**Introduction**

The goal of this home-project is to create a client-server system that will perform some

basic functions over some data. The server will also expose a RESTFul API for some

additional operations.

**Details**

**Client**

The client will be configured to connect to the server. It will then send a random-sized

array (the size of the array should be between 10-20, randomly chosen each time) of

random integers (randomly picked on each run as well). It will then wait for the server’s

response (which is another array of integers, see next section). Next step will be to check if

the array in the server’s response is sorted and finally it will close the connection.

**Server**

The server maintains an internal array of integers. This array must stay sorted in increasing

order, at any time. The array will be filled by numbers coming from the client(s), or the

“add\_value” endpoint (see below) and it is empty when the server first fires-up.

The server should be always-on and accepting new Client connections. When a client

connects, the server will wait for the client’s data and when it gets the data it will insert

them to its internal array while taking care than the internal array stays sorted. It will then

return this internal array as a response to the client. The connection will be closed by the

client.

The server should be able to deal with many simultaneous client connections.

The server should be packed in a docker container.

**REST API**

Besides accepting Socket connections from the clients, the server will expose a RESTFul API

with the following endpoints:

● /clean\_array: a DELETE request at this endpoint will result in removing all integers

for the server’s array.

● /get\_array: a GET request at this endpoint will respond with the server’s array

values.

● /add\_value: a POST request at this endpoint will add an integer to the server’s

array.