SOLACTIVE CODE CHALLENGE (v1.04)




Summary

Solactive, as one of the leading providers of financial indexes, consumes and uses real-time trading prices of tens of thousands of financial instruments from more than 100 exchanges over the world.

In order to ensure integrity of our index calculation and proper input data, our operations team needs a restful API to monitor the incoming prices. The main use case for that API is to provide real-time price statistics from the last 60 seconds (sliding time interval).

There will be three APIs:

- The first one is called every time we receive a tick. It is also the sole input of this rest API.
- The second one returns the statistics based on the ticks of all instruments of the last 60 seconds (sliding time interval)
- The third one returns the statistics based on the ticks of one instrument of the last 60 seconds (sliding time interval).

All APIs might be called in parallel. Notably; POST "/ticks" might be continuously called before the previous /ticks where finished.

Specifications

`POST /ticks`

Every time a new tick arrives, this endpoint will be called. Body:

```
{
"instrument": "IBM.N",
"price": 143.82,
"timestamp": 1478192204000
}
```

where:

- instrument - a financial instrument identifier (string; list of instruments is not known to our service in advance so we add them dynamically)
- price - current trade price of a financial instrument (double)
- timestamp - tick timestamp in milliseconds (long; this is not current timestamp)

Returns: Empty body with either 201 or 204:

- 201 - in case of success
- 204 - if tick is older than 60 seconds

`GET /statistics`

This is the endpoint with aggregated statistics for all ticks across all instruments, this endpoint has to execute in constant time and memory (O(1)).

It returns the following statistics based on the ticks which happened in the last 60 seconds (sliding time interval).
Returns:

```
{
"avg": 100,
"max": 200,
"min": 50,
"count": 10
}
```

where:

- avg is a double specifying the average amount of all tick prices in the last 60 seconds
- max is a double specifying single highest tick price in the last 60 seconds
- min is a double specifying single lowest tick price in the last 60 seconds
- count is a long specifying the total number of ticks happened in the last 60 seconds

`GET /statistics/{instrument_identifier}`

This is the endpoint with statistics for a given instrument.
It returns the statistic based on the ticks with a given instrument identifier happened in the last 60 seconds (sliding time interval). The response is the same as for the previous endpoint but with instrument specific statistics.

Requirements & Comments

- We kindly ask you **not to use standard libraries for statistic** calculation or aggregation
- Please implement an in-memory solution. Don't use any kind of database

- Please pay attention to the code quality of your solution, as you would do in your daily work
- Please provide test coverage as you deem appropriate and necessary – as you would do in your daily work

- GET /statistics should execute in constant time
- GET /statistics/{instrument_identifier} should execute in constant time

- The APIs have to be **safe against concurrent requests**
- The APIs should be able to deal with time discrepancy, which means, at any point of time, we could receive a tick which have a timestamp in the past
- Please add the following dependencies to your POM (to facilitate our evaluation)

```
<dependency>
   <groupId>net.bull.javamelody</groupId>
   <artifactId>javamelody-spring-boot-starter</artifactId>
   <version>1.81.0</version>
</dependency>
<dependency>
   <groupId>com.thoughtworks.xstream</groupId>
   <artifactId>xstream</artifactId>
   <version>1.4.10</version>
</dependency>
<dependency>
   <groupId>org.jrobin</groupId>
   <artifactId>jrobin</artifactId>
   <version>1.5.9</version>
</dependency>
```

Shipping

- Please provide a README file with following points:
  - How to run the project (preferably with maven Spring Boot plugin or alike, or as an executable jar)
  - Which assumptions you made while developing
  - What would you improve if you had more time
  - And, whether you liked the challenge or not 😊
- Please share the source code and the readme with us via **GitLab.com**

We hope you enjoy the small challenge - we're very curious to see your solution work.

Your Solactive Team.