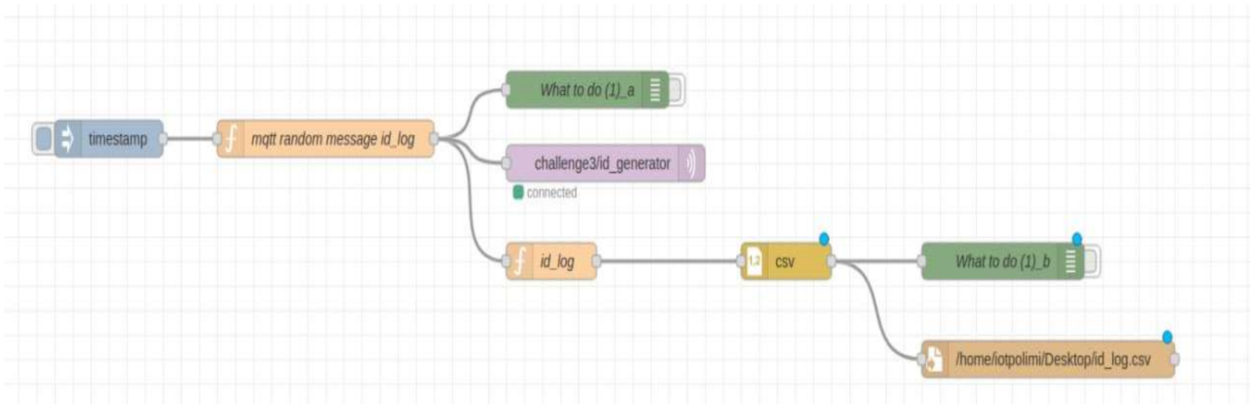
IoT Challenge 3

Iman Javaheri Neyestanak	**10976128**
Peyman Javaheri Neyestanak	**10971058**

First of all, we need to open 3 terminals on the machine: one for “mosquitto -p 1884”, another one for “mosquitto_sub -h localhost -p 1884 -t "#" ” and the last one for running the Node-Red. We set all in one flow.

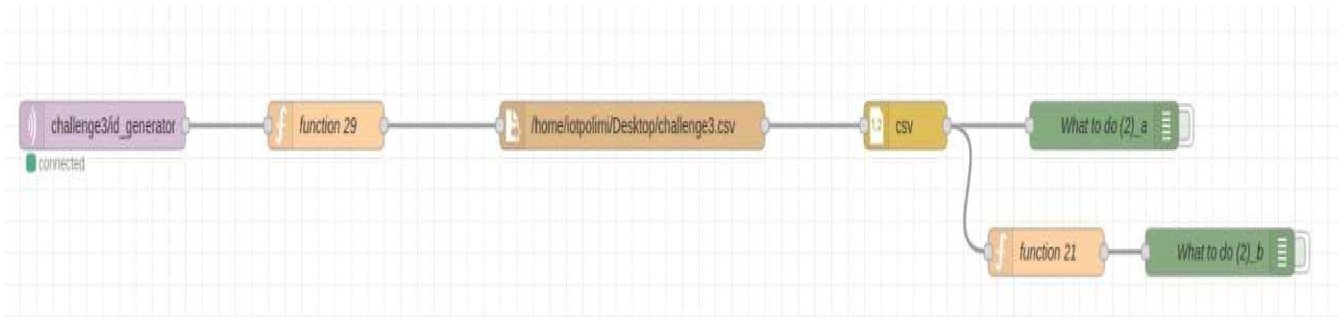
What to do? (1)

The node timestamp initiates a flow at intervals of 5 seconds, generating the current time with Date.now() function and transmitting it to the designated function. To publish MQTT messages to the local broker, the MQTT out node is utilized, and configured to connect to the server at localhost on port 1884. Within the 'mqtt random message id_log' function, the topic under which messages are published is identified as "challenge3/id_generator". A random variable is generated within the range of 0 to 30000, and both the timestamp and the random variable are merged as a payload, and transmitted to the broker. The resultant message payload takes the form of "{'id': rand_id, 'timestamp': time}". Subsequently, the task needs the storage of the generated IDs and timestamps in a CSV file, where each row corresponds to a single message. A context variable is established to track the row number, incrementing with each execution of the code and the addition of a new message. A CSV parser is introduced to organize the messages into three columns: Number, ID, and Timestamp. Numerical values are designated as the inputs for the CSV node. Here is the flow of this part:



What to do? (2)

With the node ' challenge3/id_generator ' we subscribe to the topic " challenge3/id_generator" in the local host server. Every time we get a new message subscription; we go through all the rows of the CSV file " challenge3.csv " which is provided to us. We check column N0. and compare them with the computed remainder of the id divided by 7711. The row that is matched to the value is the output payload of function 21. Here is the Flow of this part:



