M149/M116: Database Management Systems
University of Athens
Deprt. of Informatics & Telecommunications

Programming Project II
Today's Date: December 12^{th}, 2020
Due Date: January 17^{th}, 2021

PREAMBLE

In this project, you will design, implement and demonstrate a *NoSQL*–database solution to manage *"311 Incidents"* data openly published by the city of Chicago, IL. You will also provide access to your database through a *REST API*. Residents of Chicago, may dial *311* to report incidents that affect their every day life in the city.

Reports are used by the city to offer recovery, sanitation and municipal services to the public. The database termed "`NoSQL 311-Chicago-Incidents`" or `NoSQL-311CI` will be populated by data available at:

`https://www.kaggle.com/chicago/chicago-311-service-requests`

Pertinent data of the above data set have to be *stored* in `NoSQL-311CI` and can be used when the system becomes operational.

PROBLEM DESCRIPTION:

Your database should include information about the *311* incidents that have been reported and should provide various queries and/or aggregations on incidents recorded. Moreover, the database should hold citizen "*upvotes*" for the reported incidents. In particular, a citizen may vote up any recorded incident to highlight its importance. An upvote must be accompanied with a name and a telephone number. Below is a description of the general guidelines of the project. While you are working, keep in mind that these items make up a minimum set of requirements. Hence, you may extend the scope of your work as you see it appropriate and/or interesting.

311 Chicago Incidents data: the city provides its open data so that citizens can be assisted in making decisions about their every day life. Moreover, these data help pinpoint areas that the municipality has to timely expend effort and funds to improve life. `NoSQL-311CI` users are either residents or individuals carrying out data studies for enhancing their wards or neighborhoods by designing coordinated civic action(s). In this context, users can analyze the data provided either though a set of web–service requests or via having read-access to the database (for the more sophisticated).

There are *12* sets of information (files in `.csv` format) used to keep track of all types of incidents being reported in Chicago, IL. We are interested in the following *11* incidents reported:

1. Abandoned vehicles: vehicles reportedly open and left unattended on roads, highways and public spaces of the city that have to be attended to. Number of days of observed current status for the vehicle at the time of the report is provided as well.
2. Alley Lights Out: Lights on wooden poles reported as out (one or more). Duplicates might exist for multiple similar reports might have been filed.
3. Garbage Carts: New requests for carts are made to the city when existing one have gone missing and/or are (partially) destroyed. Duplicate requests might exist.

4. Graffiti Removal: List requests for graffiti removal and vandalism recovery services. Duplicates might exist as reported by different parties.

5. Pot Holes Reported: The city oversee 4,000 miles of road surfaces and receives reports for holes created on tarmac due to weather, accidents, and structural damage. Multiple requests may originate from different residents for the same incident.

6. Rodent Baiting: reports for rodents in public areas are reported and the Department of Sanitation has to react through its Ward Sanitation Branches. Duplicate requests might exists.

7. Sanitation Code Complaints: violations including overflowing dumpsters and garbage bin are reported and Depart. of Sanitation has to address the incident report that remains open for multiple parties reporting.

8. Lights All Out: reports the outage of three or more lights in city locations. The Depart. of Transportation oversees 250,000 street lights and has to rectify reported incidents. Duplicate requests may exist and remain open until addressed.

9. Street Light One Out: As above but with only one or two light out.

10. Tree Debris: Piles of branches appearing in public areas, streets and open spaces are reported fro removal. The same goes for broken trees that have to be removed by city trucks.

11. Tree Trims: requests for all tree shortening and trimming are listed. Crisis trimming is performed when hazardous conditions for the public exist. Open-duplicate requests may exist.

Most of the attributes in the above files are self-explanatory.

Using `MongoDB` Community Server,[1] create the *collections* of your database, and insert *all* pieces of incident information. You have the freedom to define the collections as per your own choice/design.

Citizen Data: should include the upvotes for each citizen, accompanied with the name, telephone number and address of the citizen.

REST API: You should create an API method for each of the following queries, as they are particularly useful to the context of `NoSQL-311CI`. The responses of your API should use the JSON format:

1. Find the total requests per type that were created within a specified time range and sort them in a descending order.

2. Find the number of total requests per day for a specific request type and time range.

3. Find the three most common service requests per zipcode for a specific day.

4. Find the three least common wards with regards to a given service request type.

5. Find the average completion time per service request for a specific date range.

6. Find the most common service request in a specified bounding box for a specific day. You are encouraged to use `GeoJSON` objects and Geospatial Query Operators.

[1] `https://www.mongodb.com/download-center/community`

7. Find the fifty most upvoted service requests for a specific day.
8. Find the fifty most active citizens, with regard to the total number of upvotes.
9. Find the top fifty citizens, with regard to the total number of wards for which they have upvoted an incidents.
10. Find all incident ids for which the same telephone number has been used for more than one names.
11. Find all the wards in which a given name has casted a vote for an incident taking place in it.

In addition, you should provide API methods for updating the database. Updates may include *i)* inserting new incidents and *ii)* casting of upvotes. In case the same user casts a vote for the same incident a second time, the vote should be rejected.

It is particularly important that you make sure that the above queries are executed *efficiently* by creating the appropriate *collections* and adding the necessary *indices*.

IMPLEMENTATION ASPECTS:

You will use `MongoDB` as your database in this project. In addition, you may use any language/framework you want including `Spring-boot`, `Laravel`, `Django`, `Ruby On Rails`, `Flask`, `Express.js` `PHP, Java, Python, PHP`, etc.

You may work in any environment you wish but at the end you should be able to demonstrate your work (with your notebook or via remote access to a machine).

OVERVIEW OF PROJECT PHASES:

There are two distinct phases in this project that you will need to work on:

◇ Database

In this phase, you are required to decide upon the database collections you will use and populate them with data. For the citizen data you are expected to generate the data yourselves. To this end you could employ an open-source library such as `faker`[2][3], `java-faker`[4], and `datafactory`.[5] You are expected to generate enough data so that at least $\frac{1}{3}$ of the reported incidents have at least one upvote, and no user has more than 1,000 upvotes.

◇ REST API

You will use the framework of your choice (such as `Spring-boot`, `Laravel`, `Django`, `RoR`, `Flask`, `Express.js` etc.) to provide an API that offers access to the database and enables users to query and update the database.

[2] `https://github.com/stympy/faker`
[3] `https://github.com/joke2k/faker`
[4] `https://github.com/DiUS/java-faker`
[5] `https://github.com/andygibson/datafactory`

COOPERATION:
You may either work individually or pick *at most one partner* for this project. If you pick a partner you should let us know who this person is.

REPORTING:
The final *typed* project report (brief report) must consist of:

1. A description of the schema design of the database used along with justification for your choices.
2. The `MongoDB` queries for the required functionality.
3. The code of your *REST API*, preferably through a link to a `git` repository.
4. Sample responses for each query.

Finally as mentioned earlier, you will have to demonstrate your work.

SUPPORT:
Panagiotis Liakos (`p.liakos+@-di.`) and George Metaxopoulos (`cs3190002-@-di.` will be escorting this assignment, fill in questions, and carry out the final interviews.