UNIVERSITÀ DI PISA

Computer Engineering

Large Scale and Multi-Structured Databases

WORKGROUP TASK 3: DESIGN DOCUMENT

Restaurant recommendations using a graph database

STUDENTS:
Cima Lorenzo
Ferrante Nicola
Pampaloni Simone
Scatena Niccolò

Academic Year 2019/2020

Abstract

This project has been developed as a workgroup during the classes of Large Scale and Multi-Structured Databases. The aim of this workgroup is to give a demonstration of the students' capabilities to handle the development of a simple Java application connecting to a graph database, starting from requirements and arriving to implementation.

Contents

1 Specifications **2**

2 Actors and requirements **3**
2.1 Functional requirements 3
2.2 Non-functional requirements 4

3 Use cases **5**

4 Data Model **8**
4.1 On-Graph Queries 10
4.1.1 Restaurant Recommendations 10
4.1.2 User Recommendations 11
4.1.3 Ranking Restaurant 11
4.1.4 Likes Restaurant 11

5 Architecture **12**
5.1 Server 12
5.2 Client 12
5.3 Third-party softwares 12

1 | Specifications

The application developed for this workgroup task is a tool for **restaurants recommendations provided by a user social infrastructure**.

The main purpose of the application is to allow restaurant owners to make their restaurants known to a large number of people and allow the customers to find the most appreciated restaurants and to create a network of friendships with other users.

The administrator of the application specifies the available cities and cuisines. Those can be set in restaurants' and users' profiles.

Before login, a user must register to the application specifying its username, password and the city where they live.

When logged in, the **user** can see the list of his favorite restaurants and the list of his friends (followed users). He/She can like restaurants or remove previously set likes. Also he/she can follow new friends or unfollow them. He can also specify the type of cuisines he is interested in. A user can view a list of restaurants that the application recommends, based on number of likes put by friends (or friends of friends). Recommendations can be filtered considering the distance between the restaurant's and user's city, the cuisine and the number of likes from the users he follows.

The application also can provide advises for new people to follow with same interests in types of cuisine.

A **restaurant owner** can add a restaurant (or many), inserting all the information as the name, the city where is located and the type of cuisine. He can also modify information whenever he wants. The owner can see the number of his restaurant's likes and see how his/her restaurants are ranked in the city where they are located considering all restaurants of the city or considering restaurants with the same type of cuisine.

2 | Actors and requirements

Main actors:

- The **Users**;
- The **Administrator**.

A user that owns at least one restaurants is also called a **Restaurant's Owner** or simply **Owner**.

2.1 Functional requirements

Registration process New users must register in the system by declaring username, password and the city where they live.

Login process The system shall handle login process with a username and password, a user identifies himself/herself within the system, so the system is able to manage all the data concerning him/her.

User Views The system shall provide appropriate viewers for the users to see list of his favourite restaurants and list of their friends.

Likes/Follow process A user should be able to like a restaurant or to follow a new friend or to see a list of most appreciated restaurants using some filters.

Restaurants handler A restaurant owner should be able to add, and if necessary modify, all the details of his/her restaurants.

Ranking system The system must be able to rank restaurants to do recommendations following filters provided by users.

Administration features The application must have administration features. The administrator can add/remove users, restaurants, cities and cuisines. He can also modify cities and cuisines.

2.2 Non-functional requirements

Usability The application must be easy to use, with a simple and intuitive graphical user interface.

Portability The application must be portable, so that any user can use it, independently from the system that he/she uses.

Data Persistence The application must use a graph database to achieve data persistence.

Concurrency The application must handle multiple users at the same time.

3 | Use cases

Figure 3.1 shows the use cases UML diagram.

The following use cases are defined (all use cases, except Login, requires the user to be logged in):

Login/Logout A user can login in the application using its credentials (username, password). When logged in, he can logout from the application at any time;

Register The first time a user uses the application he/she will be asked to register himself/herself, providing an username, a password, and the city where he/she lives;

Browse Restaurant See the list of all restaurants saved in the application;

Find Restaurant Search restaurants from the list, searching by name or selecting by recommendations parameters;

View Restaurant See all the information about a restaurant;

Put/Remove like Restaurant a customer can put or remove likes to restaurants;

View Statistics View the statistics of the restaurant, as the restaurant's likes and how restaurant is ranked in the city where it is located;

Add Restaurant Add a new restaurant, providing name, cuisine that it serves, price range, city where is located and description;

Delete Restaurant A user can delete a restaurant if it owns it. Moreover, the administrator can delete any user;

Edit Restaurant Modify the details of a restaurant;

Browse User See the list of all users saved in the application;

Find User Search users from the list, searching by name or selecting by recommendations parameters;

View User See all the information about a user;

Follow User A customer can start to follow another customer;

Figure 3.1: Use cases UML diagram.

