

Temperoget

Solution Doument v1.0

23 Jun 2021

# Objective

This document gives a technical solution for temperature collection and aggregation. The solution caters to the following requirements –

1. Sensor sends the continuous stream of temperature data to API.
2. If internet connection is broken, the temperature data is stored locally and will be synchronized later.
3. API will return the temperature aggregated data (hourly and daily) to sensor based on request.
4. Performance of aggregated data is critical, so it should return the data faster.

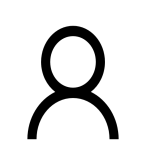
# Assumption

1. Scope of this solution is API only. Sensor design to send the data to API and displaying aggregated data as chart is not in scope.
2. Sensor has to send the temperature data in agreed format, which will be described in this document.
3. Aggregated temperature data is calculated based on –
   1. Last 24 hours data
   2. Last 7 days data
4. For continuous data streaming between sensor and API, a queue is required. This solution document is assuming that a Kafka message queue is already defined. This solution with use a separate topic ‘**QardioTopic**’ to pass the messages.
5. For persistence data by API, a database is required. This solution assumes that one MySQL database exists with name **QardioDB**. This document will give description of the table which need to be created to store the data.
6. Archiving of temperature data is not in scope of this document, ideally that would be needed on bigger solution to keep the data precise and to improve the performance.
7. To make the solution scalable, it is assumed that that there could be multiple terminals which will be sending the temperature data to central API through Kafka. Each terminal is identified by unique terminal id. Management of terminal ids are not in scope of this solution, however, how temperature data would be stored and aggregated data are retrieved using terminal id is shown here.

# Repository

[seemadubey-in/QardioTemperoget (github.com)](https://github.com/seemadubey-in/QardioTemperoget)

# Architecture



Sensor

Messaging queueu

REST API

Database

User will interact with sensor which will send the temperature data stream to messaging queue. This queue will help in storing the data from sensor even when connection between queue and API is lost. As soon as the connection is, the temperature data from received from sensor will be passed to API to get processed.

API will store all the temperature data received from queue into database. This data will be used later for aggregation.

In case of data aggregation request, sensor will directly call API’s end point. REST API will use native query to fetch the aggregated data quickly from database. Once aggregated data on hourly and daily basis is retrieved, API will pass the data back to sensor to be displayed as chart.

# Database Structure

Database: temperature\_data

|  |  |  |
| --- | --- | --- |
| Name | Type | Remarks |
| Reading\_id | Integer | Primary key, auto increment |
| Terminal\_id | Integer |  |
| Temperature | Big decimal |  |
| Date\_capture | Timestamp |  |

# Message Structure

Message: Temperature

|  |  |  |
| --- | --- | --- |
| Name | Type | Example |
| TerminalId | Integer | 123 |
| Temp | BigDecimal | 98.6 |
| dateTemp | String | 23-Jun-2021 12:45:45 |

It’s very important to pass the message in this format to queue from sensor.

# Deployment Guide

This deployment is divided into 4 parts –

1. Database creation
2. Kakfa topic creation
3. Building API
4. Starting application
5. **Database creation**

Assuming MySQL is running on local instance for this exercise. One database qardiodb must be created as local database. Let’s assume it uses user ***root*** and password ***qardio123***.

Please execute this SQL query to create temperature\_data table –

CREATE TABLE qardiodb.temperature\_data (

reading\_id int primary key auto\_increment,

terminal\_id int,

temperature decimal (6,2).

date\_capture timestamp default current\_timestamp

);

1. **Kafka Topic creation**

Assuming zookeeper and kafka server are running on local machine with default ports – zookeeper on port 2181 and kafka server is using port 9092. Please go to Kafka root folder to execute the commands –

Create a new topic QardioTopic with one partition

.\bin\windows\kafka-topics.bat --create --zookeeper localhost:2181 –replication-factor 1 –partitions 1 --topic QardioTopic

1. **Building API**

Download the repository from [seemadubey-in/QardioTemperoget (github.com)](https://github.com/seemadubey-in/QardioTemperoget). Go inside TempData folder using command prompt and execute install command

MVN CLEAN PACKAGE

1. **Starting application**

Step 3 would have created jar file in target folder. Please go inside target folder using command prompt and execute command to start the application –

Java –jar TempData-0.1.jar

This will start the application on port 8080.

# Execution Guide

This API has two end points –

1. Save temperature data
2. Get aggregated temperature data

Here is how the end point can be invoked –

1. **Save Temperature data**

Saving temperature data works through Kafka queue. As soon as the message (temperature data) comes into queue, it will be consumed by API through Kafka listener. To simulate message arrival in queue, we can use Kafka producer. Go to Kafka root folder using command prompt, and give this command to execute Kafka producer –

.\bin\windows\kafka-console-producer.bat --broker-list localhost:9092 --topic QardioTopic

Once Kafka producer is ready, we will be able to push messages into QardioTopic. Here is how a temperature message needs to be pushed –

> {"terminalId": 134,"temp": 99.7,"dateTemp": "24-Jun-2021 05:45:21"}

Please note the message format. The same temperature data will be stored on databases as soon as the message arrives on the queue.

For testing purpose, I have given the endpoint accessible for Postman as well. To execute the save end point, please use Postman to hit URL <http://localhost:8080/createTempData> . This needs the body as well containing temperature information. Here is the body which you need to pass –

{

    "terminalId": 134,

    "temp": 98.5,

    "dateTemp": "24-Jun-2021 10:20:21"

}

1. **Get aggregated temperature data**

This end point can be directly invoked by Postman. While saving temperature data, we have assumed that temperature data must be coming from terminal. Now this end point would get the aggregated data for each terminal. This will be a GET method via Postman including terminal id.

GET

http://localhost:8080/getaggregated/132

Here is the sample output which you would get –

{

    "terminalId": 132,

    "hourlyTempList": [

        {

            "dateCaptured": "2021-06-23",

            "hourCaptured": 10,

            "hourlyTemprature": 98.50

        },

        {

            "dateCaptured": "2021-06-23",

            "hourCaptured": 11,

            "hourlyTemprature": 98.20

        }

    ],

    "dailyTempList": [

        {

            "dailydateCaptured": "2021-06-23",

            "dailyTemprature": 98.50

        }

    ]

}