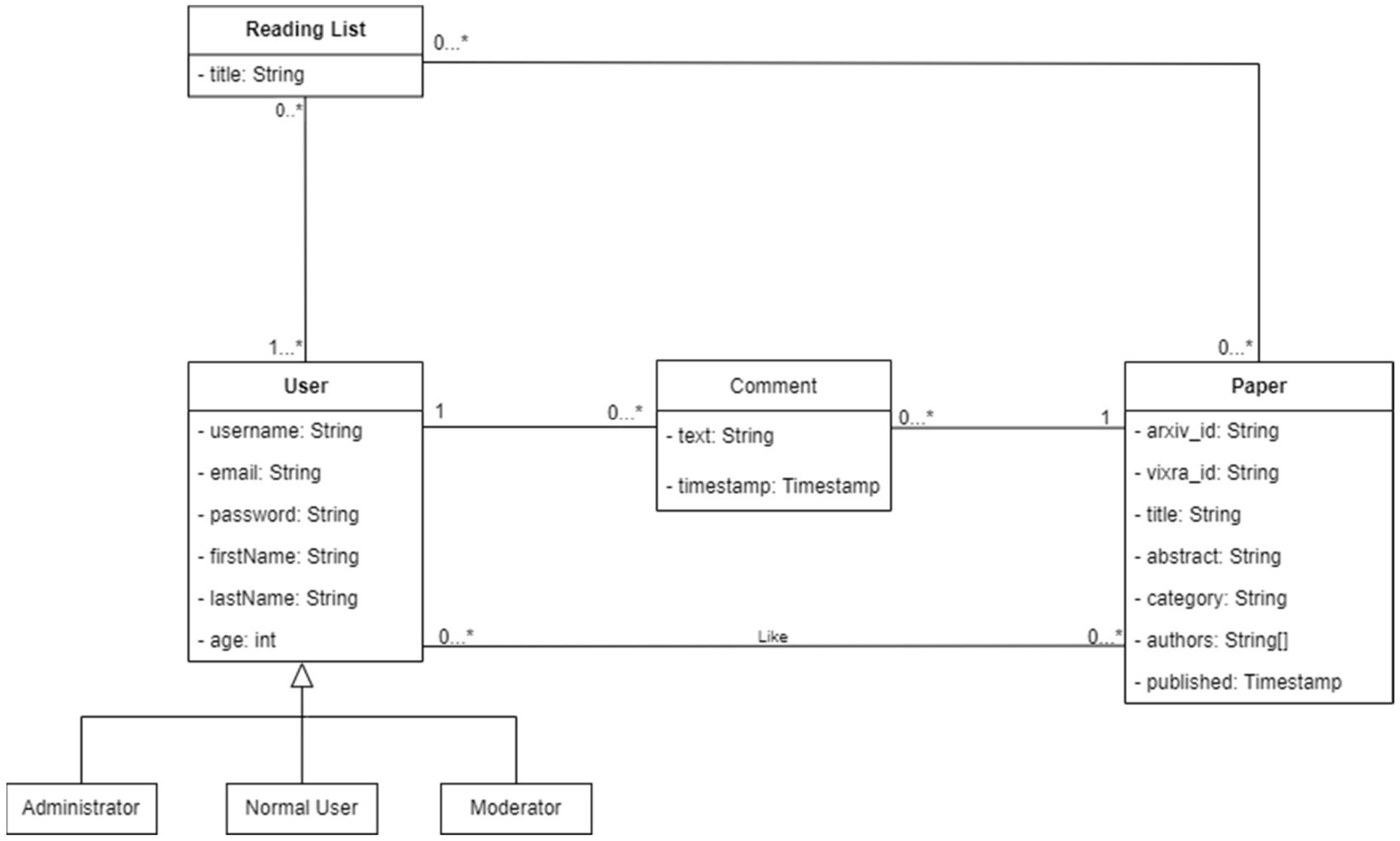
Large-Scale and Multi-Structured Databases