Unfollow User A customer can stop to follow another customer;

Delete User *(admin)* Remove a user from the system;

Browse City See the list of all cities saved in the application;

Find City Search cities from the list, searching by name;

View City See all the information about a city;

Add City *(admin)* Add a new city, providing name, latitude and longitude where it is located;

Delete City *(admin)* Delete a city among those available in the application;

Change User City A user can change his/her city where he/she lives;

Browse Cuisine See the list of all cuisines saved in the application;

Find Cuisine Search cuisines from the list, searching by name;

View Cuisine See all the information about a cuisine;

Put/Remove like to Cuisine a customer can put or remove likes to cuisines;

Add Cuisine *(admin)* Add a new cuisine, providing its name;

Delete Cuisine *(admin)* Delete a cuisine among those available in the application;

View User Preferences Show the preferences of a user (i.e. the city where the user lives and the cuisines she/he likes);

Change User City Change the city where a user lives;

The *Restaurant Recommendation Parameters* allow to select restaurants that the customer does not known by number of friends' likes, by the city in which they are located (or by the distance from the city where the customer lives), by the cuisine they offer and by the price. This is done by running the query defined in Section 4.1.1 "Restaurant Recommendations".

The *User Recommendation Parameters* allow to select users not followed by the customer by filtering them by the city in which they live (or by the distance from the city where the customer lives) and the types of cuisine they like. This is done by running the query defined in Section 4.1.2 "User Recommendations".

The *Find City* and *Find Cuisine* use cases include the case when the user must select a city/cuisine (e.g. when filling a form) and the application gives the user suggestions on the available cities and cuisines.

4 | Data Model

Figure 4.1 shows the analysis classes of the data model used by the application.

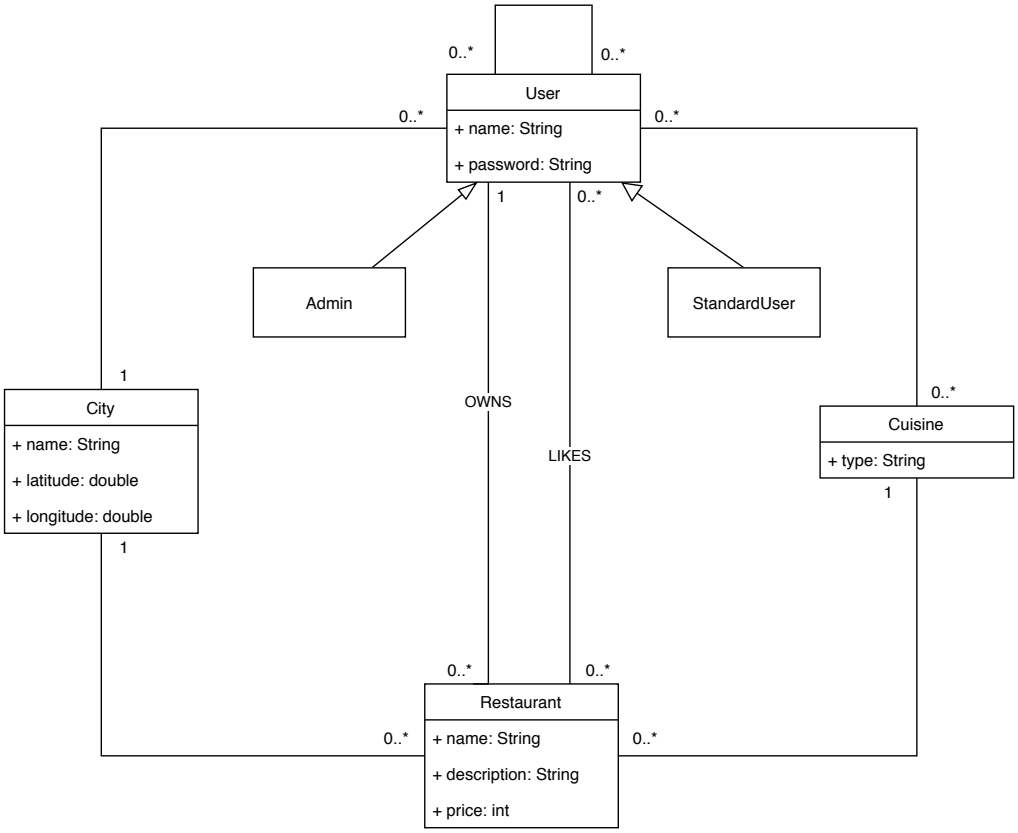

Figure 4.1: Data model UML diagram (analysis classes).

