare himself like an admin or to terminate program.
AlreadyPopulatedException	Contains AlreadyPopulatedExcpetion that is raised when the admin want to load in the graph the same first 96 k reviews coming from review MongoDB collection.
ServerWinmagOufOfServiceExcepti on	Contains the ServerWinmagOutOfServiceException that is raised if for some reason the Winmag website is down. In that case the execution will resume but taking in consideration only the 96 k reviews of Review MongoDB collection.
InsertedWrongNumberException	Contains InsertedWrongNumberException that is raised when the user want to follow a user, unfollow a friend and put like on a post choosing the post or the user/friend with a number that don't correspond to the number link to that user/friend or post.
DistinctUsers	Class that works to store all the user credentials in a MongoDB collection.
LoginAdmin	This class provides a login system for the admin.
LoginUser	This class provides a login system for the users.
Menu	The class contains the core of user/admin choice. From here the user could choice what they want to do.
InitTh	Start thread for scraping and keep it on for a period of time.

ScraperThread	Contains scraper that allow to retrieve new reviews from winemag website.

Classes Methods:

Advanced_graph	
Method	Description
Advanced_graph	The constructor that allows to start the connection with Neo4J.
Close	Close the connection with Neo4J's DBMS.
FiveMostLikePost	Find the top five post on the social according to their like, in a descending order.
suggestedUserByFriends	Suggest five users to given user, that are friend of friend that are not yet followed.

Advanced_mongo	
Method	Description
topFiveUsersHighestAvgScores	Top five users with the highest aerage of them review scores.
topTenCountriesWineries	Top ten countries that own most wineries.
topTwentyVarietiesAvgPrice	Display to twenty wines' varietis according to thei mean price.

Crud_graph	
Method	Description
Crud_graph	Constructor that allows to start the connection with Neo4J.
addPageWinery	Adding winery to the graph.
addPost	Adding post to the graph.
addPostComplete	Add a post on graph adding the relations "created by" (user-post) and "belong" (post-winery).
addUser	Create a new user.
allFollowedUserByTaster_name	Show all users that are followed by a given user of the social.
close	Close the connection with Neo4J's DBMS
countGraphNodes	Method that count the nodes in the Social
createRelationBelong	Create the relation belong between a review and a winery, to indicate to which winery that review refers to.
createRelationCreated	Create the relation created between a review and a user.
createRelationFollow	Create the relation "follow" between two users of the social, if that relation doesn't exist.
createRelationLike	Create the relation like between a user and a post.
deletePage	Delete a winery from the social.
deletePost	Delete a review from the social.

deleteRelationBelong	Delete the relation belong between a review and a winery, to indicate to which winery that review refers to.
deleteRelationCreated	Delete the relation created between a review and a user.
deleteRelationFollow	Drop the relation "follow" between two users.
deleteRelationLike	Delete the relation like between a review and a user.
deleteUser	Delete a user from the social.
randomFollowByUser	Extract ten users randomly from the social network and add the follow relation between the users, given a tester name.
randomLikeByUser	Extract ten post from the social network to put like given a tester name.
countGraphNodes	Counts all nodes inside Social Network
returnAllReviews	Return list of reviews that are in Social Netwok but only the description (body)
returnAllWinery	Return list of wineries that are inside Social Network

Crud_mongo	
Method	Description
addUser	Create a new user.
addWinery	Create a new winery.
createReview	Create a new review inside review collection.
deleteReviewsByTaster_Name	Delete all the reviews of a specific user.
deleteUser	Delete a user.
deleteWinery	Delete a winery.
findAllReview	Find all the review stored in the review collection in MongoDB.
findReviewByWinery	Retrieve all the review that refers to a specific winery.
findReviewFromId	Find review from the _id and create a new review.
showReviewFromTaster_name	Show all the review made by a specific user.
updateAllPrice	Update the price under a certain threshold.

Populating_function_social	
Method	Description
populateSocial	The method will create automatically nodes on Neo4J graph.

DistinctUsers	
Method	Description

distinctUser	The method will take from the review collection in MongoDB the distinct users and will give them a passowrd and will store the data (username and password) in a MongoDB collection called user_credentials. More, the method will check if that username and password are already inside the collection in order to don't add them again.

