Take home exercise for backend

Design a system that collects, processes and stores real-time RSVPs from Meetup.com and provides methods to query them to extract useful information.

The system must be able to listen to a real-time RSVP stream of events in meetup.com. The stream will send a new message each time someone wants to attend a specific meetup. Using those real-time messages as a source of information it must be able to implement the endpoints explained in the "REST endpoints" section below. For documentation on the available streams and their format, see: https://www.meetup.com/meetup_api/docs/stream/2/rsvps/

From all the meetup.com API methos available, only the stream APIs can be used, you can't rely on any other endpoint to fulfil the requirements.

To implement this system you can use any technologies you feel comfortable with. We want you to implement the solution using the tools you think are the best suited for the problem. This is valid for programming language, frameworks, build-dep managers, storage systems, etc. The system can be split into as many applications/services as you think is needed.

REST Endpoints

GET /near

Given `lat` and `lon` (latitude and longitude), return the `num` closest Groups in distance (using a 2D distance measure is OK)

Messages may include several locations for different entities. Use the Group's location for this endpoint.

GET /topCities

Given a `date` (using the ISO date standard) return the top `num` cities sorted by the number of people attending events in that city on that day.

Useful information

Most likely, for each message received you'll need to make several database requests (or whatever storage system you choose) to be able to fulfill the requirements.

Try to make code design choices so the system follows SOLID principles and is easy to maintain and extend in the future: obtain data from more sources, process the data in new ways, create new reports, ...

How is this exercise evaluated?

The tools you decide to use don't matter as long as they are a good fit for the problem. Use any language and build system you feel comfortable with (like scala+sbt, java+maven, python+pip*setuptools, etc). The same for database*storage system or any other framework you may want to use.

We'll look for proper abstraction usage, good code dependency graphs, error management (inputs, connection errors, etc), code comments when needed, public API documentation (javadoc, pydoc, scaladoc, etc).

Unit tests will be evaluated: how you built them, what patterns you followed, etc. If you are not very familiar with them, this is the time to show you are able to learn and apply something new :)

#eb/topics/hiring