IOT_project3

Briscini matteo [10709075]

INDEX:

Requirements summary

Use node-red to parse a sequence of MQTT messages (saved in challenge3.csv) and perform different actions based on the message's content. In particular:

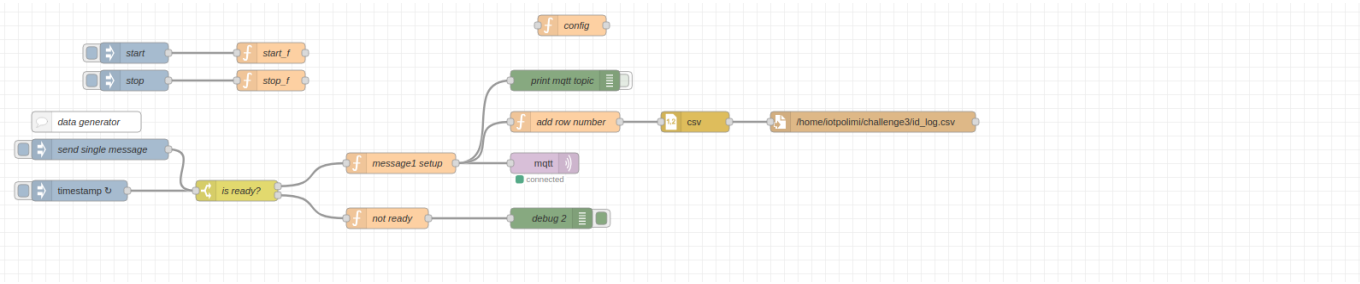
- if a message contains "Publish Message" the node will forward the message on the specified MQTT topic and the message payload is saved inside filtered_pubs.csv.
- if the message contains an MQTT ACK the node will increment a global ack counter, perform an HTTP request on ThingSpeack update the field1 value with the ack number, the message content is saved inside ack_log.csv.
- other messages will be discarded.

Implementation

We will split the whole implementation into 3 different phases: data generation, data receiving and message parsing, message reaction.

> **note:** the complete implementation is provided here

Data generation



In this phase the flow will generate random data save it in a CSV file and send it locally through MQTT on the "challenge3/id_generator" topic with the following payload. If the inject named "timestamp" is enabled message will be sent with a rate of 1 message every 5 seconds. All the messages sent in this phase are saved in a CSV file, named id_log.csv.

```
{"id": 7781, "timestamp":1710930219} //message payload example
```

"Is ready?" switch is used, combined with start and stop to emulate a physical switch on the device.

Relevant JS function & blocks

- **config (on start)**
  initializes all global variables to the desired values.

```
// Code added here will be run once
// whenever the node is started.
global.set("mqttDefaulChannel", "challenge3/id_generator");
global.set("ACKCounter", 0);
global.set("receivedMessagesCounter", 0);
global.set("tempCounter", 0);
global.set("thingSpeakKey","VI5VOWUDI8Z5GCF1")
```

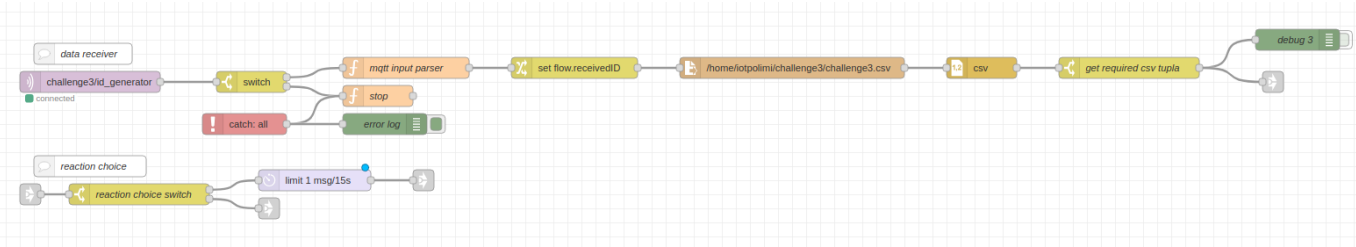
- **message1 setup (on message)**
  setup the received MQTT message.

```
let mqtt_topic = String(global.get("mqttDefaulChannel"));  //get topic
let jsonMessage = { "id": Math.random() * 5000, "timestamp": msg.payload};
//generate random paylod for the mqtt msg

msg.topic = mqtt_topic;
msg.payload = jsonMessage;

return msg;
```

