IoT Challenge 2

Fatih Temiz(10901682)
Ecem İzde Kaya(10902759)

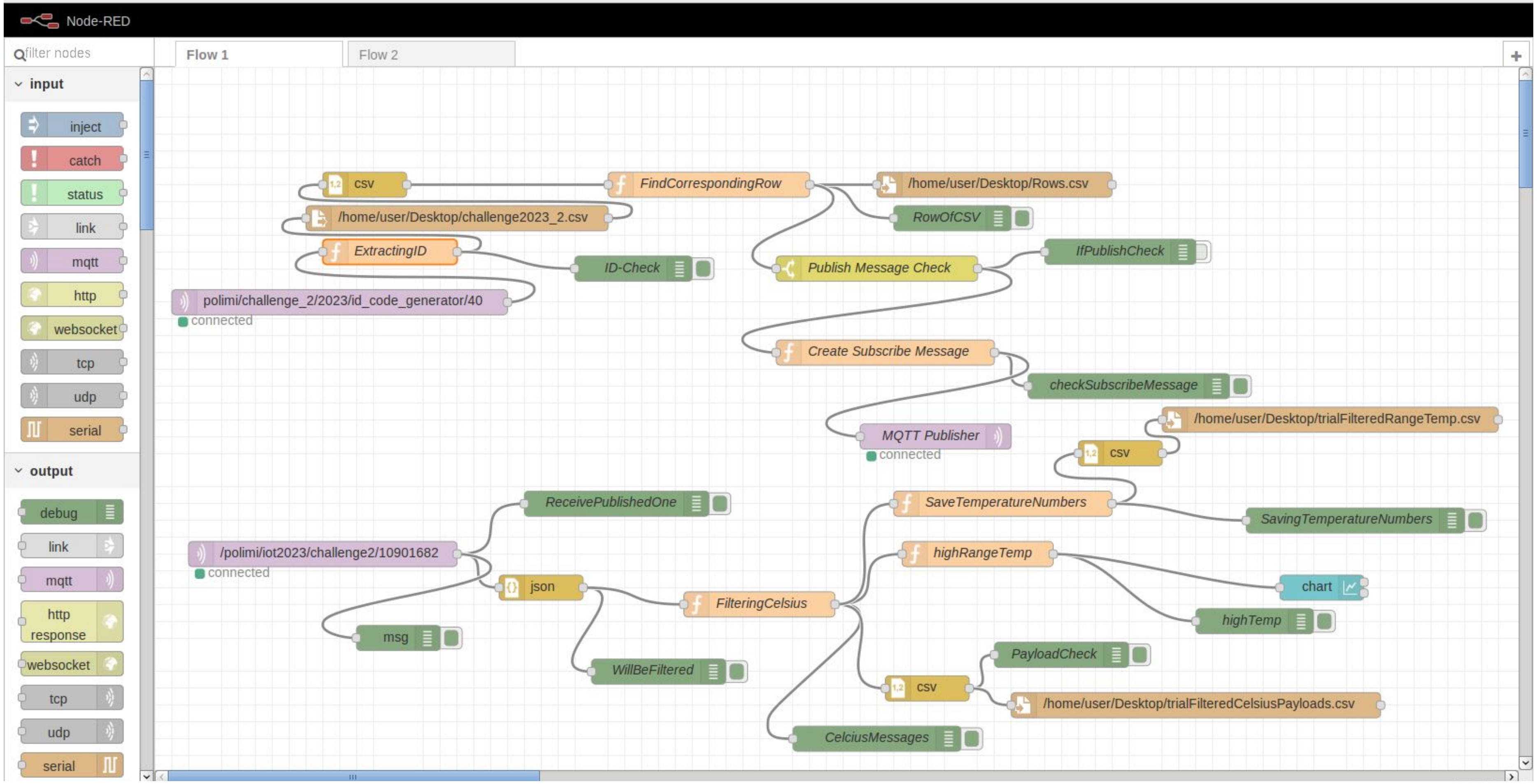
Introducing

We connect to the public HiveMQ Broker ('broker.hivemq.com' with port 1883) and subscribe to the topic “polimi/challenge_2/2023/id_code_generator”.