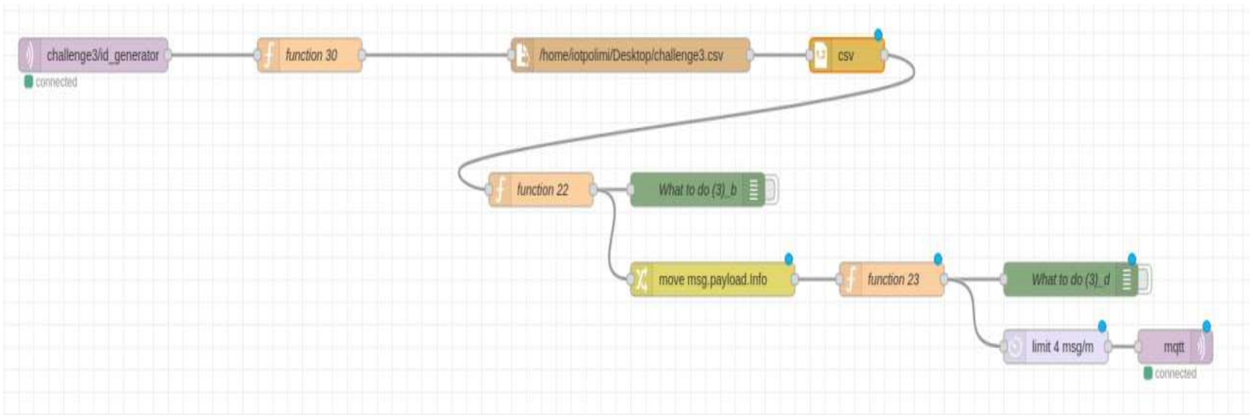
What to do? (3)

In the third part, we're tasked with sending a publish message with the same topic if the previous message found in the challenge3.csv file is a PUBLISH MQTT message. In function 22, we check if the payload's info is "Publish Message" using the code "if (payload.Info == "Publish Message")". If true, we proceed and return the message as a payload to the next node. The "move msg.payload.Info" node separates the info part of the message containing the topic and "Publish Message". In function 23, we retrieve the current time using "var time = Date.now();" and then extract the topic part of the payload message as an input using:

"let new_topic = str_check_publish.substring(str_check_publish.indexOf('[') + 1, str_check_publish.indexOf(']'));"

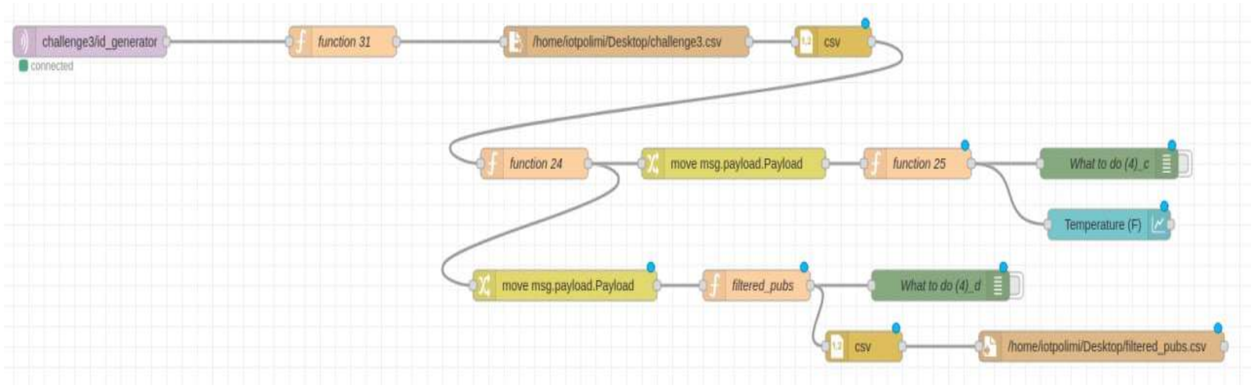
If the message is 'Publish Message', we return the payload message containing time, ID, and topic in the following format, for example: {"timestamp":"CURRENT_TIMESTAMP","id":"SUB_ID","topic":"MQTT_PUBLISH_TOPIC", "payload":"MQTT_PUBLISH_PAYLOAD"}.