Data receiving and message parsing



this block will receive up to 80 messages on "challenge3/id_generator" topic, and use the message id to get the correct row in the challenge3.csv file, based on the row content the flow will be reacting differently (as specified in requirement summary), in this phase the correct reaction is triggered.
when the flow has already received 80 messages or occurs in error (in every stage), the stop function sets "is ready?" to false stopping messages publishing on the "challenge3/id_generator" topic.

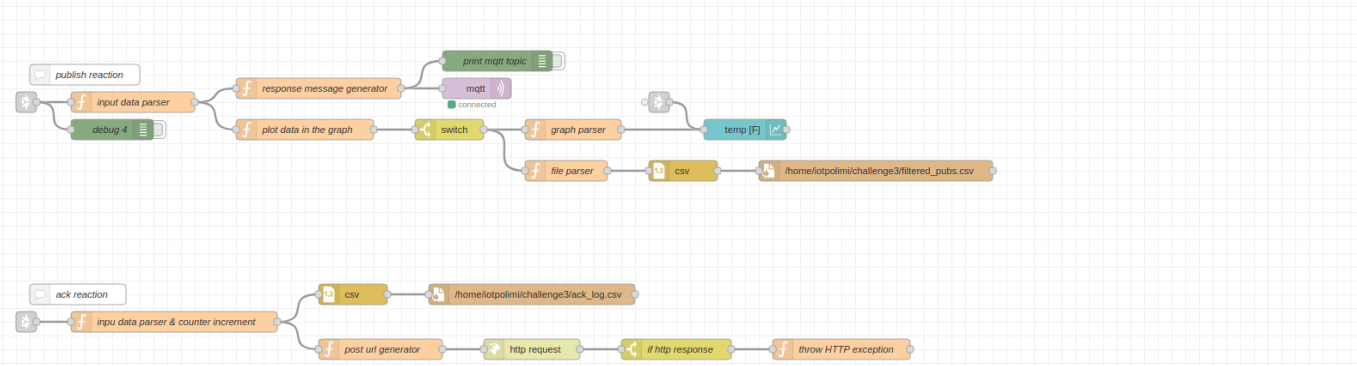
Relevant JS function & blocks

- **mqtt input parser (on message)**
  parse the received MQTT message.

```
global.set("receivedMessagesCounter",
(global.get("receivedMessagesCounter")+1));  //increment the message counter
msg.payload = parseInt(msg.payload.id % 7711);  //comput the row number
return msg;
```

- **get required csv tupla (on message)**
  select the correct row from the CSV file using the received id previously saved as a flow variable.
- **reaction choice switch**
  based on the receive message trigger the correct reaction, if required.

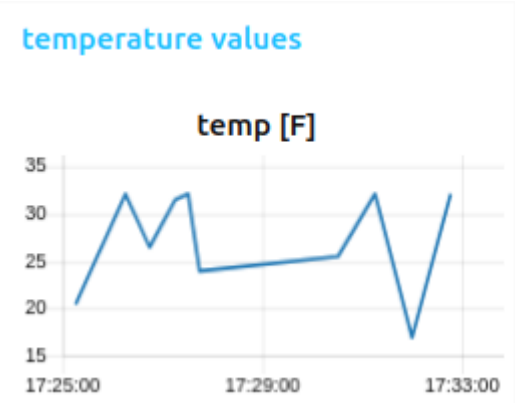
Message reaction



the requirements ask to react to 2 different message classes: published and ack messages.