We only select messages which are automatically sent by a client every 5 seconds with the following structure: {"timestamp":"yyyy..","id":"6292"} with the frame number equal to the obtained number by extracting the ID. We consider the last 4 digits of the team leader’s ID (10901682) and take the remainder (%) of division by 7711 with the following calculation: n = (1682+ID) % 7711.

After that we take corresponding row using ID from .csv file and check that messages contains publish message using switch and if it contains we create payload using timestamp, extracted id and payload of the publish messages {"timestamp":"CURRENT_TIMESTAMP","id":"PREVIOUS_ID","payload":"MQTT_PUBLISH_PAYLOAD"} and we send a publish message to the same broker to the topic /polimi/iot2023/challenge2/PERSON_CODE.
We also subscribe on same topic and get messages that we publish and converted into json object, after that we filtered messages using FilteringCelsius function that explained in following slides. After that, we get upperbound of the temperature and send it to the chart. All functions explained in detail in following slides.

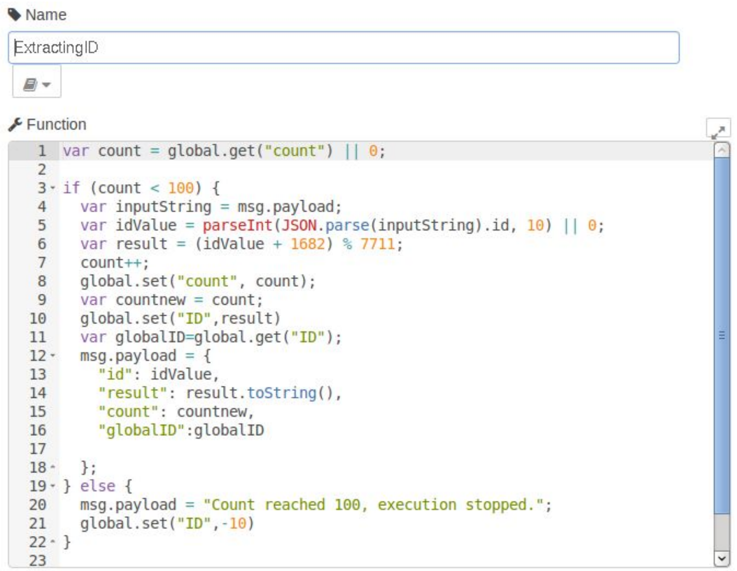
Node-Red Flow



Extracting ID and using it for finding corresponding row

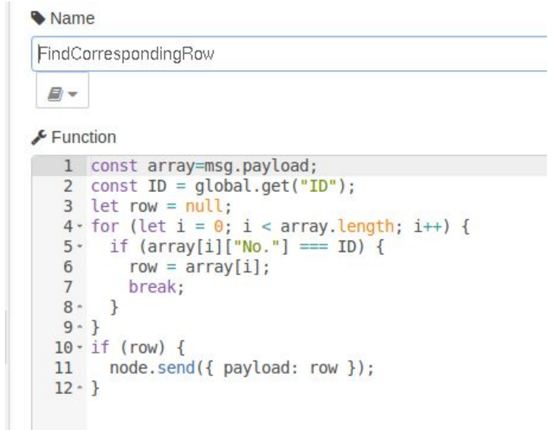
Here we are extracting ID coming from MQTT broker;

We keep count that for each ID and set the ID global after modulo operations where we will use this globalID in FindCorrespondingRow function to access row which has index globalID.


