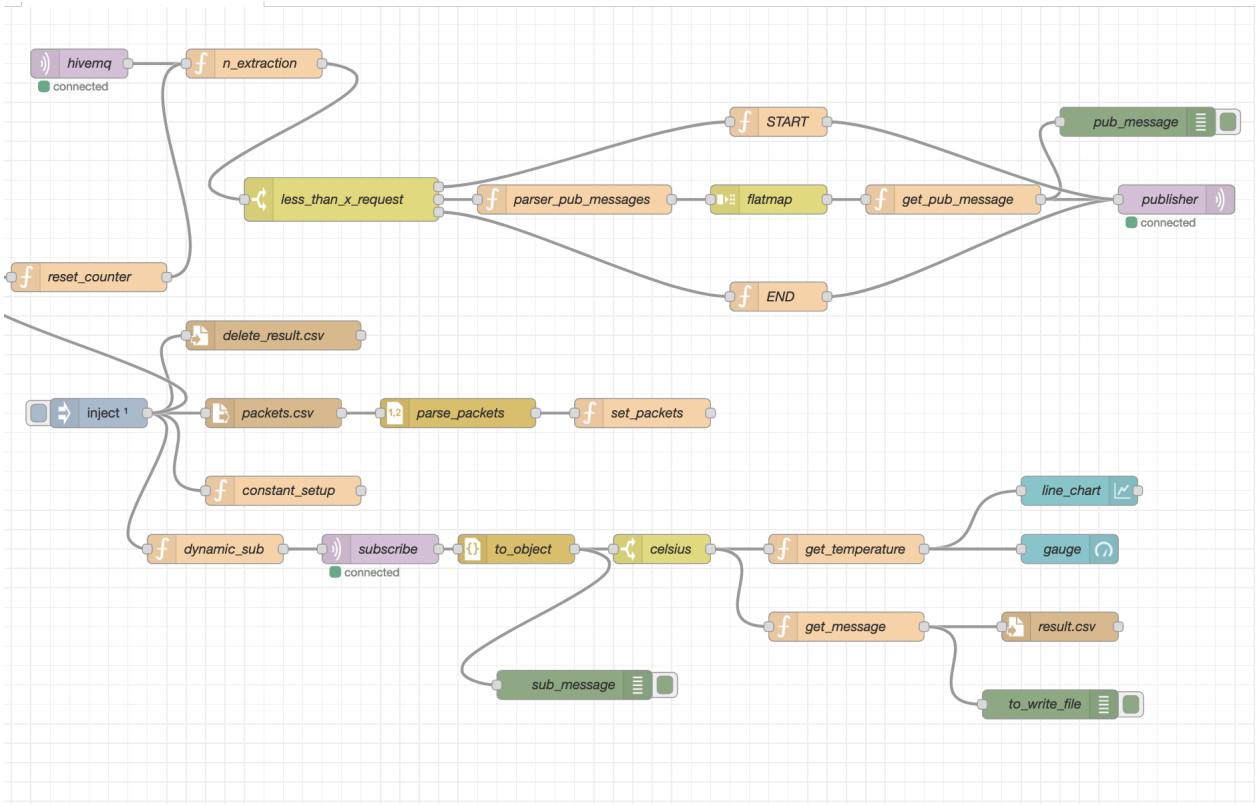
Second Challenge IOT Report

Lorenzo Campana: 10605775
Irio Castrignano: 10656379

April 25, 2023

1 Introduction



Initialization

Some Initialization is performed at the start of execution. Constant and *packets.csv* file are imported and set as flow variables.

