Activity 2: Unit Tests and Test Driven Development

Context:

Imagine a company called *SmartNet Solutions* that specializes in implementing IoT (Internet of Things) sensor networks for smart cities. These sensors continuously collect data on various parameters such as air quality, noise levels, traffic, and other environmental factors.
Each sensor sends measurement data at small intervals (10 times per second). The measurement data contains the timestamp of the lecture, and a number M representing the lecture. M is a double, $-10^{10} \leq M \leq 10^{10}$. The company has accumulated millions of data records. Due to the large volume of data and the need for efficient analysis, SmartNet Solutions wants to identify patterns and optimize their sensor network.
One of the problems they want to solve is *the efficient identification of the continuous period of time with the highest accumulated air quality, the period of time with the highest accumulated noise level, and the period of time with the highest accumulated ultraviolet radiation.* The *accumulated* value of some paramter over a period of time is *the sum of the values registered over that period.*

1 The problem:

SmartNet Solutions has hired you as a software engineer to develop an API that

- allows the registration of the lectures of noise level, ultraviolet radiation and air quality. The lectures are represented by the sensors as a `double` number (as described above). The system should be able to handle 1000 sensors in total (among noise, ultra-violet radiation and air quality sensors), and each of the sensors will produce approximately 10 reads per second.
  - In this regard, the system should be able to handle *two* kind of registrations: *single-value* registration (in which one value is registered) (this is the one that can happen 10 times per second) and *multi-value* registration (in which up to several million of values will be registered at once).
- answers the following queries:
  - *which is the highest accumulated amount of noise registered in a contiguous period of time, and in which period occurs*,
  - *which is the highest accumulated amount of ultraviolet radiation registered in a contiguous period of time, and in which period occurs*,
  - *which is the highest accumulated air quality registered in a contiguous period of time, and in which period occurs*,

  Given that these queries will be performed by hundreds of devices, the system should be able to solve efficiently these queries, even when it receives up to one thousand queries per second.

2 The expected API:

The API that SmartNet Solutions expects from you is the following:

- `register_one(timestamp, type_of_sensor, read)`
- `register_many(LIST_OF(timestamp, type_of_sensor, read))`
- `highest_accumulated(type_of_sensor)`

Where

- `(timestamp, read)` : *read* is the value observed by the sensor at the time indicated by *timestamp*.
- `type_of_sensor` : either `AIRQUALITY` or `ULTRAVIOLETRADIATION` or `TRAFFIC`.
- The `register_*` functions returns success if the value was properly registered and error otherwise.
- The `highest_accumulated` function returns a string representing a json object with the following format:

```
{
  highest_accumulated_value: <double indicating the highest
                              accumulated value>,
  from: <timestamp corresponding to the begin of the interval
         in which the highest accumulated value was observed>,
  to: <timestamp corresponding to the end of the interval in
       which the highest accumulated value was observed>
}
```

  In the case of queries of sensors with no data, return -1 in all the fields.

3 Examples:

```
REQUEST: register\_one(1, AIRQUALITY, 100.0)
RESPONSE: OK
REQUEST: register\_one(2, AIRQUALITY, 50.0)
RESPONSE: OK
REQUEST: register\_one(3, AIRQUALITY, -100.0)
RESPONSE: OK
REQUEST: register\_one(4, AIRQUALITY, 110.0)
RESPONSE: OK
REQUEST: register\_one(5, AIRQUALITY, -200.0)
RESPONSE: OK
REQUEST: highest\_accumulated(ULTRAVIOLETRADIATION)
RESPONSE:
{
  highest_accumulated_value: -1.0,
  from: -1,
```

```
  to: -1
}
REQUEST: highest\_accumulated(AIRQUALITY)
RESPONSE:
{
  highest_accumulated_value: 160.0,
  from: 1,
  to: 4
}
```

4 Expected performance and code quality:

SmartNet Solutions expects your solution to have very good code coverage. The correctness of your system should be extensively tested.
Although the HTTP endpoints are encouraged to be implemented in Python in a similar way to the previous task, the functionality is expected to be implemented in C++ and tested using GTest. Thus, you are expected to define the HTTP endpoints in python. Your python program should instantiate your C++ program and rely on it for the implementations of the endpoints.

Steps

1. (learn) Creating python bindings for a C++ object (the topic will be discussed in class. We will use `boost::python`, but any other approach is also accepted).
2. (learn) Instantiate a C++ object and call its methods from python.
3. Write the HTTP API in Python, and write the corresponding tests using pytest. (this is: the unit tests of the front end of the system)
4. Write a dummy solution in C++, in which you return incorrect values.
5. Connect the HTTP API (in Python) with the dummy C++ solution. This is: call the C++ methods from Python, and obtain their answers.
6. Write unit tests in GTest to test the correctness of the methods written in C++. (this is: write the unit tests of the backend of the system)
7. Write *integration* tests that confirms that the system works end-to-end. (the topic will be discussed further in class)