Name: ExtractingID

Function

```
var count = global.get("count") || 0;

if (count < 100) {
  var inputString = msg.payload;
  var idValue = parseInt(JSON.parse(inputString).id, 10) || 0;
  var result = (idValue + 1682) % 7711;
  count++;
  global.set("count", count);
  var countnew = count;
  global.set("ID",result)
  var globalID=global.get("ID");
  msg.payload = {
    "id": idValue,
    "result": result.toString(),
    "count": countnew,
    "globalID":globalID

  };
} else {
  msg.payload = "Count reached 100, execution stopped.";
  global.set("ID",-10)
}

```

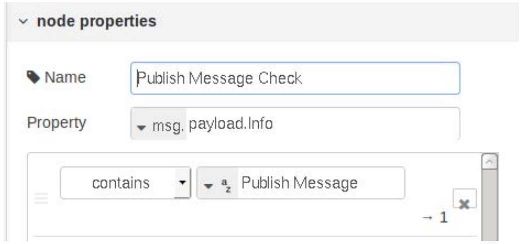

Name: FindCorrespondingRow

Function

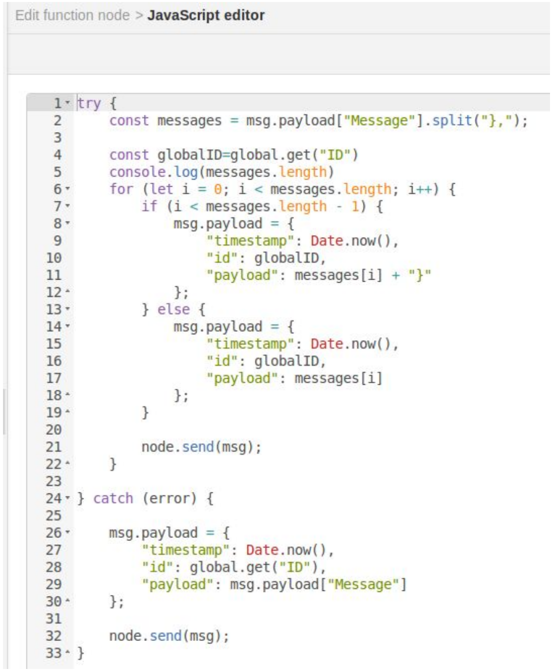
```
const array=msg.payload;
const ID = global.get("ID");
let row = null;
for (let i = 0; i < array.length; i++) {
  if (array[i]["No."] === ID) {
    row = array[i];
    break;
  }
}
if (row) {
  node.send({ payload: row });
}
```

Here we are finding corresponding rows using globalD and searching through the single array object that coming from csv module of node-red. If array[i][“No.] ==ID we are taking that row from csv.

Switch Publish Messages and Create Subscribe Messages



Name: Publish Message Check
Property: msg. payload.Info
contains ▾ Publish Message



```
try {
    const messages = msg.payload["Message"].split("},");

    const globalID=global.get("ID")
    console.log(messages.length)
    for (let i = 0; i < messages.length; i++) {
        if (i < messages.length - 1) {
            msg.payload = {
                "timestamp": Date.now(),
                "id": globalID,
                "payload": messages[i] + "}"
            };
        } else {
            msg.payload = {
                "timestamp": Date.now(),
                "id": globalID,
                "payload": messages[i]
            };
        }

        node.send(msg);
    }

} catch (error) {

    msg.payload = {
        "timestamp": Date.now(),
        "id": global.get("ID"),
        "payload": msg.payload["Message"]
    };

    node.send(msg);
}
```

We first used switch for filtering messages that contains Publish Message on their info.

Here we are trying to build;

{"timestamp":"CURRENT_TIMESTAMP","id":"PREVIOUS_ID","payload":"MQTT_PUBLISH_PAYLOAD"}

