**TASK SOLACTIVE**

| **REVISION HISTORY** | | | |
| --- | --- | --- | --- |
| **Version** | **Description** | **Changed By** | **Date** |
| 1.0 | Initial Document | Ramazan Erecir | 12.03.2020 |
|  |  |  |  |
|  |  |  |  |

# ABSTRACT

This document provides solution for the Tick Statistics APIs, steps to build project source code (See Build Project Section) and steps to run the application and includes information about development assumptions and further improvements.

# BUILD PROJECT

Maven Spring Boot plugin and Java 8 is required to build source code. Maven pom.xml is provided within the project.

Project source code can be downloaded from Github repository: <https://github.com/ramazanerecir/TickStatisticsService.git>

Please follow the below steps to build the project after downloading source code.

1. Go to project\TaskSolactive directory.
2. Run ‘mvn clean install’ to build project
3. Jar file will be created under project\TaskSolactive\target directory.

# TEST

Junit Test cases are provided within the project. There are 7 different test cases which are running as unit testing. Test results can be seen (as below) while building project.

*[INFO] Tests run: 7, Failures: 0, Errors: 0, Skipped: 0*

1. contextLoads test checks whether application is successfully initialized and running.

*SOLACTIVE TASK Service is Running*

1. tick post test checks whether tick is created.

*testTick /ticks response status (expected 201 CREATED) : 201*

1. tick which has a past timestamp but in the sliding time interval post test checks whether tick is created.

*testTickPast /ticks response status (expected 201 CREATED) : 201*

1. tick which is out of sliding time interval post test checks whether tick post request returns no content result.

*testTickNoContent /ticks response status (expected 204 NO\_CONTENT) : 204*

1. statitistics/instrument test checks whether inserted tick created a statistics.

*testStatisticsInstrument /statistics/IBM.NE response : {"avg":143.82,"max":143.82,"min":143.82,"maxDrawdown":0.0,"volatility":0.0,"quantile":143.82,"twap":0.299625,"twap2":0.01912500000000002,"count":1}*

1. statitistics/instrument test checks whether inserted tick having past timestamp created a statistics.

*testStatisticsInstrument /statistics/GOOGL response : {"avg":1092.09,"max":1092.09,"min":1092.09,"maxDrawdown":0.0,"volatility":0.0,"quantile":1092.09,"twap":18.2561045,"twap2":1.1652832659574481,"count":1}*

1. statistics test returns aggregated statitstics

*testStatistics /statistics response : {"avg":617.9549999999999,"max":1092.09,"min":143.82,"maxDrawdown":948.27,"volatility":474.13499999999993,"quantile":143.82,"twap":8.88473575,"twap2":1.2523041611685284,"count":2}*

# RUNNING APPLICATION

Application can be run as below command.

**java -jar target/TaskSolactive-0.0.1-SNAPSHOT.jar**

Application parameters: Only one parameter can be given while running application. It is configuration file path. Default value of the configuration file path is current directory. Configuration file directory should include below files.

* taskSolactive.cfg file : Configuration file includes application parameters
* log4j.properties : Log file configuration

Application is running with default parameters, it can be reached from <http://localhost:8080/>

After starting application following services will be available.

1. POST /ticks

Accepts the incoming ticks if its in sliding time interval and returns Created, otherwise returns No Content.

1. GET /statistics

Returns the aggregated statistics.

1. GET /statistics/{instrument\_identifier}

Returns the statistics for a given instrument.

# CONFIGURATION FILE

Configuration parameters can be modified in taskSolactive.cfg file which can be found in the project directory.

**Parameters**

tick.sliding.time.interval : Sliding time interval period parameter in milliseconds

tick.cleaner.interval : Delay time interval in milliseconds between each cleaner process for out of sliding time interval ticks

tick.process.pooler.size : Number of threads which processes tick messages from Message blocking queue

tick.calculation.pooler.size : Number of threads which calculates instrument statistics which are taken from Instrument blocking queue

calculation.aggregated.timer : Timer Thread period in milliseconds to trigger aggregated statistics calculation

statistics.lambda : Decay factor parameter to determine tick weights in twap2 statistics calculation

statistics.percentile : Lower percentile parameter for quantile calculation

statistics.scale : Statistics scale parameter. Not used.

statistics.price.scale : Scale parameter for price related statistics. Not used.

