Large-Scale and Multi-Structured Databases

Project Design

SocialRestaurant

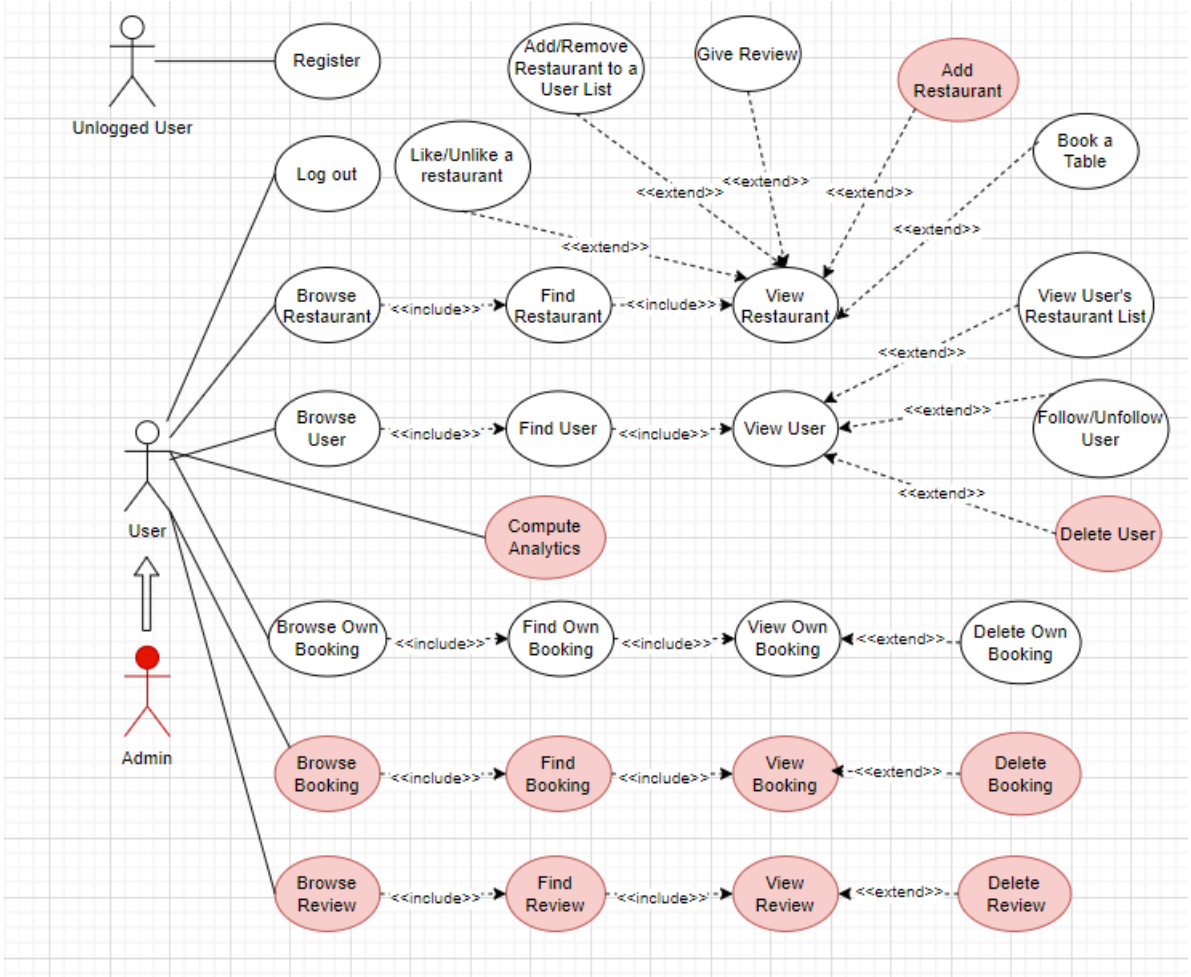
Leonardo Turchetti

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

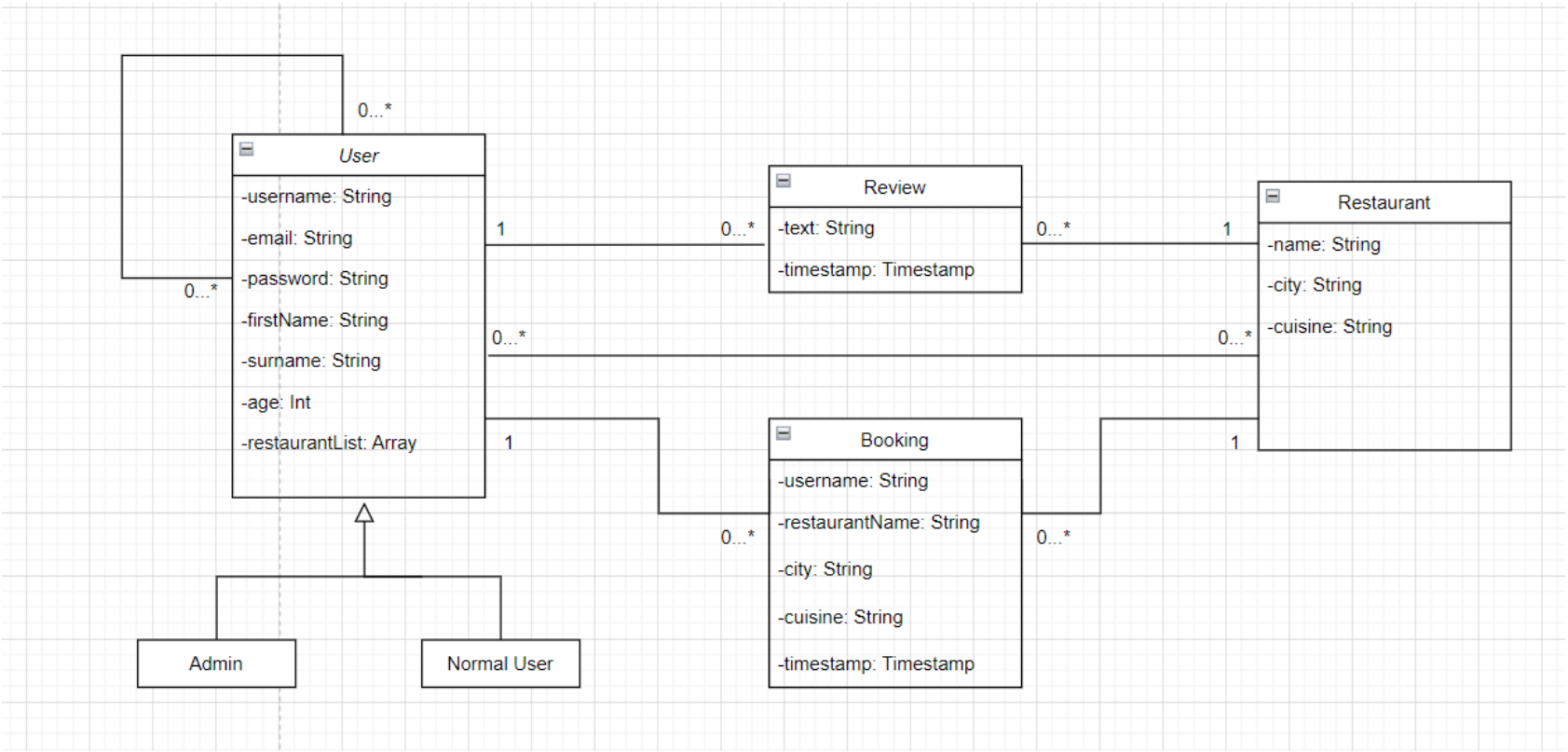
UNIVERSITÀ DI PISA

CROSSLAB
Innovation for industry 4.0

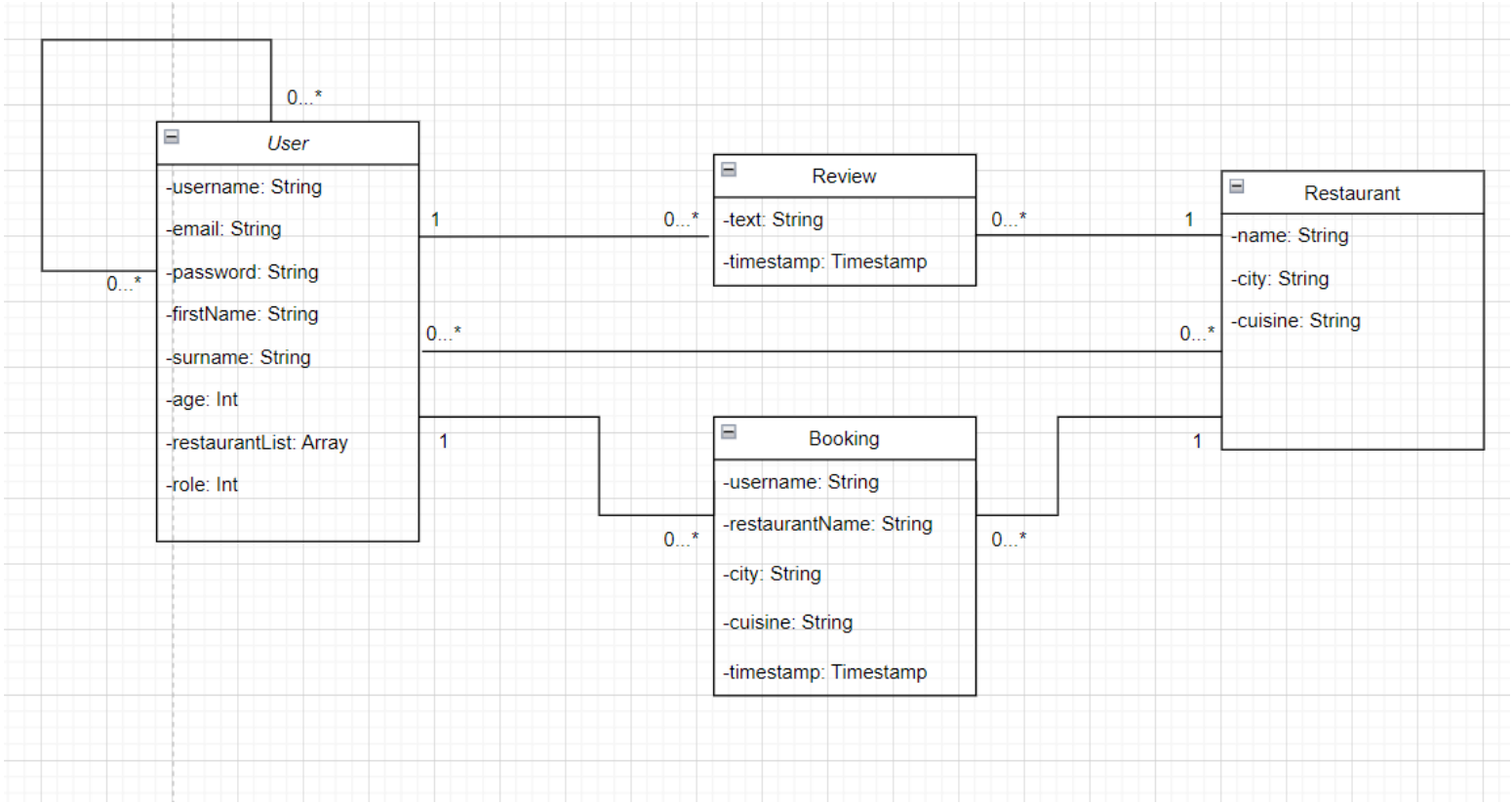
Use Case Analysis



UML Class Diagram Analysis



UML Class Diagram Analysis



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

Dataset Description

Source:

- Trip Advisor (Kaggle)
  https://www.kaggle.com/datasets/stefanoleone992/tripadvisor-european-restaurants
- Zomato (Kaggle)
  https://www.kaggle.com/datasets/himanshupoddar/zomato-bangalore-restaurants

Description:

The first component of the application database are the restaurants. In order to respect the variety constraint we downloaded restaurants information from two different sources: the first source is a dataset from Kaggle that restaurants from European Cities from TripAdisor; the second source is another dataset from Kaggle that includes Bangalore's restaurants from Zomato. The other components of the dataset are the Users and the Bookings and in this case, we populated manually the database about this two type of data.

