Backend Challenge

You work in the Department of Transportation in your city. Among other responsibilities, you need to research ways to improve transportation in your city. The best way to do that is through data and trying to get insights from it. Some time ago, you implemented some processes in your city, together with public transportation drivers, to make sure you have access to data about passenger pickups and drop-offs in your city. That is raw data, and now you want to build a simple tool that allows you and other colleagues in your department to get some statistics easily from this data.

The simplest way to do it and share it is automating a process that fetches that raw data every month and loads it into a SQL database.

You are going to keep the backend that gives you the data, based on the parameters, separated from the frontend. You'll do this because at some point, you will open your backend API so the city's citizens can build their own frontends.

As an example, we will use some data provided by NYC regarding taxi trips during certain days of January 2018. This data can be downloaded from:

https://easyupload.io/m/xg063v

- Green taxis -**green_tripdata_2018-01_01-15.csv**
- Yellow taxis -**yellow_tripdata_2018-01_01-15.csv**
- Zones list -**zones.csv**

(This data is just a simplification from the original data, that can be found at https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page . For this exercise purposes we suggest to use the simplification version)

Automating the data load

Your mission, as mentioned, above is to automate the loading of this data into a SQL database that you choose. Whatever this automation is, write it into code that we can easily run to load the data ourselves when we review the exercise. ++Please be aware++ that this is an important part of the exercise and we flavor fast and elegant solutions. There are three datasets: green taxis, yellow taxis, and zones. The green and yellow taxis datasets contain the trips data. The only information you need from those datasets is the pickup and drop-off datetimes and the pickup and drop-off location ids. You should discard/ignore the rest of the information contained on those datasets. For this exercise purposes, load both types of taxis into the same table. We don't want to see the differentiation of the two types once loaded into the database.

The zones dataset contains the id and names of the different zones of the city. These ids are the same ones that are referenced from the taxi trips pickup and drop-off locations. Keep in mind this relationship between trips and locations when you create your SQL tables.

Building the backend

Your backend will communicate with the database in which you have loaded the datasets. You can choose whatever programming language and/or framework you desire. We suggest you to use the **Java Spring Boot**.

Your backend will be exposed through a HTTP server and will contain 3 HTTP endpoints that a frontend could use to retrieve the data.

/top-zones

This endpoint will return a list of the first 5 zones order by number of total pickups or the number of total drop-offs. This means that this endpoint will accept 1 parameter (either GET or POST): • order : values can be "dropoffs" or "pickups"

An example of the response:

```
{
    "top_zones": [
        {
            "zone": "Midtown East",
            "pu_total": 435,
            "do_total": 321
        },
        {
            "zone": "Jackson Heights",
            "pu_total": 324,
            "do_total": 456
        },
  ....
    ]
}
```

/zone-trips

This endpoint will return the sum of the pickups and drop-offs in just one zone and one date. This means that this endpoint will accept 2 parameters (either GET or POST):

-
- zone : value must be the zone id of any of the available zones date : value must be a date

An example of the response:

```
{
    "zone": "Bushwick North",
    "date": "2018-01-12",
    "pu": 3245,
    "do": 1345
}
```

/list-yellow

This endpoint will return data from the yellow trip file with pagination + filtering and multiple sort This means that this endpoint will accept several parameters (either GET or POST):

- ## Technical requirements
  - Java Spring boot
  - Unit Tests
  - Performance tests
  - Open Api 3
  - ORM
    - Migration Strategies
  - Clear curl instructions
  - Docker
    - Injestion of initial data
  - Centralized logging using containers

What we're looking for:

- clean project setup and well documented elegant/fast way to upload data ability
- to dive into a new topic, extract the important points and then code it up.
- make it snappy
  document your approach, your decisions and your general notes Fulfill
- the exercise expectations .
- OpenAPI on REST endpoints. Write clear and well-structured code. You won't work
- alone. It should be easy to understand and modify your code. Spend no more than 4
- hours . We estimate that this exercise could be done in 4 hours. You already have
- things to do in your own time and we don't want this exercise to alter too much
  your life. • Send working instructions . Do you imagine you buy a product that
- you need to install by yourself and when you follow the instructions provided, it does not work? It would be frustrating. • Use technologies you know . This is not time to test something new unless you are really sure about it. Use technologies you already know and have experience with.

- **We expect a README file in the repository**. This README must explain very clearly the instructions for us to start the application without any type of debugging or complication. The instructions must work out of the box. **This is one of the most important part of the problem**. You need to think always about who is going to use it and try to reduce frustration.
- **Share the repo** /Github/Gitlab/Bitbucket)
  - Any code you write must be committed to a Git repository. Please, start committing since the beginning. Don't wait to have the entire exercise and just send one commit with all the code. We would like to see your progress.

We don't expect you to:

- Be a craftsman . We don't expect to see the most optimized code. Be pragmatic with the time you have and the things you need to do. Implement a production
- service . Forget about authentication, users, SSL, ...
  Just focus on the functionality described above.
-

Implement the most performant solution .

Have fun!