LoginAdmin	
Method	Description
addAdmin	Add the admin figure to the user_credentials collection with name and password.
getNameAdmin	Executes the comparison between the name that the admin has inserted and the username stored in mongoDB collection that store all the users' name and passwords.
getPwdAdmin	Executes the comparison between the password that the admin has inserted and the username stored in mongoDB collection that store all the users' name and passwords.

LoginUser	
Method	Description
getName	It's to take only the user's name
getNameUser	Executes the comparison between the name that the user has inserted and the username stored in mongoDB collection that store all the users' name and passwords.
getPwdUser	Executes the comparison between the name that the user has inserted and the username stored in mongoDB collection that store all the users' name and passwords.
logIn	This method it's called by the menu and user could insert his credentials.
openDB	Open connection with LevelDB

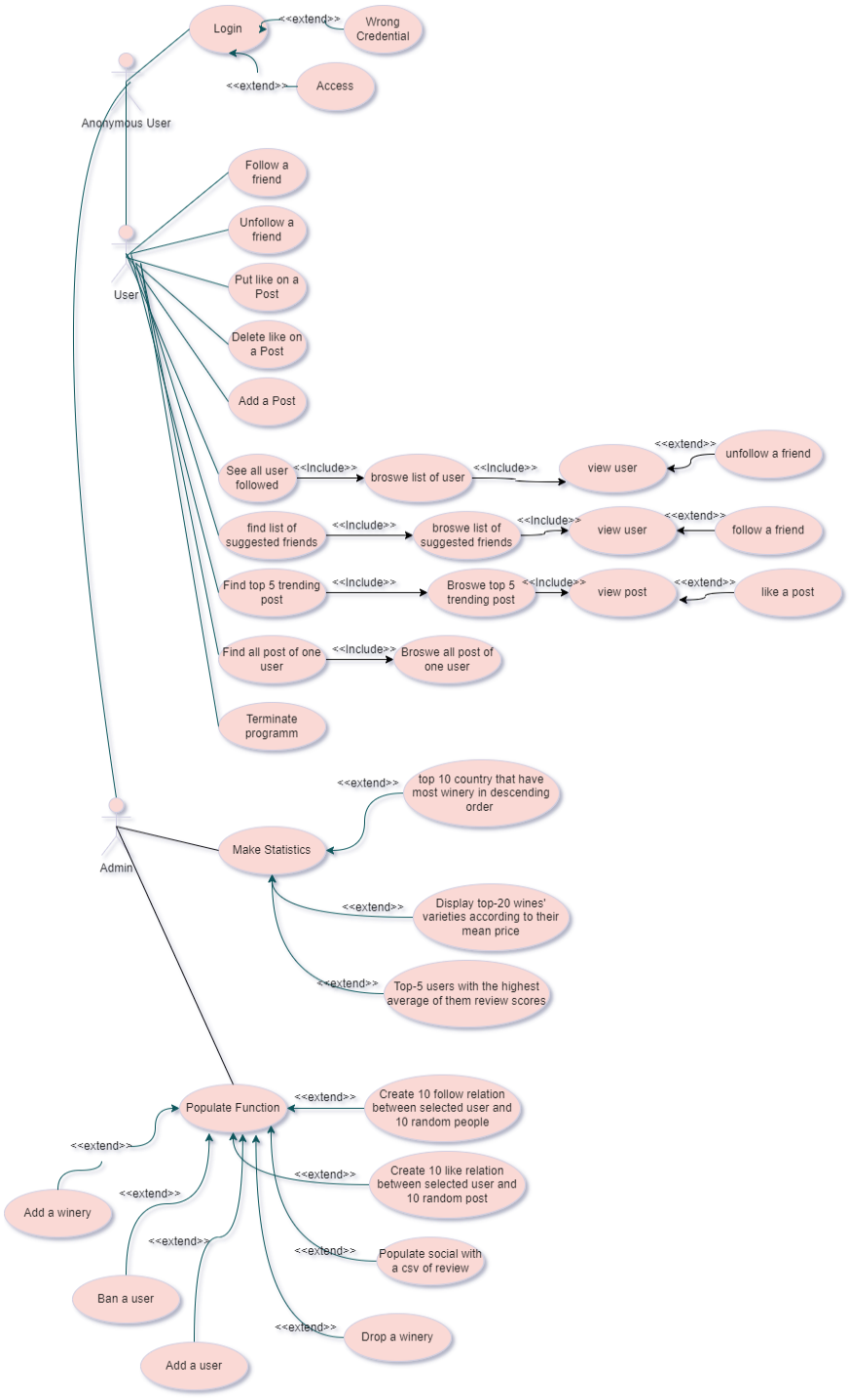
Menu	
Method	Description
MainMenu	Here there are the two different menus that will appear with respect to if the person that want to login as a normal user or an admin. Here, also, are initialized all the other classes like LoginAdmin, LoginUser, DistinctUser, Crud_mongo, Advanced_mongo, Crud_graph, Advanced_graph
onlyDigits	Check if the given a string, its contains only digits and not string.

