**IoT challenge2**

**Marco Barbieri 10684602 (leader), Matteo Bianchi 10698168**

**Intro:**

in this challenge we had to retrieve 100 random messages from a CSV file following the give instructions. For each received message we had to create a new message with the following payload: {"timestamp":"CURRENT\_TIMESTAMP","id":"PREVIOUS\_ID","payload":"MQTT\_PUBLISH\_PAYLOAD"}

in which:

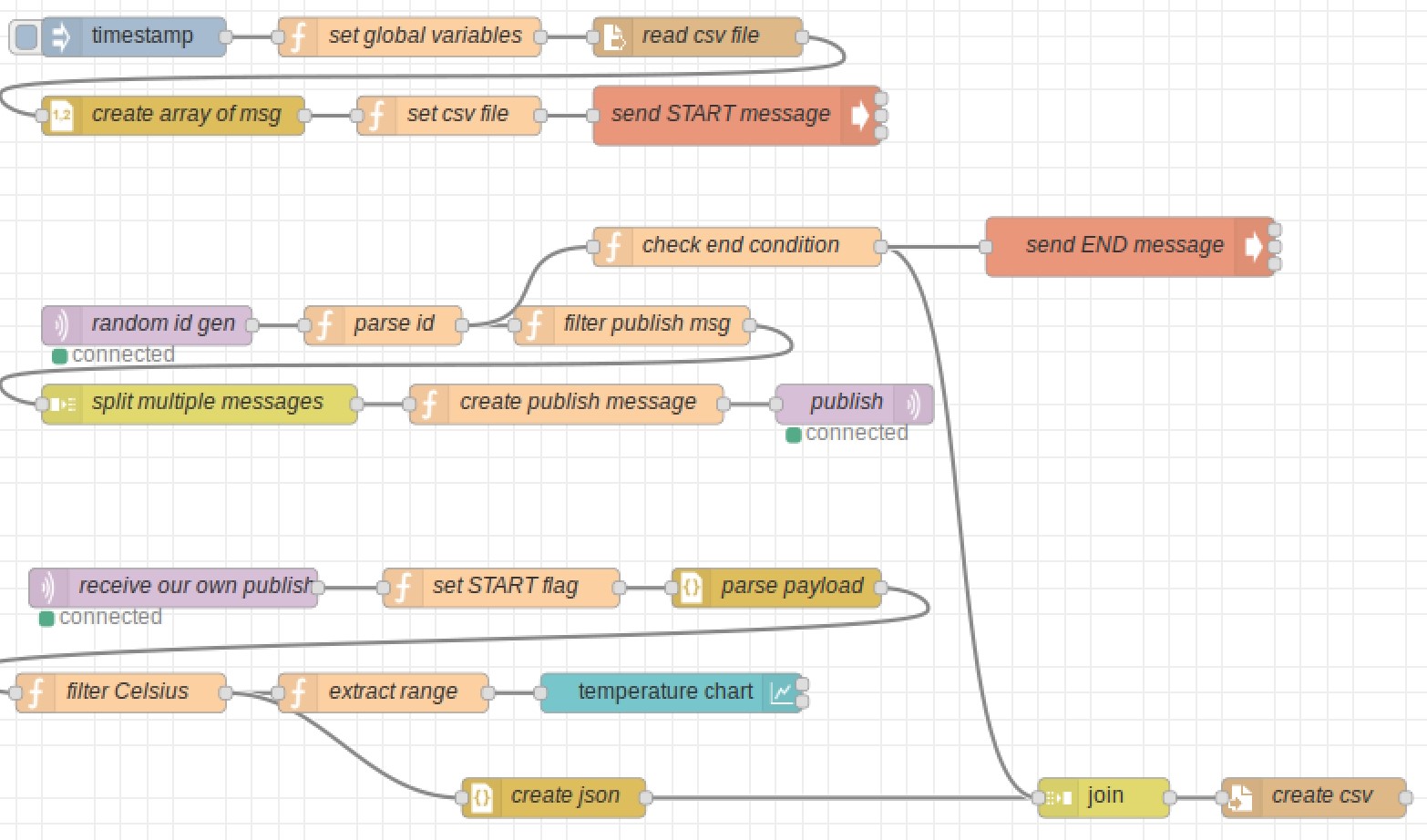
* CURRENT\_TIMESTAMP Current time at the moment of the sending.
* PREVIOUS\_ID Message id received from the subscription.
* MQTT\_PUBLISH\_PAYLOAD Payload from the CSV of the Publish message with frame number n.