Extract the ID

Hivemq node is the MQTT client that receives id random generated requests, which are used to first extract the n identifier based on the rules defined on the requirement file. Along with the extracted n, a counter updated in the context environment is passed through the *less_than_x* switch. For the first request, so when the counter is one, when send a mqtt publish request with a payload containing *"START"*. If $0 < counter < max\_req == 100$, then the flow moves along the *parser_pub_messages*, which is in charge of finding the packet having the specified id and producing the corresponding publish messages to send later to the MQTT publisher. The code of the parser can be seen here below, which first fetch the flow variable initialized at the start of the process by reading the *packets.csv*

file. First we filter all the packets having the specified *id* and starting with *Publish Message*, then we leverage on two regex expression to transform topics and messages strings into array, that are flattened into array of objects, each of them representing a publish message request to send to the broker. These array of messages is then handled as a stream of objects and published one at a time in the MQTT client publisher.

```
const {n, id} = msg
const packets = flow.get("packets") ?? []
const person_code = flow.get('person_code')
const timestamp = new Date().toISOString()
const topic = `/polimi/iot2023/challenge2/${person_code}`

const messages = packets.filter(r => r['No.'] == n && r['Info'].startsWith('Publish
    .map(r => {
        var { ['No.']: No, ['Info']: info, ['Message']: message, ...rest } = r;
        const regex_infos = /(?<=\[)[^\[\]]*(?=\])/g;
        const topics = info ? info.match(regex_infos) : []
        const regex_messages = /{[^}]+}/g;
        const messages = message ? message.match(regex_messages) : []
        return { ...rest, topics, messages };
    })
    .flatMap(r => {
        const res = [];
        const topic_len = r.topics ? r.topics.length : 0;
        for (let i = 0; i < topic_len; i++) {
            const {topics, messages, ...rest} = r
            res.push({
                ...rest,
                topic: topics[i],
                message: messages[i]
                })
        }
        return res;
    })
    .map(p => {
        const {message} = p
        const payload = {
        'CURRENT_TIMESTAMP': timestamp,
        'PREVIOUS_ID': id,
        'MQTT_PUBLISH_PAYLOAD': message
        }
        return {topic, payload};
    })
return {payload: messages};
```

parser_pub_messages

Subscription

The MQTT subscriber is dynamically subscribed based on the person_code set in the initialization phase. Then for each message received, it is first translated into a javascript object, and passed through the flow if contains a celsius temperature.

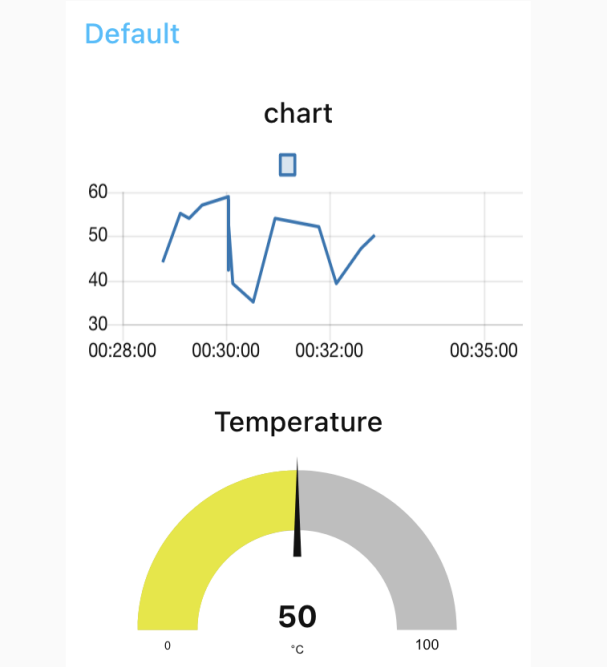


Chart and gauge of the temperatures collected

Nodes

n_extraction

Logic for extractions of n, which also contains some control logic for the reset of the context variable *counter*, which keeps tracks of the number of requests handled.

```
const { reset_counter = false } = msg;
if (reset_counter) {
    context.set('counter', 0);
    return;
}
const {payload} = msg;
const divider = flow.get('divider');
const person_code = flow.get('person_code');
var {id} = payload
var counter = parseInt(context.get('counter') ?? 0);
context.set('counter', ++counter)
if (!id || !divider) return;
const n = (parseInt(id) + parseInt(person_code.slice(-4))) % divider
return {n, id, counter};
```

constant_setup

```
const {payload: packets} = msg;
flow.set('person_code', "10605775");
flow.set('divider', 7711);
return msg;
```

dynamic_sub

```
const person_code = flow.get('person_code');
const topic = `/polimi/iot2023/challenge2/${person_code}`
const action = 'subscribe';
return {action, topic};
```