ACTOR

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

- **Anonymous User :** user not yet logged into the app, so they do not have access to the app
- **Standard User:** has successfully logged in, can carry out all operations on the social network, such as following / unfollowing a person, putting / unlike a post, writing a post, finding recommended friends, finding trending posts on the social network, viewing the friends list
- **Administrator:** has the task of monitoring the social network, having the ability to ban a user, or a winery. But it can also populate the social network by creating synthetic interactions useful especially at the beginning of the social network. Obviously it can also do statistics and analysis.

USE CASE DIAGRAM



ARCHITECTURAL DESIGN

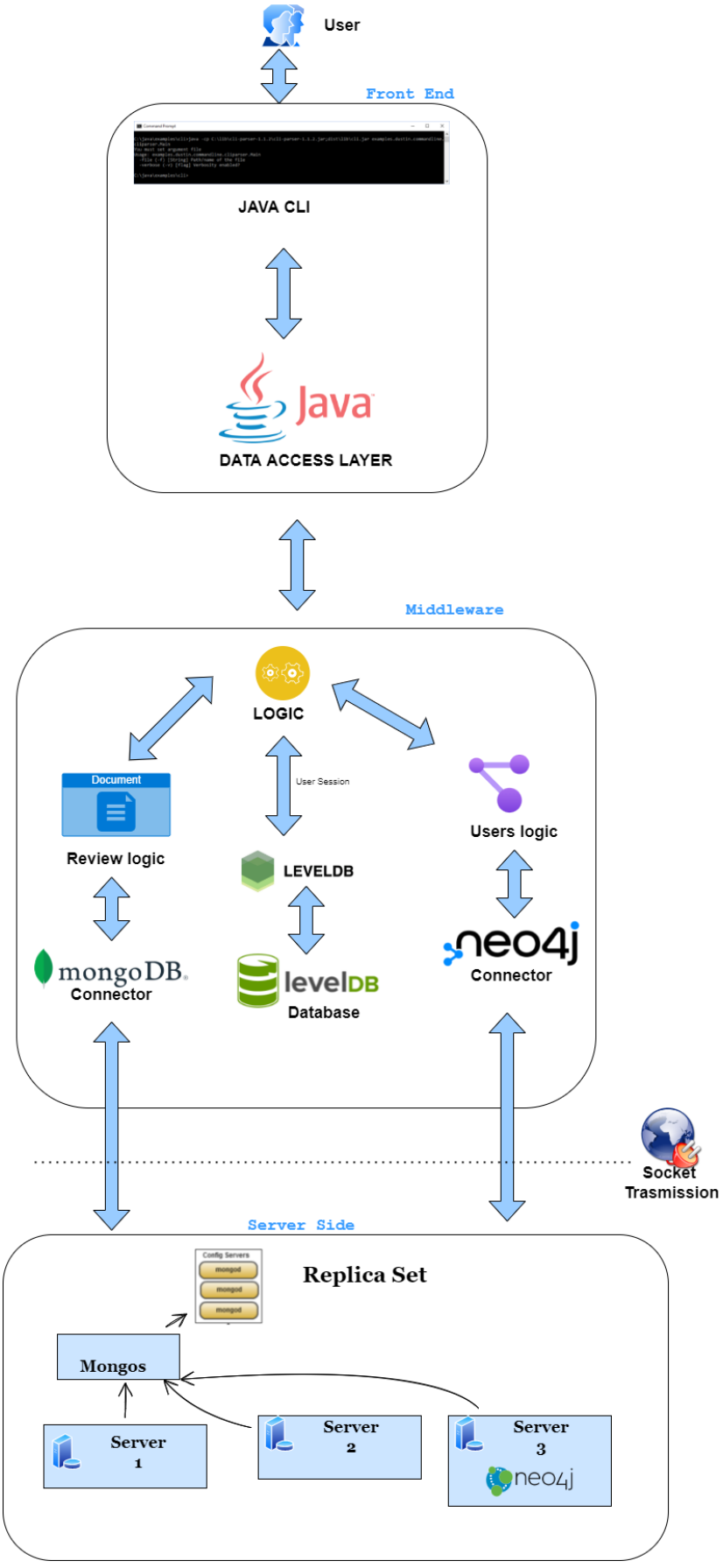
For the application no graphic interface is foreseen so all the application must be run from the IDE. Has been using Java 13 as core programming language.