Volume: 100Kb

Variety:

We use two different sources for building the dataset of the web application

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.
- The application needs to be user-friendly so a GUI must be provided

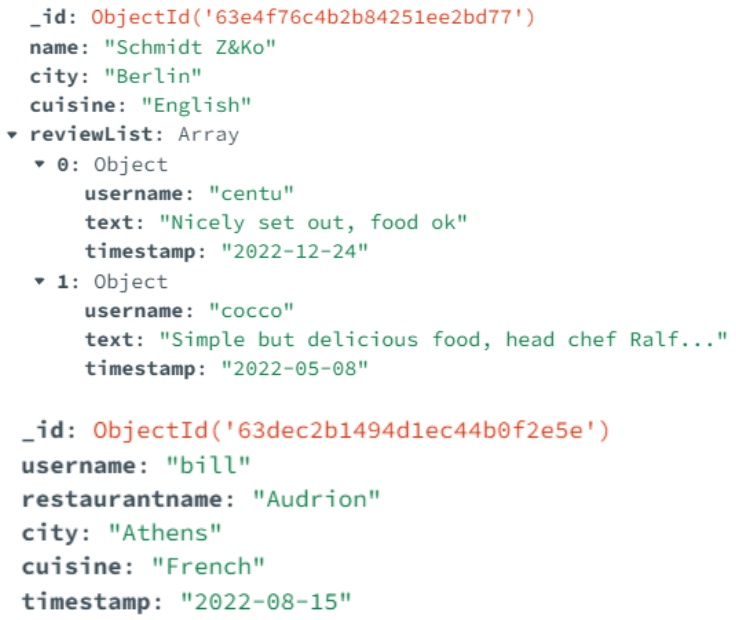
Non functional Requirements and CAP Theorem

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 in MongoDB

- Document database manage most of the data regarding Users, Restaurants and Booking
- Allow us also to embed object that are commonly used together, in our case restaurant list inside the user collection and review inside the restaurant collection



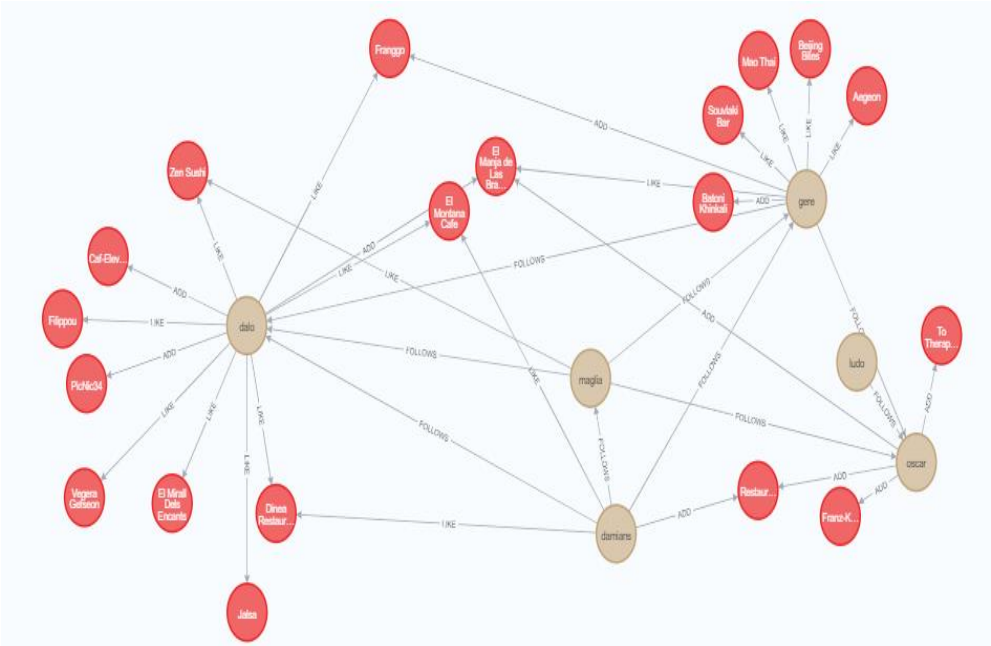
Database Design Neo4j

Entities:

- User
- Restaurant

Relationship:

- User->FOLLOW->User
- User->ADD-> Restaurant
- User->LIKE-> Restaurant



Most Relevant Queries MongoDB

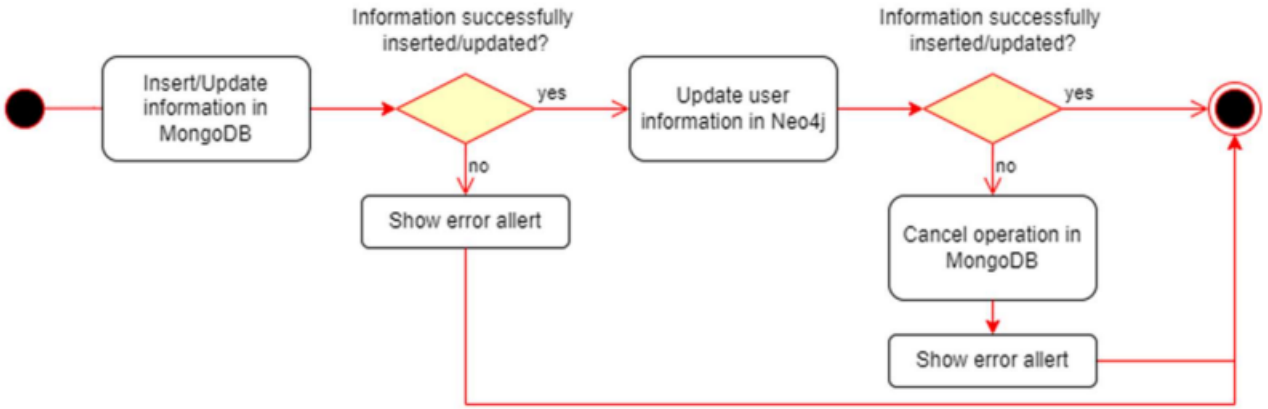
- **Get Most commented Restaurant:** Select restaurants with the highest number of reviews in a specific period.
- **Get Categories summary by Comments:** This function returns the categories of cuisine ordered by the number of reviews in a specific period.
- **Get Cities summary by Booking:** This function returns the cities of ordered by the number of booking in a specific period.

Most Relevant Queries Neo4j

- **Most followed User:** The query returns the list of the most followed users
- **Most liked Restaurant:** The query returns the list of the most liked restaurants in a specific period.
- **Category summary by Likes:** The query returns a list of the most liked categories of cuisine in a specific period.
- **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 Restaurants:** The query returns a list of suggested restaurants for the logged user. Suggestions are based on restaurants liked by followed users (first level) and restaurants liked by user that are 2 FOLLOWS hops far from the logged user (second level). Restaurants returned are ordered by the number of times they appeared in the results, so restaurants that appear more are most likely to be like the interests of the logged user.

Consistency

The operations which require cross-database consistency management are Add/Remove/Update a User Add/Remove Restaurant. For the update of the users, we have only to keep consistency on the field “username”, because it is the only information updatable in Neo4J entity.