# ***PaperRater***

Edoardo Ruffoli, Tommaso Baldi,

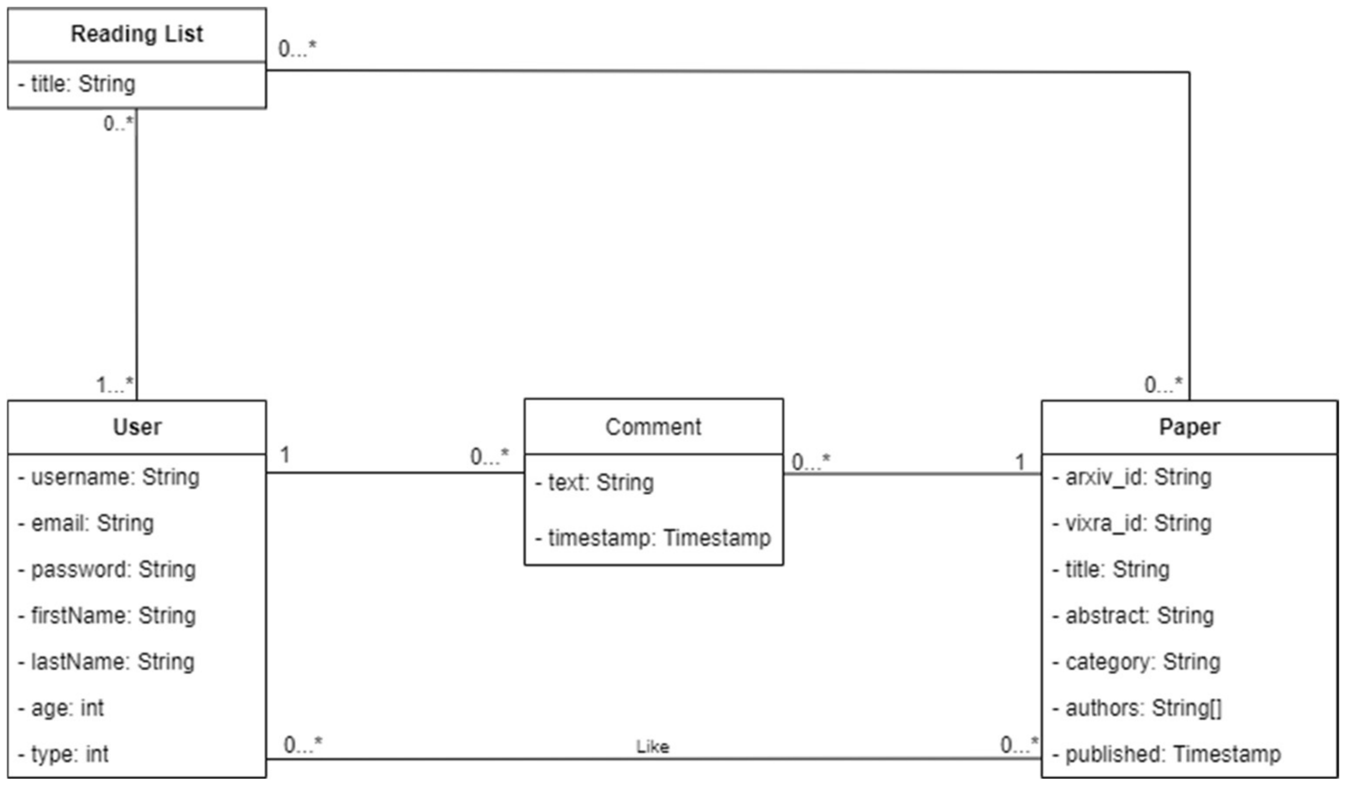
Francesco Hudema

# UML analysis class diagram



There are five main entities: Paper, User, Reading List, and Comment. In the diagram, User is a generalization of the three main actors (Registered User, Moderator, Administrator) of the use case diagram, each actor can perform in addition to its own action, the same action of Registered User.

# UML analysis class diagram



We resolve the generalization adding one attribute to the class user for specify the role of the generic registered user.