APPLICATION CORE

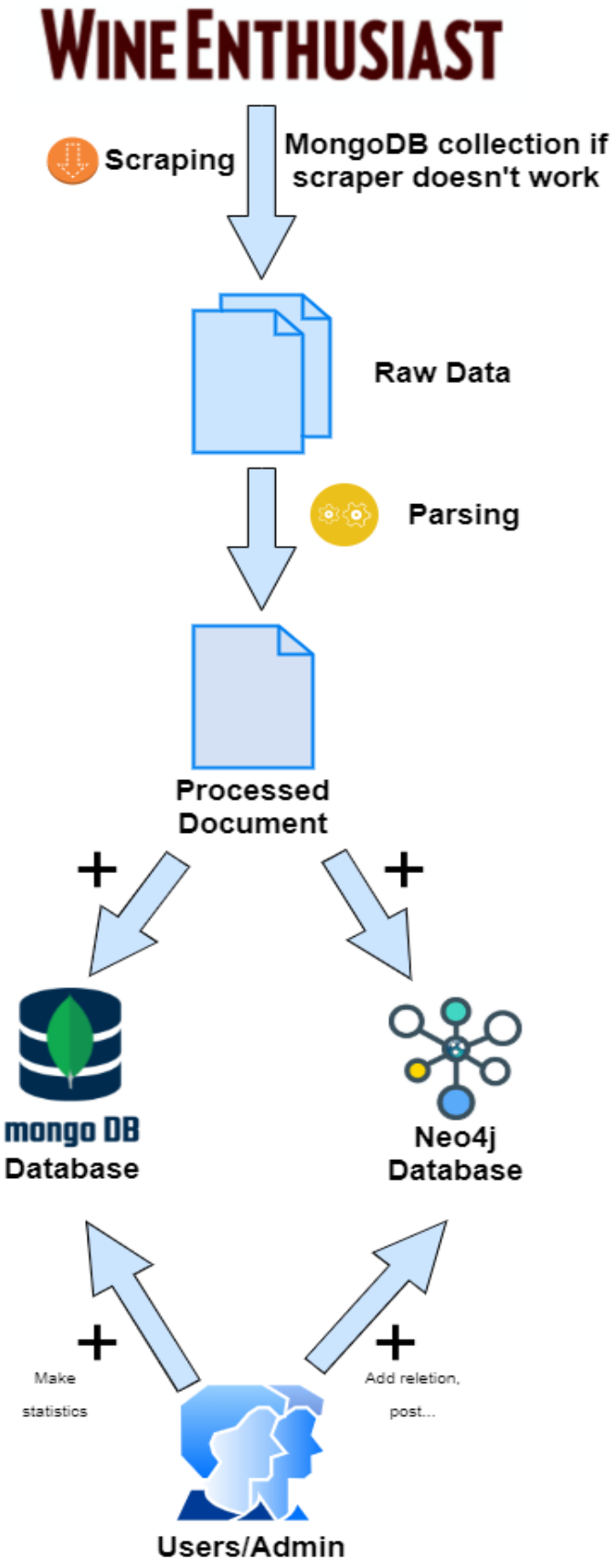
The core of the application is the usage of three different databases that allow managing several use cases from the emulated logging scenario with a temporary save, to admin's statistics to the social part.