# LOG FILE

Application logs can be managed by the log4j file which can be found in the project directory.

# DEVELOPMENT ASSUMPTIONS

1. **Ticks post workflow**: When a new tick is posted to application; if it is not in the sliding time interval, it is discarded. If it is in the sliding time interval, first it is put in TickProcess BlockingQueue. This Queue is listened by a thread pool, size of the thread is determined by the configuration parameter “tick.process.pooler.size”. One of these threads takes tick message and then try acquire a Lock for the tick instrument. After acquiring lock, tick is added to concurrent hashmap which stores ticks list with instrument key. Then, tick instrument is put into InstrumentCalculation BlockingQueue for statistics calculation. This Queue is also listened by a thread pool, size of the thread is determined by the configuration parameter “tick.calculation.pooler.size”. One of these threads takes the instrument from queue and tries to acquire Instrument lock. After acquiring instrument lock, it calculates statistics for this instrument and put Instrument-Statistics concurrent hashmap.
2. **Aggregated statistics calculation workflow**: Timer thread is run periodically and calculates aggregated statistics. First, all the ticks are collected from instrument-tick list map. Other processes wait for this job. After collecting ticks, other processes continues their job. After collecting ticks, aggregated statistics is calculated and put into Instrument-Statistics hashmap wih a specific instrument name for aggregation.
3. **Tick Cleaner workflow**: One thread is run to remove the ticks from Instrument-Tick list map. This thread checks each instrument in this map. First it acquires instrument lock. Then updates the map with filtering ticks which are out of sliding time interval. If instrument still has ticks, it puts instrument into InstrumentCalculation BlockingQueue for statistics recalculation. If all the instruments in the map are checked, it waits for a while according to configuration parameter “tick.cleaner.interval”. Then it repeats same process.
4. **Statistics get workflow**: Statistics are stored in a concurrent map with instrument key. Aggregated statistics is also stored in the same map with a specific name. So, get requests will be returned in constant time.
5. If incoming tick is not in the sliding time interval, it is discarded at the beginning and it is not taken into account in statistics calculation.
6. While calculating statistics and removing tick and its effect on statistics, timestamp is taken from incoming tick message. The timestamp when tick message is received is not taken into account, it is used while checking whether tick is in sliding time interval.
7. Aggregated statistics is calculated by a timer thread whose period can be set with a configuration parameter named as “calculation.aggregated.timer”. So, result for the request of aggregated statistics might be a past data according to this parameter.
8. While calculating aggregated statistics, the currency of the instruments are discarded. All instruments are considered to be on the same currency.
9. Aggregated statistics is calculated from all ticks similar to instrument statistics calculation in which statistics are calculated from the ticks belonging to corresponding instrument. Note: There might be a confusion about aggregated statistics calculation.

# FURTHER IMPROVEMENTS

1. Security control checks can be done while accessing APIs.
2. A warning mechanism or action options can be created/set with a threshold parameter in case there is a drastic increase/decrease on one after another ticks.
3. Statistics values should be scaled.
4. Triggered action can be put in Instrument Calculation Blocking Queue to identify which action caused statistics calculation.
5. The latest statistics calculation timestamp and the statistics get request timestamp can be stored in Statistics object for reporting purpose.
6. Ticks which are not in the sliding time interval can be logged for reporting purpose.
7. Acquiring lock on instrument by the threads can be decreased.
8. The aggregated calculation time interval can be decreased.
9. Complex test cases can be added.

# RELEASED FILES

**Project:** Project includes the project sources codes and Maven pom file.

**Document:** Documentation of the task is provided.