## Dataset Description

***Source:***
https://arxiv.org/
https://vixra.org/

***Volume:*** 66.8MB of papers data (50k entities), 36MB from arXiv and 30MB from viXra.

***Variety***: we downloaded papers information from two different sources: the first and main source is arXiv.org an open-access archive that contains almost 2 million scholarly articles in scientific fields; the second source is vixra.org, an e-print archive set up as an alternative to arXiv.org that contains almost 40K scientific papers.

## Dataset Description

**Papers:**

We obtained arXiv papers metadata using the Python wrapper for the arXiv official API (https://github.com/lukasschwab/arxiv.py) while viXra papers metadata were obtained by performing web page scraping using *Beautiful Soup*, a Python library that it is used for parsing HTML documents.

We merged both the metadata and we applied some pre-processing:

We removed possible duplicates and as arXiv and viXra have almost the same paper categories but with similar name, we adapt the names of categories.

**Users, reading lists:**

We populated our database using a Python script that randomly generates 1000 users obtained with *randomuser* API, it also generates a random number of comments and a random number of Reading Lists for every user. We also populated the social network part of the application by adding Like and Follows relations between generated users, papers and reading lists.

## Non-functional requirements

- The application needs to be highly available and always online.
- The system needs to be tolerant to data lost and to single point of failure.
- The application needs to provide fast response search results to improve the user experience.
- Flexibility is needed for manage different sources id of the papers.
- The application needs to be user-friendly so a GUI must be provided.

## Non-functional requirements and CAP theorem

According to the *Non-Functional Requirements*, our system must provide *high availability, fast response times* and to be tolerant to data *lost* and *single point of failure.*
To achieve such results, we orient our application on the **A** (Availability), **P** (Partition Protection) edge of the *CAP triangle*. In our application we want to offer a high availability of the content, even if an error occurs on the network layer, at the cost of returning, to the users, data which is not always accurate. For this reason, we adopt the *Eventually Consistency* paradigm on our dataset.

- **High availability** of the service due to the way we handle the writes operation. In fact, after receiving a write operation we will update one server, and the replicas will be updated in a second moment. In this way writes operation don't keep the server busy for too much time.
- **Partition Protection** of the service is guarantee by the presence of replicas in our cluster, if one server is down, we can continue to offer our service by searching the content in the replicas.

# Database design MongoDB

- Document database manage most of the data regarding users and papers.
- We can have document with missing fields, in our application we use different fields to identify the source of a paper: we use an *arxiv_id* field to identify papers coming from arXiv.com and we use *vixra_id* to identify papers coming from viXra.com.
- Allows us also to embed object that are commonly used together, in our case reding lists inside the user document and comments inside the paper document

**Paper document**

```
{
  "_id": {
    "$oid": "61d9c4e9a11b97f779e3a081"
  },
  "arxiv_id": "1501.00297",
  "title": "Complete Homology over associative rings",
  "_abstract": "We compare two generalizations of Tate homology: stable homology and the\nJ-
completion of Tor, also known as complete homology. For finitely generated\nmodules, we show that
the two theories agree over Artin algebras and over\ncommutative noetherian rings that are
Gorenstein, or local and complete.",
  "category": "Mathematical Physics",
  "authors": [
    "Olgur Celikbas",
    "Lars Winther Christensen",
    "Li Liang",
    "Greg Piepmeyer"
  ],
  "published": "2015-01-01",
  "comments": [
    {
      "text": "Commento",
      "timestamp": {
        "$date": "2022-01-08T06:58:56.000Z"
      },
      "username": "blueswan235"
    },
    {
      "text": "Commento",
      "timestamp": {
        "$date": "2022-01-08T06:58:56.000Z"
      },
      "username": "brownswan112"
    }
  ]
}
```