It's worth noting that the relationships between `User`/`Restaurant` and `City`, and the relationship between `Restaurant` and `Cuisine`, may not be always present. This happens, for example, when the administrator deletes a `City`/`Cuisine` that is related with a `User`/`Restaurant`. In this case, the application will show `N/D` (Not Defined) when the user views a `User` or `Restaurant` with a deleted relationships. The user or the restaurant's owner may re-add the relationship with the `City`/`Cuisine` afterwards.

Figure 4.2 shows the metagraph of the database. The blue node is a User node. Figure 4.3 shows a snapshot of the database.

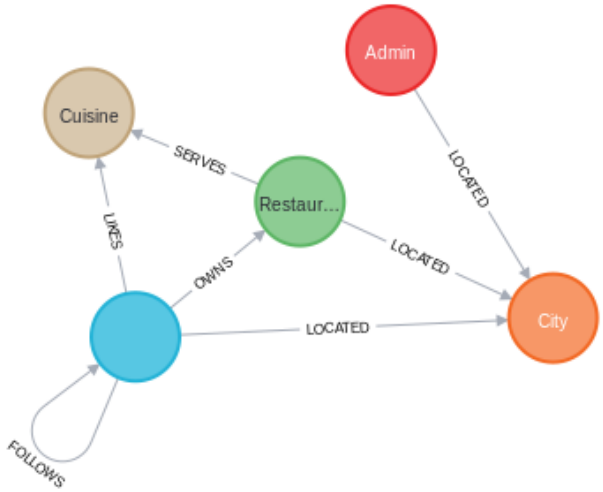

Figure 4.2: Metagraph with all nodes, labels and relationships.

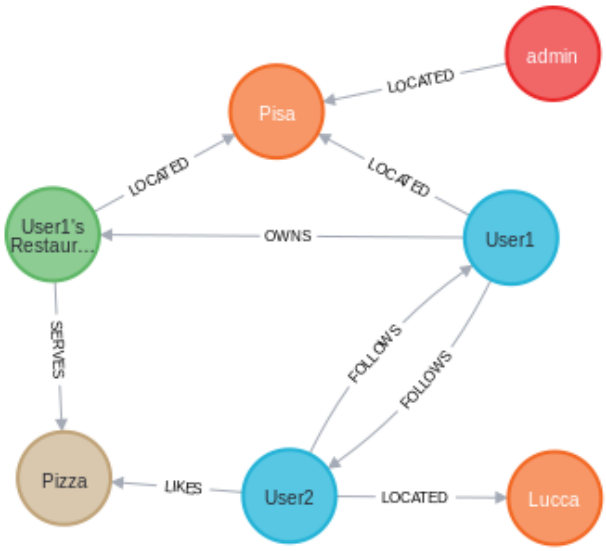

Figure 4.3: Graph snapshot (example).

The following nodes and relations are defined:

User *(node)* Represents a user registered in the system. It contains the following properties:

name The name of the user, provided during registration;

password The hash of the password used by the user to login.

Additionally the Administrator has an additional label **Admin** to distinguish him from other regular users.

Restaurant *(node)* Represents a restaurant. It contains the following properties:

name The name of the restaurant;

price An enumerated type that represents the expensiveness of the restaurant;

description A textual description of the restaurant, provided by the restaurant's owner.

City *(node)* Represents a city. It contains the following properties:

name The name of the city;

latitude The latitude, used to compute distances;

longitude The longitude, used to compute distances.

Cuisine *(node)* Represents a cuisine type. It contains the following properties:

name The cuisine name (e.g. Pizza).

OWNS *(relation)* Connects a restaurant with his owner;

LOCATED *(relation)* Connects a user or a restaurant with the city where he/she lives in or where it is located;

FOLLOWS *(relation)* Connects a user with another user. A user receives recommendations based on the likes of the users he follows;

LIKES *(relation)* Connects a user to the restaurants/cuisines that he likes;

SERVES *(relation)* Connects a restaurant to the type of cuisine that it serves.

4.1 On-Graph Queries

Four typical "on-graph" queries have been defined. The queries allow to implement the function of restaurant recommendations, user recommendations and statistics of restaurant of the application on the graph database.

4.1.1 Restaurant Recommendations

Domain-Specific Find restaurants, which a given user has not liked yet, which are located in a given city, or are located a given number of kilometers from a given city, which make a certain cuisine and which have a certain price. Sort restaurants based on the number of likes of friends, or friends of friends, of the given user.

Graph-Centric Find the restaurant nodes that have a `LOCATED` relation with a given city or that have a `LOCATED` relation with a city whose distance from the given city is under a given number of kilometers, and have a `SERVES` relation with a given cuisine and with a given value of price attribute. For each resulting restaurant node, count the number of `LIKES` relation that it has with the user nodes that have a `FOLLOWS`

relation to the given user or a `FOLLOWS` relation with user nodes that have a `FOLLOWS` relation with the given user. Sort resulting restaurant nodes for the number of `LIKES` relation.