- **MongoDB** (Admin): DB used for statistical purpose and basement for Neo4J;
- **Neo4J** (User/Admin): DB for the Social Network based on collection on MongoDB;
- **LevelDB** (User/Admin): DB used only for credentials check and to maintain the entry credentials of the users for the session in which it is logged.

CLIENT - SERVER



SERVER POPULATION



MONGO DB DESIGN AND IMPLEMENTATION

DOCUMENTS ORGANIZATION AND STRUCTURE

In MongoDB are stored the following collections:

Collection Name	Documents	Avg. Document Size	Total Document Size	Num. Indexes	Total Index Size
review	91,501	612.2 B	56.0 MB	1	1.1 MB
user	40	62.7 KB	2.5 MB	1	36.9 KB
user_credentials	1,127	64.8 B	73.0 KB	1	61.4 KB
winery	13,832	167.3 B	2.3 MB	1	364.5 KB

In ***review*** are stored all the reviews taken from Kaggle and here the scraper add all the reviews that each time detect.


```
_id: ObjectId("61ad0df127de528fef1a21f1")
points: "87"
title: "Quinta dos Avidagos 2011 Avidagos Red (Douro)"
description: "This is ripe and fruity, a wine that is smooth while still structured...."
taster_name: "Roger Voss"
taster_twitter_handle: "@vossroger"
price: 15
designation: "Avidagos"
variety: "Portuguese Red"
region_1: null
region_2: null
province: "Douro"
country: "Portugal"
winery: "Quinta dos Avidagos"
```


In ***user*** are stored all the distinct users taken from the review collection and for each user there is a nested array that contains all the score that the user gave to in his reviews. (Was useful to make statistics).


```
name: "Roger Voss"
score: Array
  0: "87"
  1: "87"
  2: "87"
  3: "87"
  4: "86"
  5: "86"
  6: "85"
  7: "86"
  8: "86"
  9: "86"
  10: "86"
  11: "86"
  12: "88"
  13: "88"
```

In ***user_credentials*** are stored all the users taken from the review collection and for each user is assigned a password. The users of this collection are the users who can log into the application. Moreover, inside this collection is stored the Admin that is a particular user.

```
Name: "Alexander Peartree"
Password: "abcd"

_id: ObjectId("61c1f5f8092b382c49c0235f")
Name: "Anna Lee C. Iijima"
Password: "abcd"

_id: ObjectId("61c1f5f8092b382c49c02360")
Name: "Anne Krebiehl MW"
Password: "abcd"

_id: ObjectId("61c1f5f8092b382c49c02361")
Name: "Carrie Dykes"
Password: "abcd"
```

In the winery are stored all wineries that are involved inside reviews. For each winery there is a nested array that contains the id of the reviews. In this way it's easy to obtain the review.

```
_id: ObjectId("61b1b43aa520daf17e15226d")
name: "Nica"
reviews: Array
    0: ObjectId("61b0e7cbca9399b7d599e675")
    1: ObjectId("61b0e7cdca9399b7d59a27e7")
    2: ObjectId("61b0e7d0ca9399b7d59ab7be")
    3: ObjectId("61b0e7d5ca9399b7d59b7912")

_id: ObjectId("61b1b43aa520daf17e15226e")
name: "La Voix"
reviews: Array
    0: ObjectId("61b0e7d1ca9399b7d59ae7e5")
    1: ObjectId("61b0e7d2ca9399b7d59b0cea")
    2: ObjectId("61b0e7d2ca9399b7d59b0cf5")
    3: ObjectId("61b0e7d3ca9399b7d59b3a24")
    4: ObjectId("61b0e7d5ca9399b7d59b6367")
    5: ObjectId("61b0e7d5ca9399b7d59b63c2")
```