**User document**

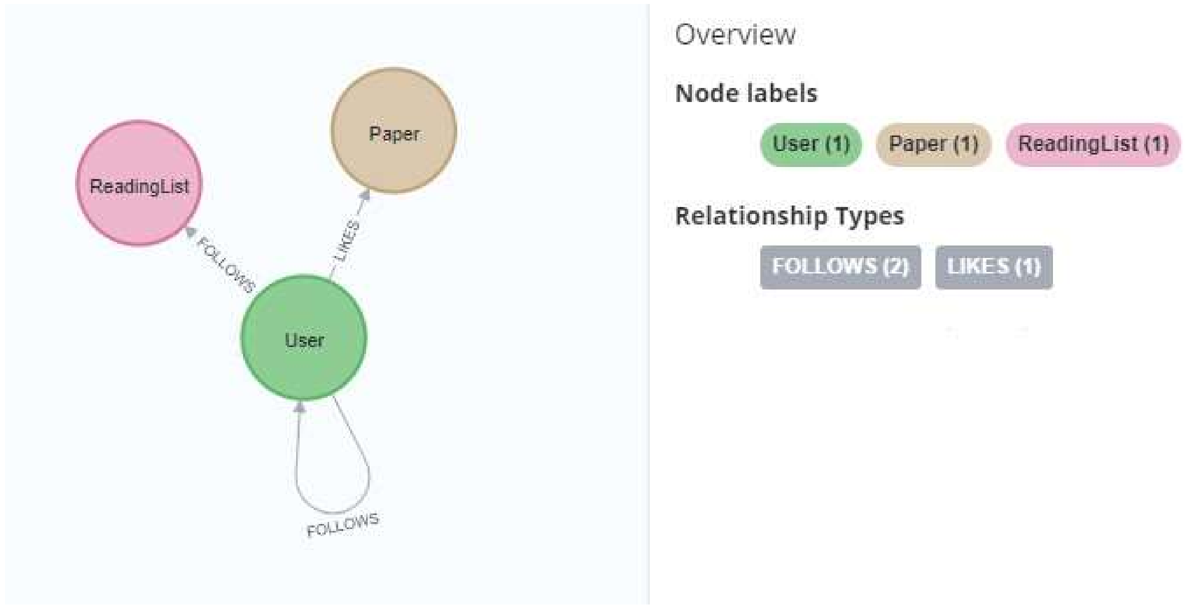
```
{
  "_id": {
    "$oid": "61d9de6ea11b97f779e463f1"
  },
  "username": "happytiger362",
  "email": "ozsu.kaplangi@example.com",
  "password": "sabres",
  "firstName": "Özsu",
  "lastName": "Kaplangı",
  "age": 67,
  "readingLists": [
    {
      "title": "Math_lover",
      "papers": [
        {
          "arxiv_id": "1501.00297",
          "title": "Complete Homology over associative rings",
          "authors": [
            "Olgur Celikbas",
            "Lars Winther Christensen",
            "Li Liang",
            "Greg Piepmeyer"
          ],
          "category": "Mathematical Physics"
        },
        {
          "arxiv_id": "2106.10032",
          "title": "Fourier formula for quantum partition functions",
          "authors": [
            "Andras Suto"
          ],
          "category": "Mathematical Physics"
        }
      ]
    },
    {
      "title": "Favorite<3",
      "papers": []
    }
  ],
  "type": 0
}
```

# Database design Neo4j

Neo4j manages the social part of the application and allow to make suggestions based on the network's connections.