and send the packet to the topic /polimi/iot2023/challenge2/10684602.

Then we had to subscribe to the same topic and read all the received messages, filtering the ones with temperature in Celsius and create a graph of the temperature over time.

In conclusion we had to create a CSV file containing the payload of the filtered messages.

**Node-red explanation:**



The diagram is divided in three phases: setup, publish and subscribe.

**Setup phase:**

In the setup phase we reset the global variables to their default value (START flag set to false, counter to 0 and the list of messages to null). We then read the csv file, create an array of messages and save it in the “file” global variable. We conclude this phase with the command to publish the START message:

**“mosquitto\_pub -h broker.hivemq.com -p 1883 -q 2 -m START -t /polimi/iot2023/challenge2/10684602”**

**Publish phase:**

In the publish phase we retrieve the id generated from the topic “polimi/challenge\_2/2023/id\_code\_generator/3”.

We had a function node to parse the message received, setting the “id” global variable to the correct value ((generated\_id + person\_id) % 7711) and handle the counter so that this node returns a message only if the counter is less than or equal to 100. This node is also used to end the process when the counter reaches 100, in this case a message with payload equal to “END” is sent and the START flag is set to false.

In the “filter publish message” node we use the array saved in “file” to access the message at index specified in the previous id. When we have the message we check that it is actually an MQTT message and that the payload contains the string “Publish Message”, in this way we only get the publish messages and we also exclude the publish receive messages.

Because the messages saved in the “file” array have a different structure from the messages used by node red we stored msg.Message inside msg.payload and return the message.

With the “split multiple messages” node, we handle the messages with multiple payloads, creating one message for each payload.

Then we use a function node called “create publish message” to construct the message to be sent, this node is very simple as we retrieve the needed values and concatenate them to create the publish message with the correct sintax.

To finish the publish phase, we used the mqtt publish node to send the created message at topic “/polimi/iot2023/challenge2/10684602”.

When the counter reaches 100, we send the END message as stated before. This activates the “send END message” node and executes the command:

**“mosquitto\_pub -h broker.hivemq.com -p 1883 -q 2 -m END -t /polimi/iot2023/challenge2/10684602”**

It also sends the same message with the property complete set to true so that the join node can complete its execution (join node is explained in the subscribe phase).

**Subscribe phase:**

In the subscribe phase we use a mqtt node to subscribe to the topic “/polimi/iot2023/challenge2/10684602”.

the first message received is the START message sent at the setup phase, when we receive this message, we set the START flag to true, by doing so we activate the publish phase.

We then use the “parse payload” node to create a JSON of the payload of the message received, in this way it is easier to filter the messages. If we receive malformed messages (missing parenthesis or unexpected tokens) this node throws an exception and stops the parsing of that message. This means that malformed messages are ignored, but the flow will continue to parse the next messages.

In the “filter Celsius” node, we just retreive the unit property of the payload and check if it equals “C”, only if this happens we return the message.

In the “extract range” node we just extract the upper bound of the range and send it to the chart node.

When the filtering of the message is complete, we also send the message to a JSON node so that it transforms the JSON into a string and sends it to a join node.

The join node then retrieves all these strings and waits to receive an END message. The END message has its complete property set to true so that the join node knows when to stop collecting strings.

To finish the publish phase, we take the list of strings generated by the join node and create a CSV file with it.

**Graph showing the results:**