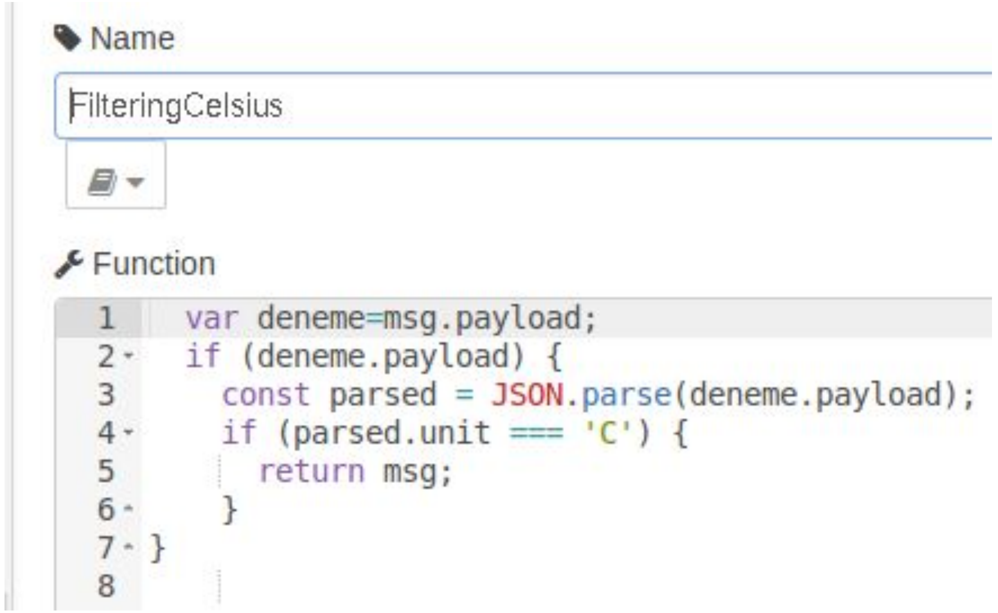
Message payload that will be published to broker.

We first splitting published messages where payloads are separated by comma, then if it includes publish messages we are creating different objects in terms of “payload” where timestamp and id are same.

If it does not include many publish messages we are sending only the payload.

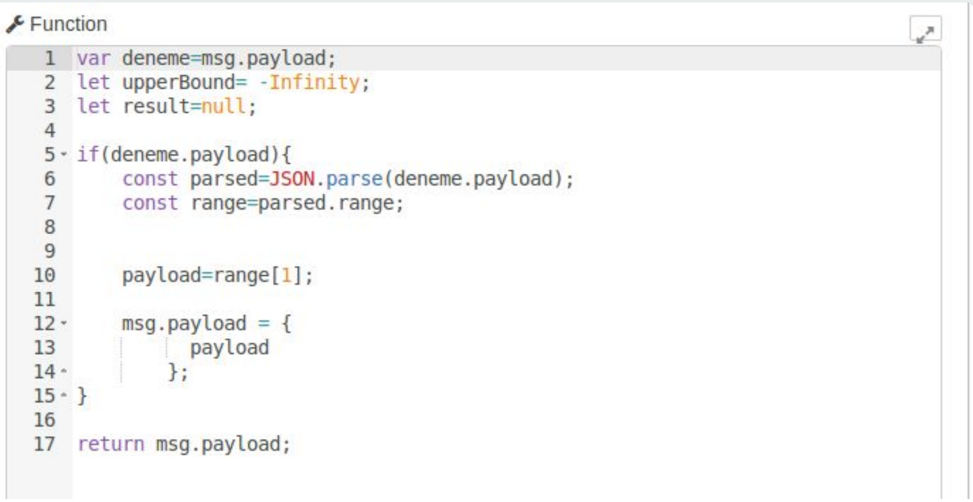
Filtering Celsius Publish Messages

This function represents FilteringCelsius in node-red flow, after we get subscribe message on topic /polimi/iot2023/challenge2/10901682 we first send it to json module to create a json object and we are taking only the temperature that has unit C which corresponds to Celsius.