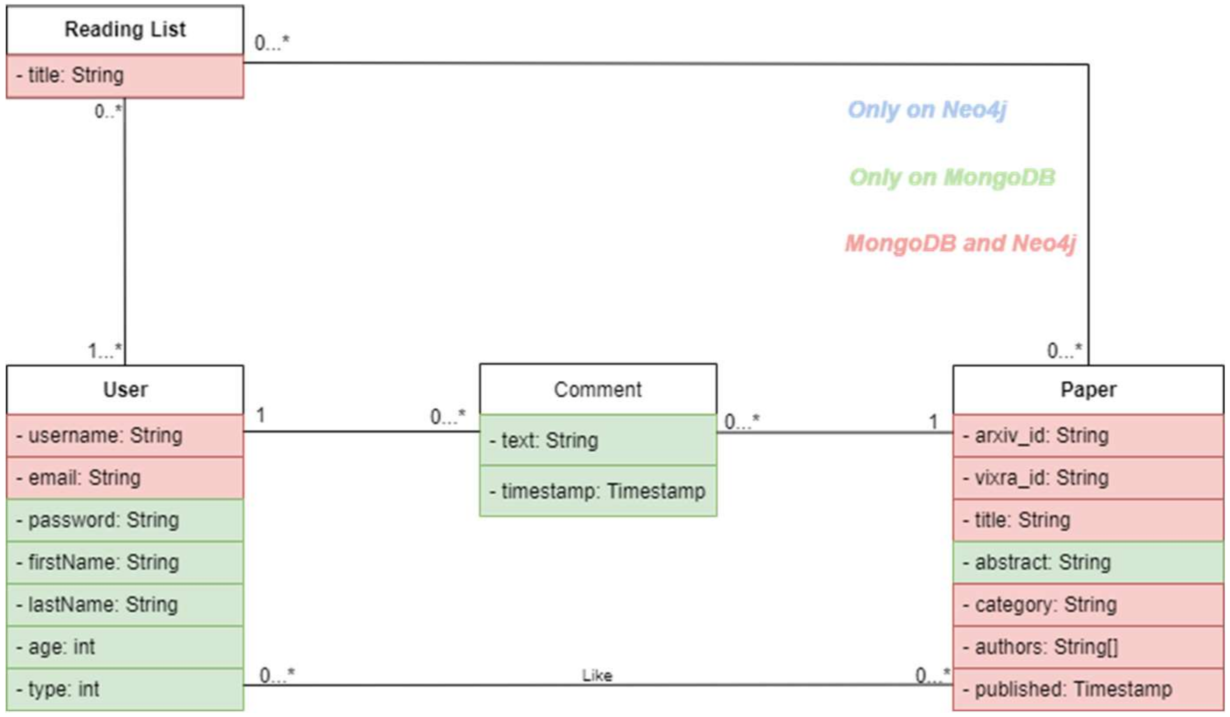
***Entities***:
- Paper
- User
- ReadingList

***Relations***:
- User - Paper: Likes
- User - User: Follows
- User - Reading List: Follows



# Data among databases

We stored the information needed to load a preview (a card) of the various entities also in the graph database so we can display in the application the results of the social network analytics without having to access to MongoDB.
In the following figure is shown the storing strategy.



# Most relevant queries

**MongoDB:**

- **Get most versatile users:** *Select users that have the highest number of categories in their reading list.*
- **Get the most commented papers:** *Select papers with the highest number of comments.*
- **Get categories summary by number of papers published:** *Return the categories ordered by the number of papers.*
- **Get categories summary by comments:** *This function returns the categories ordered by the number of comments.*