QUERIES IMPLEMENTATIONS

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

Operation	Mongo Db query
Top ten countries that own most wineries.	db.review.aggregate([{ $group: { _id: { country: "$country" }, wineries: { $addToSet: "$winery"} } }, { $unwind:"$wineries" }, { $group: { _id: "$_id", wineryCount: { $sum:1} } },{$sort:{wineryCount:-1}},{$limit:10}]);

Display twenty wines' varieties according to their mean price.	db.review.aggregate([
	{$match:
	{"price": {$ne:null}
	}},
	{
	$group: { _id: { variety: "$variety" }, prices: { $addToSet: "$price"} }
	},
	{
	$unwind:"$prices"
	},{
	$group: { _id: "$_id", avgPrice: { $avg:"$prices"} }
	},{$sort:{avgPrice:-1}},{$limit:20}]);

Top five users with the highest average of their review scores.	db.review.aggregate([{ $group : { _id : "$taster_name", avg: { $avg: {"$convert":{"input":"$points","to":"int"}}}}},{$sort:{avg:-1}},{$limit:5}])

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 MONGO DB INDEXES

Using indexes for the three Mongo DB queries in our project has no impact on the performance of those queries and that is because of using aggregate functions in those three queries.
We elaborate this fact by analyzing the execution of the following query before and after creating indexes by using the explain function.
The following table shows the indexes created on the review collection:

	1	2	3	4	5
Index Field	_id	taster_name	points	country	winery
Index Type	Default	Simple Index	Simple Index	Simple Index	Simple Index

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)
False	129991	0	129991	355
True	129991	0	129991	347

As we can see from the previous table, having indexes does not have any improvement of the query performance and the indexes have not been leveraged during the query execution. The same logic goes for the other two queries.

Impact of Indexes for find review by winery and find review by taster name:

Using indexes in this case makes queries execution much faster than executing the queries without having indexes.
We elaborate this fact by analyzing the execution of those two queries before and after creating indexes by using the explain function.

- **db.review.find({winery:"A.P. Vin"})**

The following table summarizes the execution performance of the previous query before and after creating an index on winery field in the review collection.

Index	Document Returned	Index Keys Examined	Documents Examined	Execution Time(ms)
False	4	0	129991	144
True	4	4	4	12

- **db.review.find({taster_name:"Jim Gordon"})**

The following table summarizes the execution performance of the previous query before and after creating the index on the taster name field in the review collection.

Index	Document Returned	Index Keys Examined	Documents Examined	Execution Time(ms)
False	4178	0	129991	132
True	4178	4178	4178	23

As we can see from the previous table, having indexes means having magnificent improvement of the queries performance and the indexes have been leveraged during the queries execution.

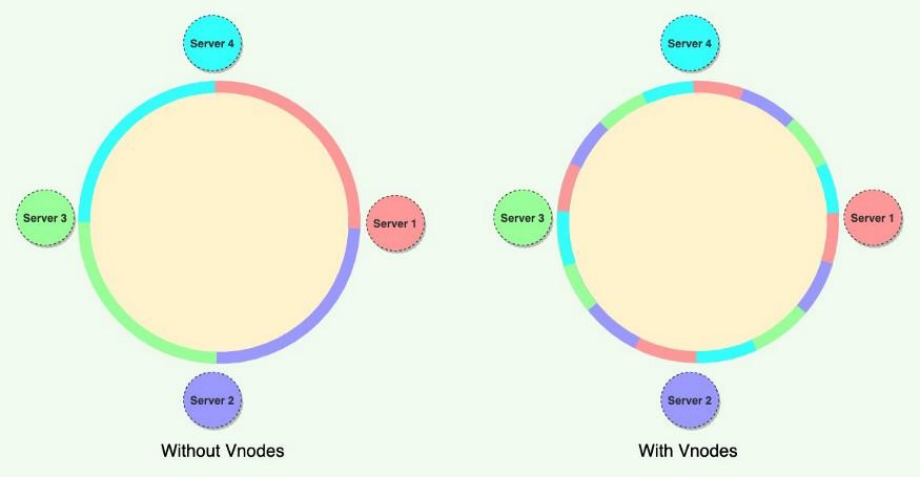
SHARD CONFIGURATION

To divide our data in shards, it has been decided to choose a partitioning algorithm based on hashing, in order to guarantee a balanced load, among the 3 servers we have at the Unipi's disposal.

