Task: Bookmarks and graphs

Cloud engineering

Cloud engineering in Memgraph is all about getting a Memgraph instance up and running within a few clicks in your browser. We take the day to day hassle of management operations out of your hands so you can focus on building your business.

1. Goal

The goal is to build a simple REST API server application in Node.js that is used to handle bookmark information stored in Memgraph.

API Endpoints:

It is up to you to define the REST API endpoints and request parameters, but the following user requests should be handled:

1. Save a bookmark that contains the following parameters:
    a. **`url`** - valid textual HTTP/S url *[required and unique parameter]*
    b. **`tags`** - list of textual tags where every tag consists of the following characters: a-z, 0-9, and hyphen (-) *[required parameter]*
    c. **`author`** - author username, consists of the following characters: a-z, 0-9, and hyphen (-) *[optional parameter]*
2. Get a saved bookmark (**`id`**, **`url`**, **`tags`**, **`author`**) by bookmark Id
3. Get a list of bookmarks (**`id`**, **`url`**, **`tags`**, **`author`**) - sorted by created date where the last added bookmark is the first one in the response
4. Get a list of bookmarks (**`id`**, **`url`**, **`tags`**, **`author`**) filtered by author name defined as an input
5. Get a list of bookmarks (**`id`**, **`url`**, **`tags`**, **`author`**) filtered by tags defined as an input. It should return bookmarks that contain any of the input tags where bookmarks are sorted by how many input tags it contains. Users can define between 1 and 15 different tags as inputs to REST API. *(e.g. for input tags "graph", "theory", "db", it should return all bookmarks that contain any of these three tags, but bookmarks that contain all three should be the first one in the response followed by bookmarks that contain only two of the tags followed by bookmarks that contain only one of the input tags)*
6. Get a list of unique tags (**`tag`** value, **`tag`** frequency) with the ability to return them sorted by frequency in all stored bookmarks. By default, returned tags should be sorted by the created

date where the last added tag is the first one in the response. *(e.g. frequency is a ratio of how many bookmarks tag belongs to compared to the total number of bookmarks)*

Model:

Regarding the bookmark model, input fields, as noted above, are: **`url`**, **`tags`**, and optional **`author`**. Feel free to add any other fields (e.g. id, createdAt, updatedAt) that will help you in the implementation of the service.

As bookmarks with tags and authors should be stored in Memgraph, it is also up to you to define a graph model where you can decide which objects will be nodes and how you will be connecting them with edges.

The server should start by running command **`npm start`** as noted below:

Examples

```
$ npm start
Application started and listening on port 3000
```

Server:

- Should be listening on port 3000
- There is no need to add any authentication or authorization
- Feel free to combine and use any Node.js framework or library
- Structure the project and directories however you like, but **`npm install`**, **`npm start`** and **`npm test`** should work from the root directory

Memgraph:

- Download community version of the Memgraph database from Memgraph Download Hub
- Use the Cypher query language to manage data from/to Memgraph database
- Use any Node.js library that supports Bolt protocol for connecting your service to Memgraph (check Appendix for guidance)
- Use Memgraph Lab to run Cypher queries and visualize results from Memgraph or Memgraph Client to run Cypher queries in terminal

2. Requirements

Main requirements:

- Working E2E solution with all 6 user requests noted in API endpoints
- Baseline technology is Node.js (version 12 or above)
- Data is stored in Memgraph database

- At least one automated test case with Jest or any other Node.js testing tool that should be runnable by calling **`npm test`**

Bonus requirements:

- The project should be runnable with Docker — setup **`docker-compose.yml`** for the service and Memgraph database
- Typescript is used instead of Javascript

3. Process

Once you receive a task, you can ask us anything about the task (or life in general). Structure and organize your questions wisely because you can send us a list of questions **only twice** during this process. Once you are done with the project, send us a link to the git repository or zip the project and send it as a .zip archive.

4. Review

The idea is to present a correct, simple, and complete solution for the given task. The solution will be evaluated upon various criteria such as correctness, software design, proper testing, maintainability, extensibility, documentation, performance.

5. Appendix

Any Node.js library that supports Bolt protocol can be used to connect to Memgraph. We recommend using Neo4j's Node.js library, neo4j-driver. Make sure to use version **`1.7.6`** because any newer version doesn't work with Memgraph Bolt protocol implementation. For further details on how to use a driver, make sure to check out the official Neo4j Driver Manual, version 1.7.

File **`package.json`** should look like this:

```
{
  ...
  "dependencies": {
    ...
    "neo4j-driver": "^1.7.6",
    ...
  }
}
```