IoT 2023 Challenge 2

Group Members

Team Leader: Riaz Luis Ahmed.

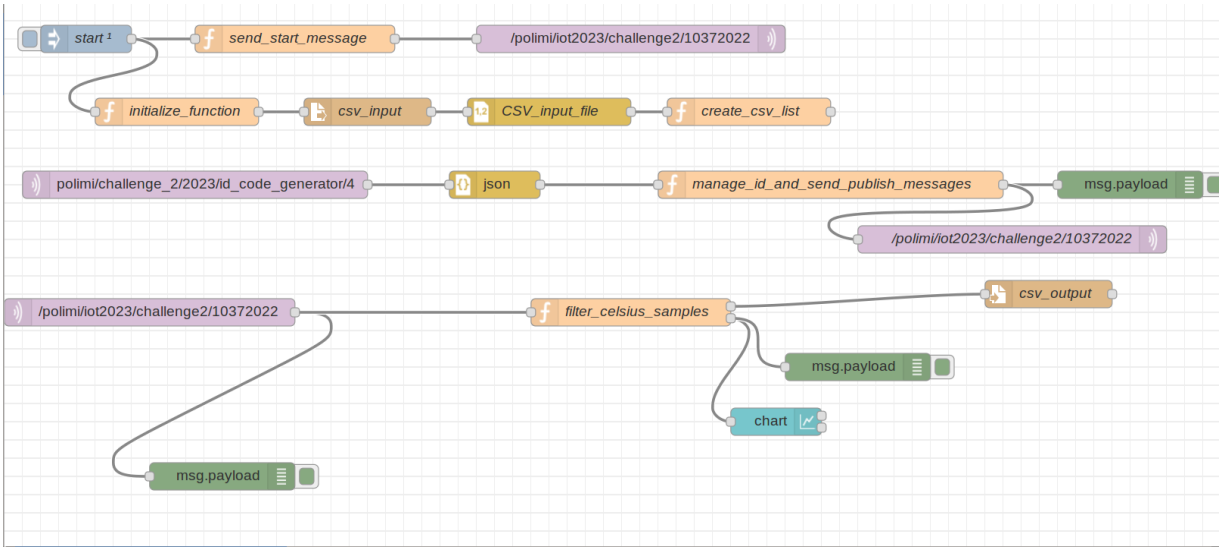
Personal Code: 10372022

Second Member: Mario Cela.

Personal Code: 10685242

Report

In the following image, you can see all the nodes that we used to implement the requested system:



It is provided an explanation on behaviors and relations between node, in order to allow a better comprehension of the implementation.

- Inject node: **start**. The node is used as a trigger to start the execution of the nodes it is linked to: **send_start_message** and **initialize_function**. The trigger is sent 0.5 seconds after the deploy action is performed.
- Function node: **send_start_message**. As specified in the slides, the computation must start with a message containing a “START” string. This message is created by this node, which is ready to send the content to the topic **/polimi/iot2023/challenge2/10372022**. For this reason, this node is linked to the following MQTT node. Notice that in the code of the node it is implemented and used a `sleep` function, which is used to satisfy one of the requirements shown in the slides: the subscription to the topic **/polimi/iot2023/challenge2/10372022** must be performed before starting sending messages to it. So, the sleep is used to coordinate the nodes and make sure that the subscribe is done before that other nodes publish into that topic.
- MQTT node: **/polimi/iot2023/challenge2/10372022** (1). As mentioned before, this node performs the publish of the “START” message in order to effectively start the processing of the messages.
- Function node: **initialize_function.** This is the second node triggered by the **start** inject node. It initializes all the global variables that will be used by the system. These variables are:
  - `csv_file`: it is an array where we saved all the rows of the the csv file provided in input (challenge2023_2.csv);
  - `count`: used to count the number of messages read, in order to satisfy the request of processing a total of 100 messages;
  - `initialized`: when the **filter_celsius_samples** function node (which is subscribed to the topic **/polimi/iot2023/challenge2/10372022**) receives the “START” message, it sets the value of this global variable to 1, and the processing of the messages in input performed by the **manage_id_and_send_publish_messages** function node is allowed when the value of `initialized` is 1.
- Read File node: **csv_input**. This node is used to open a file in read mode from the local machine, specifying the path in which it is located.