Consistent hashing is for us, it consists of remapping in case a server is no longer available, and of guaranteeing a balanced load.

Also, since we only have three physical servers, we thought about using virtual nodes.

Vnodes help spread the load more evenly across the physical nodes on the cluster by dividing the hash ranges into smaller subranges, this speeds up the rebalancing process after adding or removing nodes.



NEO4J DESIGN AND IMPLEMENTATIONS

GRAPH DESIGN

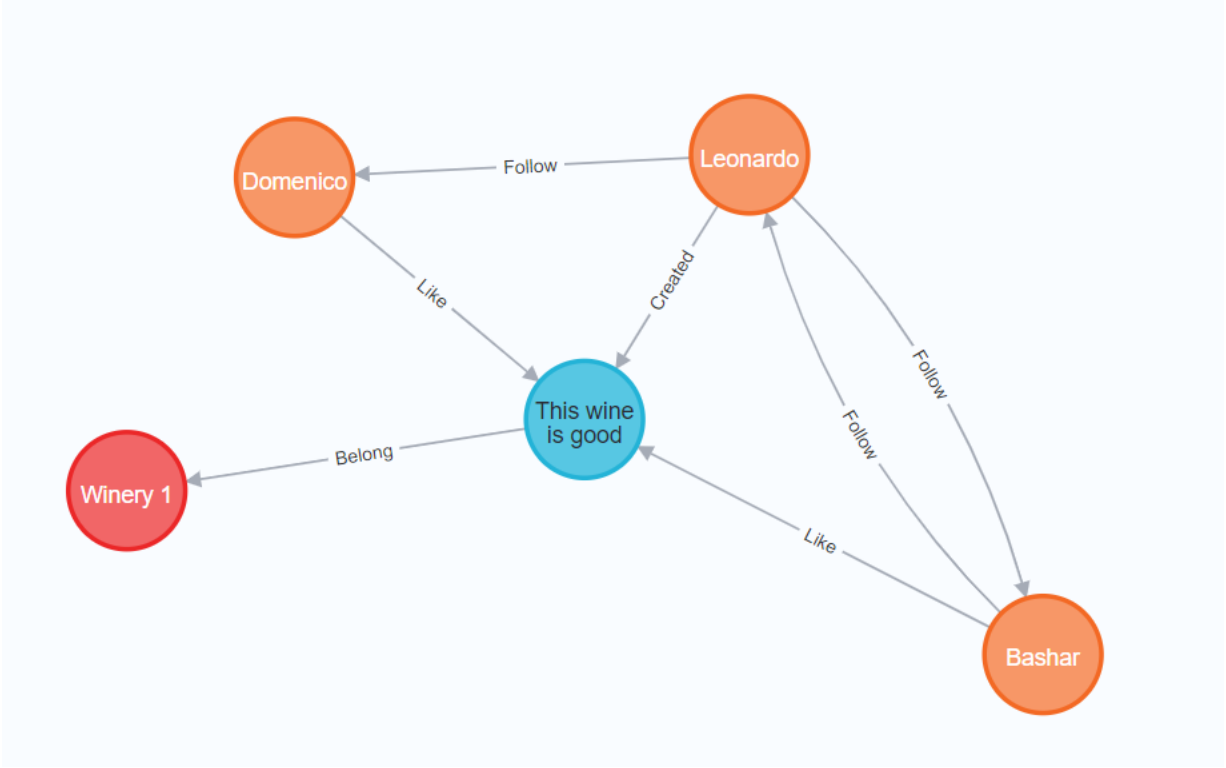
Nodes:

There are 4 types of nodes:

- The **Users Nodes** : represent the user registered on the social network, this node have the “taster_name” attribute;
- The **Posts Nodes** : represent the posts that are written by users, this node has as attributes: "description", "titlePost";
- The **Wineries Page Nodes** : represent the pages of the wineries to which the posts that are written by users are linked. Each winery has many reviews attached to it.

Relationship:

- **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;
- **Winery Page -> BELONG -> Post :** this relationship serves to identify the winery to which the review of that wine is associated.