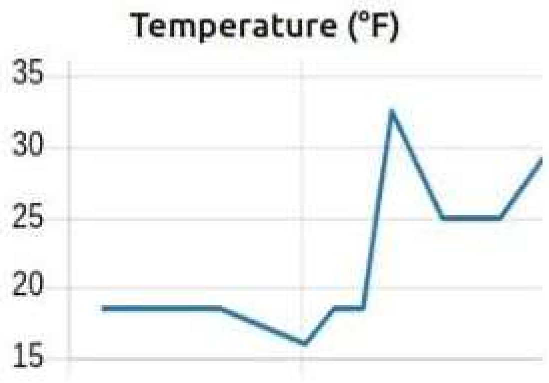
To comply with the requirement of limiting the number of messages to 4 per minute, we include a 'delay node' to enforce this restriction, adjusting the rate to 4 when double-clicking on the node. With the message format requested for publishing to the localhost ready, we transmit it to the 'mqtt out' node for publication. Here is the flow of this part:



What to do? (4)

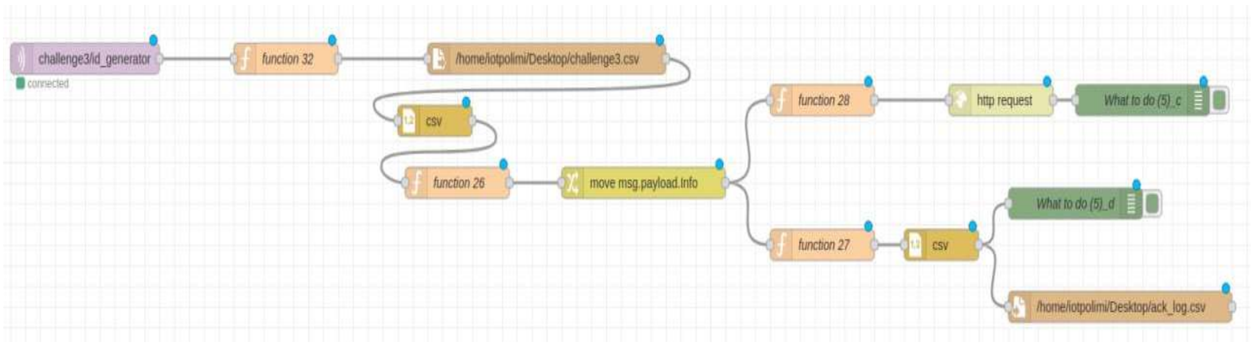
In this part, we are asked to check if the publish message we published in the previous part contains temperature in Fahrenheit in the payload, get it, and plot a chart showing temperature values as shown in the below figure, Temperature(F). In function 25, we check for every payload if it is temperature in F, we get the range, compute the mean value, and return the mean as payload msg. Then to plot it we insert the node 'chart' and set the Label, x-axis, y-axis, and range in the properties of the node. Afterward, for saving every message in Temp(F) to the 'filtered_pubs.csv' we separate the No.,LONG, LAT, MEAN_VALUE, TYPE, UNIT, DESCRIPTION and make the requested format for payload message. Here is the output chart temperature and flow of this part:

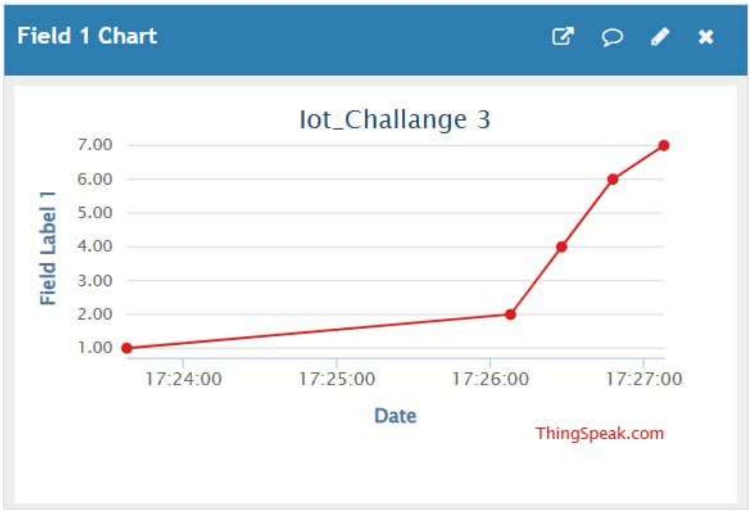




What to do? (5)

In this part, we examine the message matched to the row with No = N, where N is the remainder of the ID divided by 7711. If the message contains ACK Messages (Publish Ack, Connect Ack, Sub/Unsub Ack), we increase the global counter and save it in the ack_log.csv file with the requested format, such as ' 1,1713741543728,32456,Publish Ack (id=12551) '. This output represents the timestamp, ID of the subscription message, and the type of ACK, respectively. Subsequently, to transmit the global ACK counter to thingspeak.com, we verify if the payload info contains ACK. If it is ok, we establish a global counter and increment it with each additional ACK message, as illustrated in field 1 in Thinsspeak. Initially, we make the Thingspeak channel link public and define the API key and URL address in the code to access our channel. Then, we provide this payload to an HTTP GET request, which facilitates communication with Thingspeak. Here is the Thingspeak chart and flow of this part:





Channel ID: 2930856