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:

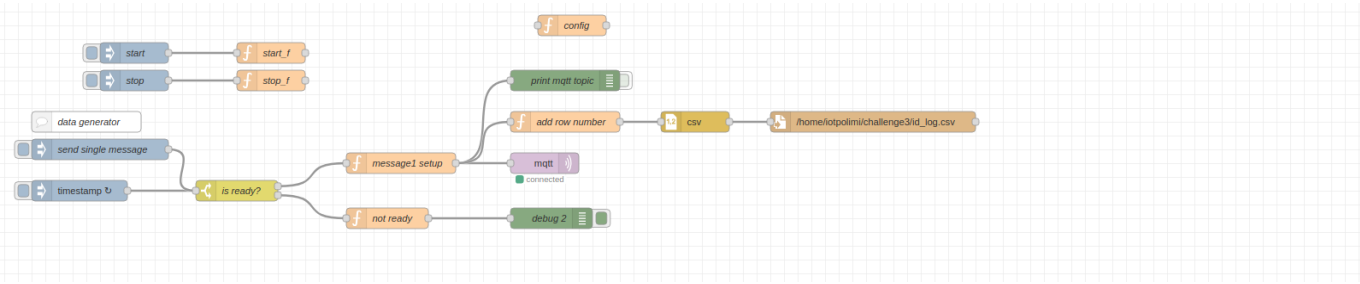
- if a message contains "Publish Message" the node will forward the message on the specified MQTT topic and the message payload is saved inside filtered_pubs.csv.
- if the message contains an MQTT ACK the node will increment a global ack counter, perform an HTTP request on ThingSpeack update the field1 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")
```

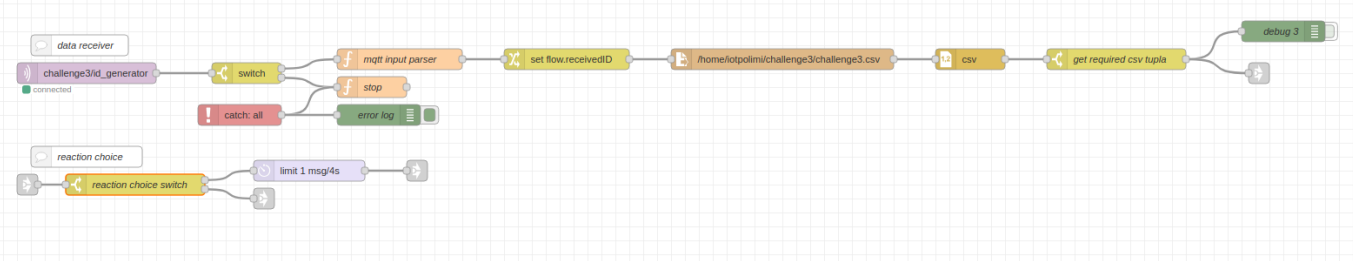
- **message1 setup (on message)**
  setup the received MQTT message.

```
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.

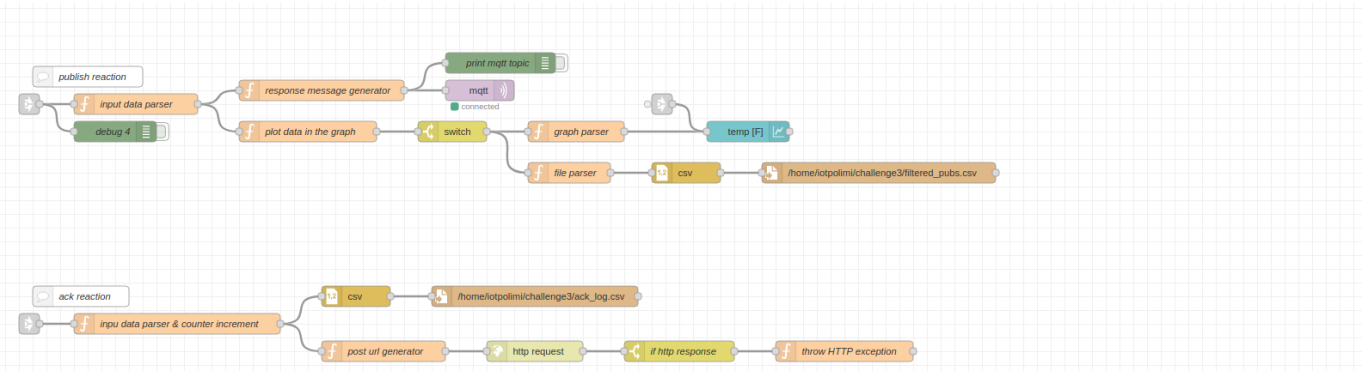
Relevant JS function & blocks

- **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.

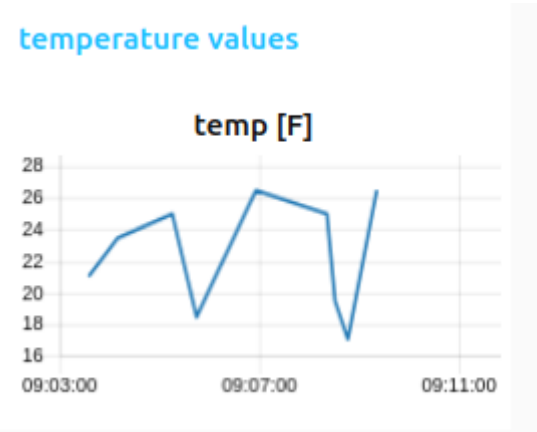
Message reaction



the requirements ask to react to 2 different classes of messages: pubblish messages and ack messages.

publish reaction

This block react to MQTT messages of class "Publish Message", in particular pubblishing messages with specified payload on the required MQTT topic. Additional 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 contains multiple MQTT topic and payload, in this terms the "input data parser" function block has the role to split the input message (a single string) in two arrays with the topics and the playload 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 react to MQTT messages containing an ACK (of any types); ack messages are save on ack_log.csv file in terms of: timestamp, sub_id, msg_type, also the flow will count the total ammount of ack messages (on a global counter) and pubblish that data on ThingSpeack.

Relevant JS function & blocks

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