Sharding

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

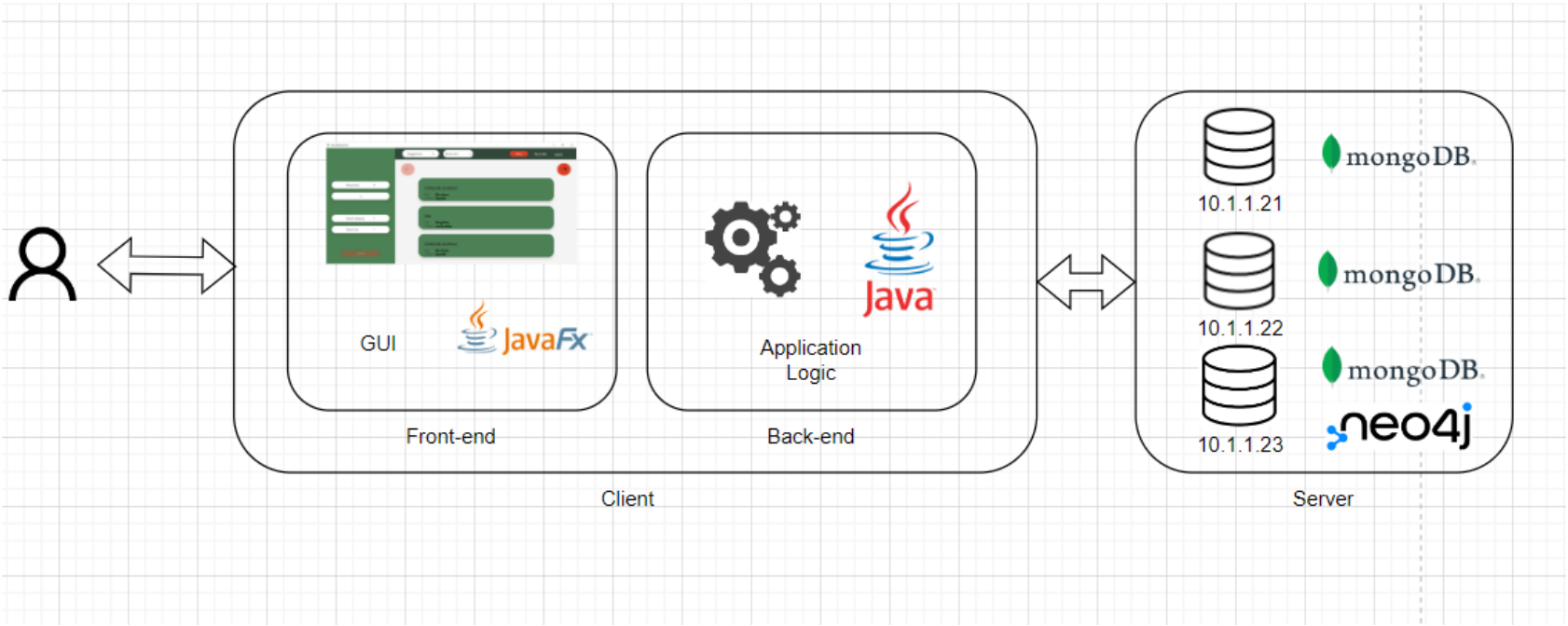
- For the User Collection we choose the attribute "**username**", which is unique among the users, as sharding key.
- For the Restaurant Collection we choose the attribute " **city**" as sharding key.
- For the Booking Collection we choose the attribute "**timestamp**" as sharding key.

Software and Hardware Architecture

Programming Language:
- Java
- JavFX

DBMs:
- MongoDB
- Neo4j

Frameworks:
- Maven



Final Consideration

As the query analysis has revealed, our application is read heavy, so we introduced indexes to tune queries that are executed more frequently.

```
\\Query SearchRestaurantbyName("a")

Result without index:
executionTimeMillis: 0
totalKeysExamined: 100
totalDocsExamined: 100

Result with index "name":
executionTimeMillis: 1
totalKeysExamined: 100
totalDocsExamined: 78
```

```
\\Query
SearchRestaurantbyCuisine("Italian")

Result without index:
executionTimeMillis: 1
totalKeysExamined: 100
totalDocsExamined: 100

Result with index "cuisine":
executionTimeMillis: 0
totalKeysExamined: 100
totalDocsExamined: 13
```

Project repository: https://github.com/Leonardo-Turchetti/Progetto-Large-Scale