4.1.2 User Recommendations

Domain-Specific Find users who are not followed by a given user, who lives in a given city, or lives a certain number of kilometers away from a given city, who likes a certain cuisine. Sort users by distance from the given city.

Graph-Centric Find the user nodes that have a `LOCATED` relation with a given city or that have a `LOCATED` relation with a city whose distance from the given city is under a given number of kilometers, and have a `LIKES` relation with a given cuisine and the given user does not have a `FOLLOWS` relation with them. Sort resulting user nodes by distance with the given city.

4.1.3 Ranking Restaurant

Domain-Specific Given a restaurant, find its position in the ranking of restaurants, sorted by the number of likes received by users, that are located in the same city, or that make the same cuisine of the given restaurant or both.

Graph-Centric Find restaurant nodes that have a `LOCATED` relation with the same city node with which the given restaurant has a `LOCATED` relation or that have a `SERVES` relation with the same cuisine node with which the given restaurant has a `SERVES` relation or both of them. For each resulting restaurant nodes and for given restaurant, count the number of `LIKES` relation that it has with user nodes. Sort the resulting restaurant nodes for the number of `LIKES` relation and return the position of the given restaurant.

4.1.4 Likes Restaurant

Domain-Specific Find the number of users who have liked a certain restaurant.

Graph-Centric Count the number of user nodes that have a `LIKES` relation with the specified restaurant.

5 | Architecture

The application is compound by two components: Server; Client.
Figure 5.1 shows the application's architecture.

5.1 Server

The server contains the core logic of the application. It receives commands from the client, performs the required operation, and returns the result to the client. Even administration commands are handled by the server.

5.2 Client

The client is the program run by the user. It just reads the input from the user from which builds the requests for the server and send them out. When the server responds, the client shows the results to the user and waits for new input. A graphical interface is provided: the user will issue commands by selecting tables' elements and by filling forms.

5.3 Third-party softwares

The following frameworks and libraries are used:

Neo4j OGM Java driver v3.2.14 *(Server)* the Object-Graph Mapper of Neo4j;

Neo4j OGM Bolt driver v3.2.14 *(Server)* for connecting to Neo4j with the Bolt protocol;

XStream v1.4.12 *(Client, Server)* for XML serialization/deserialization to transmit objects over the network;

Apache Commons CLI v1.4 *(Client, Server)* for command-line options parsing.

In the final implementation of the application, newer versions of the aforementioned dependencies may be used (if they will be released during the development). All dependencies are handled with Maven. Other frameworks or libraries may be added in future.

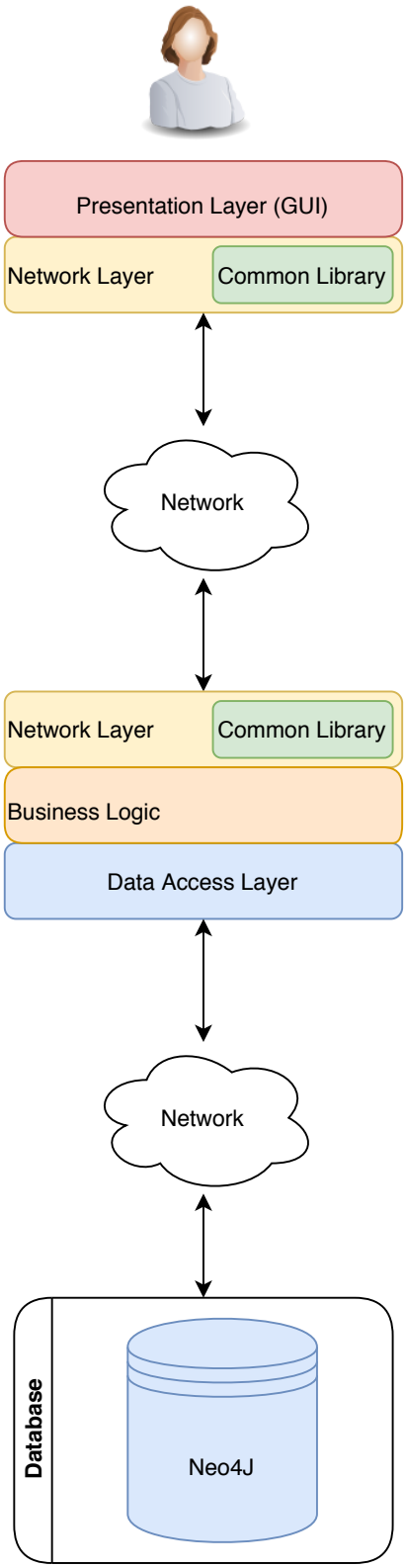

Figure 5.1: Application architecture.