Publish reaction

This block reacts to MQTT messages of class "Publish Message", in particular publishing messages with specified payload on the required MQTT topic. Additionally, if a message payload contains a temperature in Fahrenheit that payload is saved in the filtered_pubs.csv file and its value is plotted on a UI graph, a screenshot of this graph is provided below.



As shown in the following example a single can contain multiple MQTT topics and payload, in this terms the "input data parser" function block has the role of splitting the input message (a single string) in two arrays with the topics and the payload of all the messages.

```
// received info & payload example, containing all the message payload and top
where forward
"Publish Message [hospital/room2], Publish Message [hospital/building5], Publish
Message [hospital/department2]","{""range"": [4, 50], ""description"": ""Room
Temperature"", ""type"": ""temperature"", ""unit"": ""C"", ""lat"": 66, ""long"":
92},{""type"": ""temperature"", ""lat"": 81, ""long"": 95, ""unit"": ""C"",
""range"": [3, 50], ""description"": ""Room Temperature""},{""description"":
""Room Temperature"", ""lat"": 55, ""unit"": ""K"", ""type"": ""temperature"",
""long"": 88, ""range"": [7, 41]},"
```

Relevant JS function & blocks

- **input data parser**
- **response messages generator**
- **plot data in the graph**
- **graph parser**
- **file parser**

Ack reaction

This block reacts to MQTT messages containing an ACK (of any type); ack messages are saved on ack_log.csv file in terms of timestamp, sub_id, msg_type, also the flow will count the total amount of ack messages (on a global counter) and publish that data on ThingSpeack.

Relevant JS function & blocks

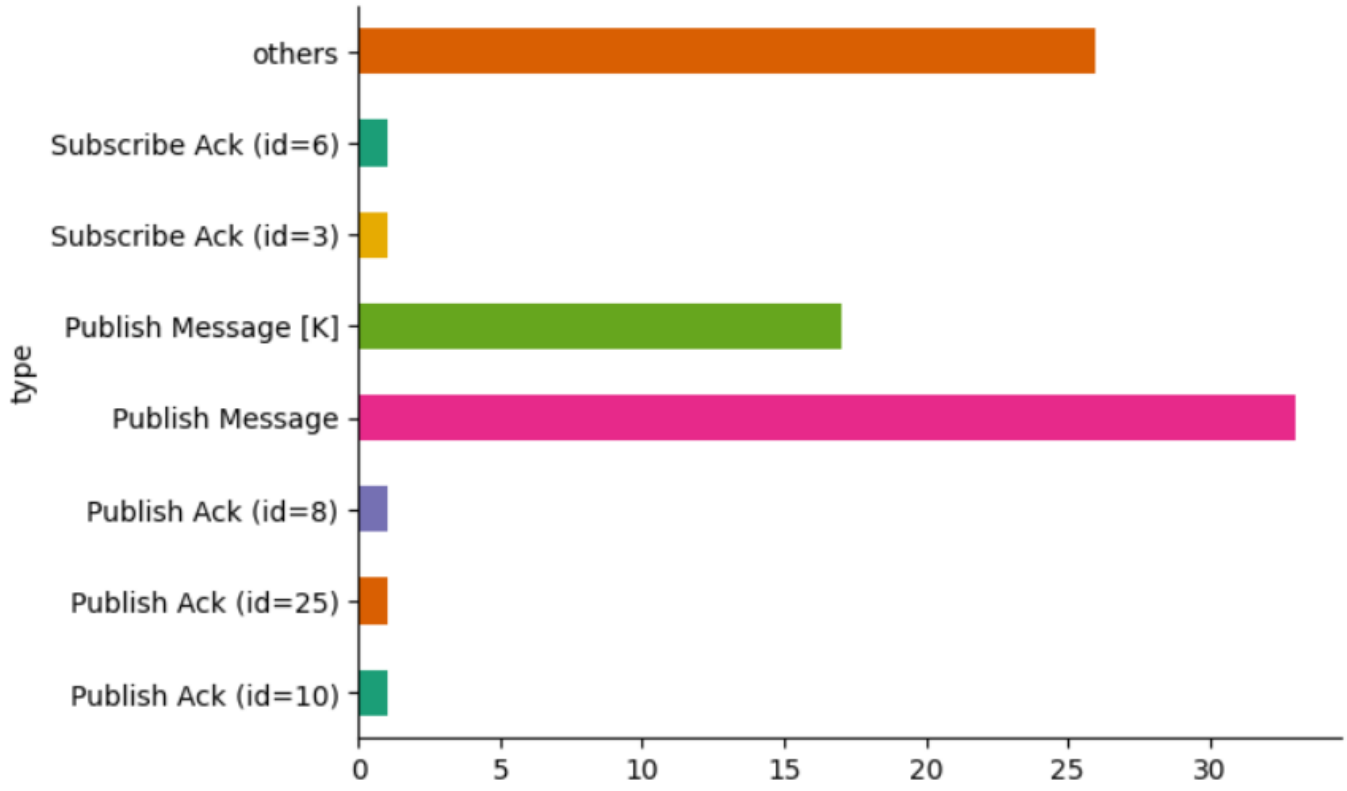
- **input data parser & counter increment**
- **post url generator**
- **throw HTTP exception**

