My thoughts on the project:

1. I use Spring Boot framework to build the application. The reason I use it because it helps to build the application faster. It takes care of database connection, database mapping ORM, and APIs. It‘s also a light weight framework and support to connect to other frameworks.
2. If I have more time and have a proper environment, I would build Kafka and MQTT between my application and Meraki devices. MQTT would listen to the devices and send their data to Kafka. Kafka can hold more than 10,000 data points. My application would subscribe to Kafka and get one record at a time from Kafka to process.
3. Since I don’t have the #1 set up, I send 1 record at a time using postman.
4. I use Post method on localhost:8080/datastream to send the devices’ data to my application. There, I store them in a map for 1 minute (based on the given timestamp). However, I believe in real time, the timestamp from devices would match with the system timestamp. This would give me a chance to count the elapse time so that I can store the last record to database. Right now, I don’t know which one would be the last record, so I had to fake 1 dummy record 1 minute over to complete the process.
5. The map has key is device\_id, and value is the list of min, max, sum, and count. I compare and calculate each time the record is read to get min, max, sum, and count. Sum and count are used to calculate average when reaching 1 minute. Doing this way will save space in case we have a million of records. If values are stored in the list instead, it would take a lot of space if we have a million of records, and at the end when calling Collections.min() and Collections.max(), it still has to traverse through the whole list to compare and find the right values.
6. Once a minute is up, I store all the calculated data to a list, and persist them all to database at once using jdbcTemplate.batchUpdate(). This way, I only connect to db 1 time and could store all the objects to it. The performance is a lot better.
7. Once a record read in meet 1 minute, after saving all records in the map to database, I make a copy of the map in case the next record has a timestamp falls into the previous minute (e.g. 1611741659 goes after 1611741660). I assume the time difference is seconds, not minute. In this case, I use the copy map to compare and calculate min, max, sum, and count. I then update database right away. The original map got clear after records saved to db and ready to store the new record that not falls into the previous minute. The copy map will be clear automatically when I copy a map to it. The drawback of this approach is that it connects to db each time of update. If I have a large number of records need to update, then it will reduce the performance a lot.
8. For retrieving all the records, I use the built in findAll() method from JpaRepository to get the records. I use get method on localhost:8080/datastream endpoint for records retrieval purpose.