- CSV node: **CSV_input_file**. This node is used to create a flow of messages where each message represent a row of the CSV file that has been opened before. These messages will be handled by the **create_csv_list** function node.
- Function node: **create_csv_list**. This node is used to save the content of the whole csv file into the global variable `csv_file` explained before. It works as follow: every time that the function node receives a message, it is receiving a row of the csv file in input. So, **create_csv_list** puts into a local variable the content of the global variable, and pushes into the end of the array the new received row. Finally, stores back the array into the global variable `csv_file`.

- MQTT node: **/polimi/challenge_2/2023/id_code_generator/4**. We subscribe to this topic in order to receive messages containing the ID to be processed. Notice that we used the topic shown in the slides adding "/4" at the end of it, since there were people publishing into the topic and it was impossible to implement and debug the system.
- JSON node: **json**. This node is used to convert the messages received from the topic (that are in JSON format) into JavaScript objects. Then, these objects are sent to the core of the system: **manage_id_and_send_publish_messages**.
- Function node: **manage_id_and_send_publish_messages**. Starting from line 27 of the code, we see three variables:
  - `check`: it contains the value of the `initialized` global variable;
  - `c`: it contains the value of the `count` global variable;
  - `msgs`: it is an array where we will save the message that will be sent to the MQTT node to be published. We used an array since it can happen to have more publish messages from the same row of the csv file.

  Then we have an if, which decides if the received message can be processed or not. The conditions are:

  - `check == 1`: the initialized global variable must be set to 1, otherwise we would not be able to receive the messages that we will publish;

- `c < 100`: we must process a total of 100 messages;
- `msg.payload !== ""`: we ignore those messages with null payload.

Once the conditions are met, we can process the message. We generate the ID as specified in the slides and we get the corresponding row of the csv file using the global variable.

Then, we check if it is a publish message or not.

If it is, we split the publish messages (since they may be more than one) and their contents (for the latter, it is used the `get_contents` function, defined between lines 13 and 25). Then, for each of them, we create the payload of the message that will be then published to the **/polimi/iot2023/challenge2/10372022** topic using the `generate_payload` function, defined between lines 1 and 11. Finally, we generate the message object and we push it into the `msgs` array mentioned before.

When all the processing is done, the messages are published.

- MQTT node: **/polimi/iot2023/challenge2/10372022** (2). This is the node used to publish the messages coming from the **manage_id_and_send_publish_messages** function node. Notice that in the setting of the MQTT node it has not been specified the topic, since all the incoming messages have the topic filed already specified.

- MQTT node: **/polimi/iot2023/challenge2/10372022 (3)**: this MQTT node outputs to node **filter_celsius_samples** the messages which were sent in the channel **/polimi/iot2023/challenge2/10372022** by **node manage_id_and_send_publish_messages**.
- Function node: **filter_celsius_samples**. This function node takes input messages from **MQTT /polimi/iot2023/challenge2/10372022 (3)** and outputs messages to nodes **csv_output**, **chart**. When the node detects a START message it sets the global variable `initialized` to 1 otherwise when it detects an END message it sets the `initialized` to 0. The `initialized` global variable is then used by node **manage_id_and_send_publish_messages** to trigger the computation of the `n`

variable and the publishing of messages. We use the condition `else if(msg.payload != "END")` to receive all the messages before the END message is published. Then we get from the global variable `temperatures_received` the number of temperature data we have already received in this node; this data is then used to update the number of temperature samples received and update its value in the global variable `temperatures_received`. We also exclude messages that don't contain valid payloads and messages that don't contain measures taken in Celsius degrees using the

```
f (p !== undefined) {
       if (p["unit"] == "C")
...
```

clauses. Only in the case where the if conditions are respected we return the received messages and the temperature sample in two different outputs through the command `return [msg,temp_msg];`

- Write File node: **csv_output**. This node writes the messages sent by node **filter_celsius_samples** to an output.csv file.
- Chart node: **chart**. This node draws a temperature vs time plot using as input the temperature samples `temp_msg` messages sent by node **filter_celsius.**