Testing

A Python tool for testing purposes is provided here. The idea for the testing is to verify the correct correlation between the various CSV published from the flow on running.

> **note:** to generate a CSV file useful for testing, it is necessary to disable the limit on messages per second for the "publish reaction" brach otherwise some messages can be discarded. CSV files generate in such way are provided in this folder.

following is provided a test output:



Entire output

index	id	type	response
0	989	Publish Ack (id=25)	
1	1008	others	
2	845	Publish Message [K]	
3	2964	Publish Message	
4	1154	Publish Message [K]	
5	2529	Publish Message [K]	
6	446	Publish Message	
7	3362	others	
8	4807	Publish Message	
9	154	others	
10	3765	Publish Message	
11	1565	Publish Message [K]	
12	3464	Publish Message [K]	
13	1536	others	
14	1632	Publish Message	

index	id	type	response
15	498	Publish Message	
16	2496	Publish Message	
17	4636	others	
18	2265	others	
19	399	Publish Message [K]	
20	2063	Publish Message	
21	3673	others	
22	1496	others	
23	1247	Publish Message	
24	3465	Publish Message [K]	
25	339	others	
26	1514	Publish Message [K]	
27	632	Publish Ack (id=10)	
28	4742	Publish Message [K]	
29	2693	Publish Message	
30	1244	others	
31	2093	Publish Message	
32	3773	Publish Message	
33	1434	Publish Message	
34	3709	Publish Message	
35	162	others	
36	2827	Publish Message [K]	
37	3762	others	
38	2275	Publish Message	
39	4015	Publish Message	
40	3349	Publish Message [K]	
41	15	others	
42	3567	Publish Message	
43	4883	others	
44	1907	others	

index	id	type	response
45	3529	Publish Message	
46	101	Subscribe Ack (id=6)	
47	2820	others	
48	238	others	
49	3845	Publish Message	
50	4179	others	
51	1509	others	
52	226	others	
53	649	Publish Message [K]	
54	1279	others	
55	4732	Publish Message	
56	623	Publish Ack (id=8)	
57	3397	Publish Message	
58	1940	others	
59	3873	Publish Message [K]	
60	654	Publish Message	
61	2956	Publish Message [K]	
62	4226	Publish Message	
63	2340	Publish Message	
64	4664	Publish Message [K]	
65	921	Publish Message	
66	3334	Publish Message	
67	1795	Publish Message	
68	3307	others	
69	4433	Publish Message [K]	
70	986	Publish Message [K]	
71	1190	Publish Message	
72	391	Publish Message	
73	3709	Publish Message	
74	3746	others	

index	id	type	response
75	1965	Publish Message	
76	1807	others	
77	3600	Publish Message	
78	3810	Publish Message	
79	118	Subscribe Ack (id=3)	
80	1623	others	