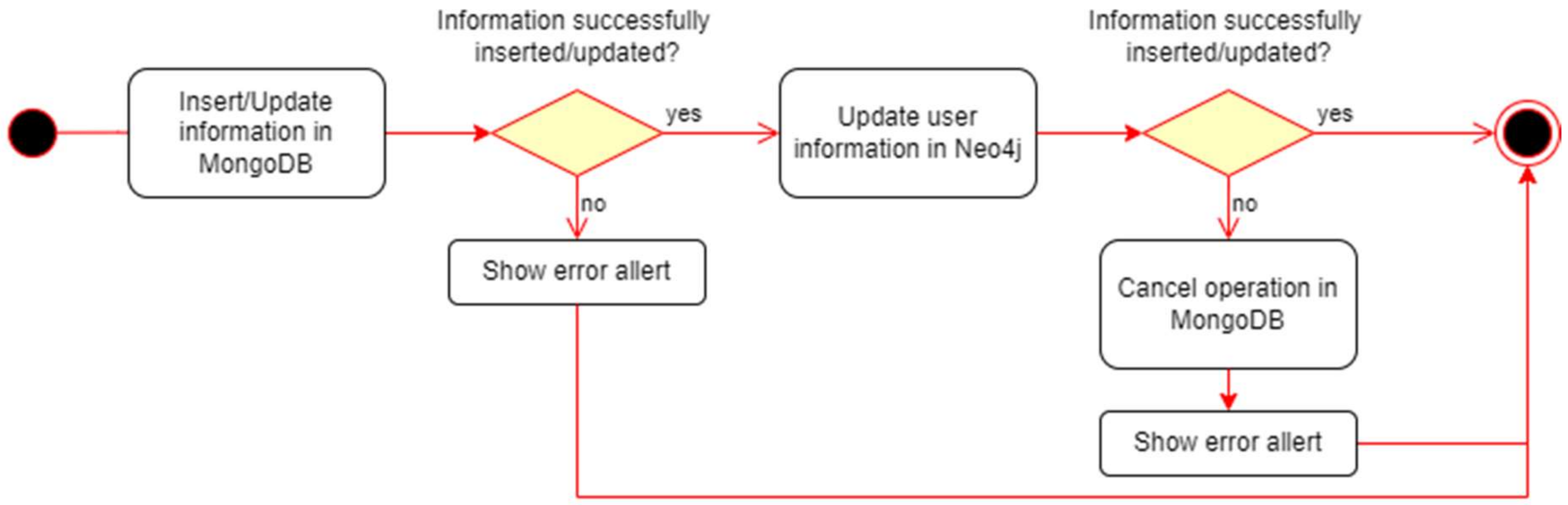
**Neo4j:**

- **Suggested papers:** *The query returns a list of suggested papers for the logged user. Suggestions are based on papers liked by followed users (first level) and papers liked by user that are 2 FOLLOWS hops far from the logged user (second level).*
- **Suggested users:** *The query returns a list of suggested users for the logged user. Suggestions are based on most followed users who are 2 FOLLOWS hops far from the logged user (first level), while the second level of suggestion returns most followed users that have likes in common with the logged user.*
- **Suggested reading lists:** *The query returns a list of suggested reading lists for the logged user. Suggestions are based on most followed reading lists followed by followed users (first level), and most followed reading lists followed by users that are 2 FOLLOWS hops far from the logged user (second level).*
- **Get categories summary by likes:** *This query returns the categories ordered by the number of likes.*

# Consistency

We must consider the problem of consistency on information shared by the two databases in the following case:

- create/delete user
- add/remove reading list
- update user information



Insert/Update information in MongoDB
Information successfully inserted/updated?
yes
no
Show error allert
Update user information in Neo4j
Information successfully inserted/updated?
yes
no
Cancel operation in MongoDB
Show error allert

## Sharding

***Sharding***

To implement the sharding we select two different sharding keys, one for each collection of the document database (User Collection and Paper Collection), for both collections we use as **partitioning method** the *Consistent hashing,* because if the number of nodes of the cluster will change, we can relocate data. The sharding keys are:

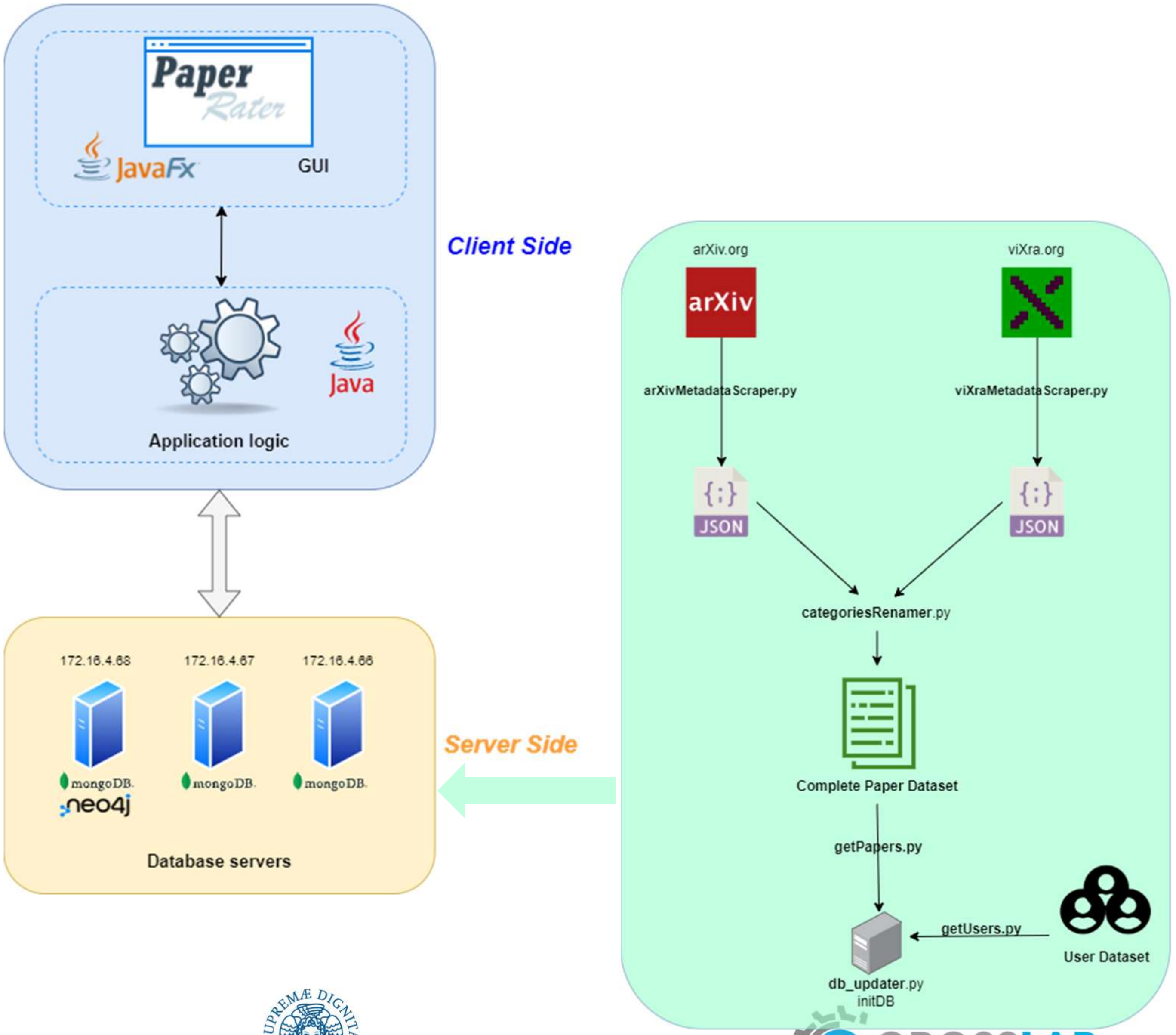
- For the User Collection we choose the attribute "*username",* which is unique among the users, as sharding key.
- For the Paper Collection we choose the attribute "_id", which is automatically generated by MongoDB, as sharding key.

# Software and Hardware architecture

***Programming Language***:
- JavaFX
- Java
- Python

***DBMSs***:
- MongoDB
- Neo4J

***Frameworks***:
- Maven



DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

UNIVERSITÀ DI PISA

CROSSLAB Innovation for industry 4.0

# Final considerations

Our application is read heavy, so we introduced indexes to tune queries that are executed more frequently. One of the main purposes of the application is to provide fast search results regarding papers, so we will focus mostly on indexes related to papers.

Indexes regarding "Papers" collection are particularly convenient because paper documents are not updatable by users. There will not be any overhead related to updates operations and the overhead related to insert operation will affect only the *DB Updater* and not the users.

You can view our application on the link below:

Github link: [https://github.com/edoardoruffoli/PaperRater](https://github.com/edoardoruffoli/PaperRater)

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

UNIVERSITÀ DI PISA

CROSSLAB Innovation for industry 4.0