Name: FilteringCelsius

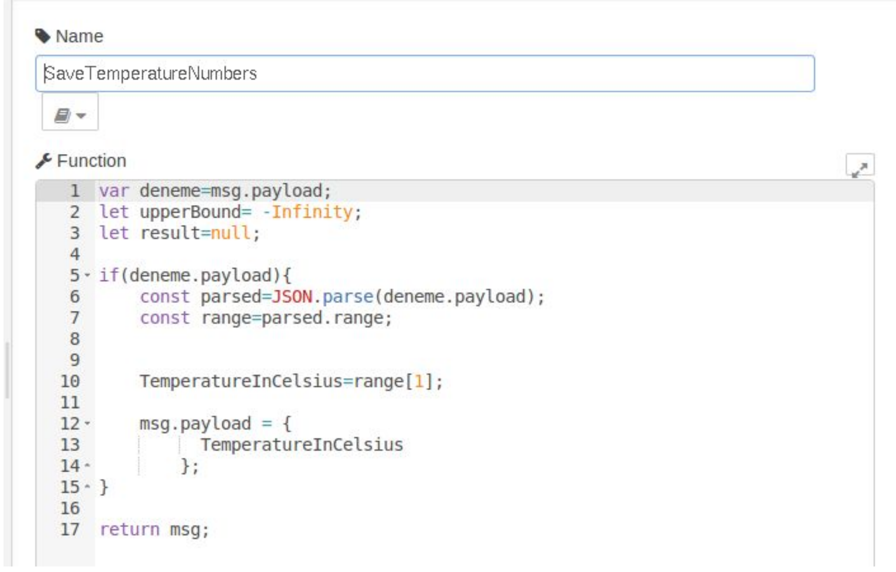
```
var deneme=msg.payload;
if (deneme.payload) {
  const parsed = JSON.parse(deneme.payload);
  if (parsed.unit === 'C') {
    return msg;
  }
}

```


Function

```
var deneme=msg.payload;
let upperBound= -Infinity;
let result=null;

if(deneme.payload){
    const parsed=JSON.parse(deneme.payload);
    const range=parsed.range;


    payload=range[1];

    msg.payload = {
        payload
    };
}

return msg.payload;
```

It is highRangeTemp function where after we are filtering messages according to temperature unit, we are taking upper bound of range and sending the msg.payload which is the temperature value for chart

It is same as above function, the only difference is we are taking not only temperature values but the temperature as an object.


Name

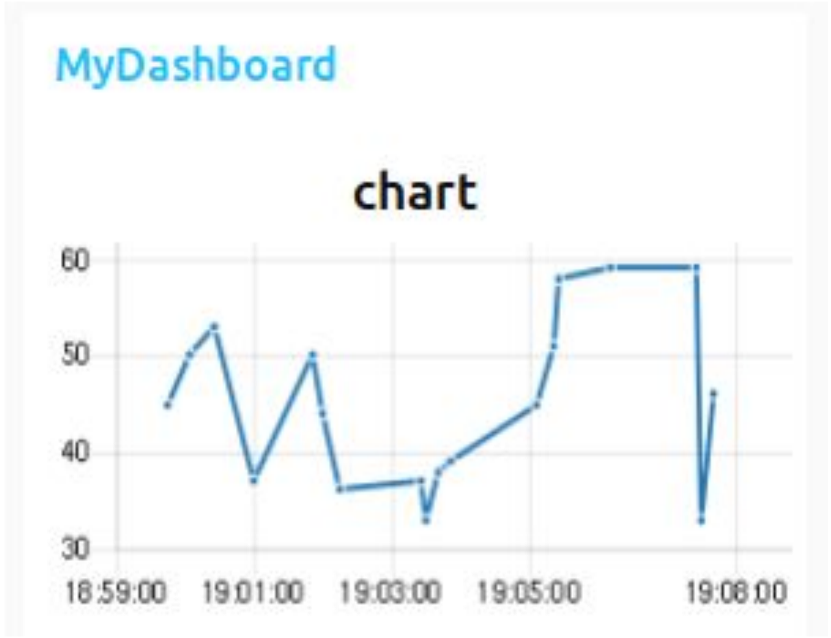
SaveTemperatureNumbers

Function

```
var deneme=msg.payload;
let upperBound= -Infinity;
let result=null;

if(deneme.payload){
    const parsed=JSON.parse(deneme.payload);
    const range=parsed.range;


    TemperatureInCelsius=range[1];

    msg.payload = {
        TemperatureInCelsius
    };
}

return msg;
```

Temperature Chart

Here we get temperature values from filtered publish messages that contains temperature with unit only in celsius. Y-Axis represents temperatures which is upperBound of the range in the payload of publish messages.


MyDashboard
chart
18:59:00
19:01:00
19:03:00
19:05:00
19:08:00