QUERIES IMPLEMENTATIONS

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

Operation	Cypher implementation
Retrieve all followed user by the name of the user	`"MATCH p=(n:User{taster_name: $taster_name})-[:Follow]->(u:User)\n" + "RETURN u.taster_name AS taster_name"`

Extracts 10 users and creates the follow relationship with the selected user	`"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 posts and creates the Like relationship with the selected user	`"MATCH (p:Post) RETURN p.titlePost as titlePost LIMIT 10"`
	`While:`
	`createRelationLike(titlePost, selected_taster_name);`

Count the graph Nodes	`"MATCH (n) RETURN count(n) as count"`
Recommend friends to the user based on his friendships	`"MATCH p=(n:User{taster_name: $taster_name})-[:Follow]->(:User)<-[:Follow]-(u:User)\n" + "WHERE NOT EXISTS ((n)-[:Follow]-(u))\n" + "WITH u, rand() AS number\n" + "RETURN u.taster_name AS taster_name\n" + "ORDER BY number\n" + "LIMIT 5",`
Retrieve the list of 5 trending posts	`"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"`

IMPLEMENTATION DETAILS

- Functions have been created such as to populate the social network randomly, which can be used by the admin. In the first function called "randomFollowByUser" 10 random users will be extracted and followed by the selected user, in the second function called "randomLikeByUser" 10 posts will be selected randomly and the like relationship will be created between the selected user and the posts;
- 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;
- The "populateSocial" function has been created which, from the reviews of a mongo db database, manages to populate the database in neo4j correctly with all nodes and relationships. So if you have a json / csv file you can populate neo4j using this function
- Every time a relation is created, it is checked if it is already present in order to avoid unnecessary edges;
- In order not to enter the username for social operations every time, a session memory with levelDb has been designed;
- It was considered appropriate to leave the node of the winery pages, because even if not used in the advanced queries of our project, it would have been useful to have them in analysis queries in larger projects to make statistics on reachability or on appreciation.
- in our opinion, the queries made seem to be suitable for a graph database, in fact there are no mathematical operations such as the average, but we try to exploit the structure of the nodes and edges.

LEVEL DB DESIGN AND IMPLEMENTATIONS

IMPLEMENTATION DETAILS

LevelDB it's used only to temporarily store the name of the user logged into the social network, after the syntactic checks. When the user is logged inside the application in the folder "userlog" is saved a file with the entry credentials that the user used.
Each time that a user logged himself inside the application, that file will be overwritten and the entry credentials of the previous user will be lost.
LevelDB was preferred due to his retrieve velocity.

OTHER IMPLEMENTATIONS DETAILS

SCRAPER

- Scraper is link to this website ***https://winemag.com/ratings/#***. It takes all the useful information of each site review: score, username, title, description, twitter name …;
- From this website the scraper takes the review that will be inserted inside the MongoDB Review collection. If the scraper finds reviews that it had previously scraped will not insert them because of checks that are implemented inside;
- With the review it obtains users' data. This data (taster_name) will be inserted, if it doesn't exist, inside MongoDB User_credentials collection;
- Moreover, after the insertion inside MongoDB collections the scraper will insert inside the Social Network the reviews that it has found with *addPostComplete (crud_graph)* function;
- 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 were databases. For this reason the menu is created by using the CLI (Command Line Interface).
Menu are two, depending on which user log-in. The user will declare himself as an Admin or User. After checks, If the user is a Standard User (or Admin because Admin is anyway a user) he will have available all commands regarding Social Network activity, instead, if the logged one is an Admin he will available all command queries to do different statistics of the Social Network.

LOGIN

For the login phase it is required for the user to insert Username and Password. For simplicity the **Standard User** has to log-in with a username already existing on Winemag platform.For the same reason the password is **abcd.**
For the **Admin** the username is **admin** and **password** is root.
To check if the entry credentials are correct, after the insertion, a comparison will be made between the inserted data by user and the username and password stored in user_credentials collection in MongoDB.
If the inserted credentials are equals to what is inside that collection then the menu is shown otherwise the application will stop itself due to incorrect credentials.
Here LevelDB will start its work because after the successful login, username will be stored inside DB and retrieved from methods that allow the user to make operations on the Social Network (add relation between itself and another user or between itself and a post…).