Third Challenge - Internet Of Things

Professor: Redondi Alessandro - Academic Year: 2024/2025

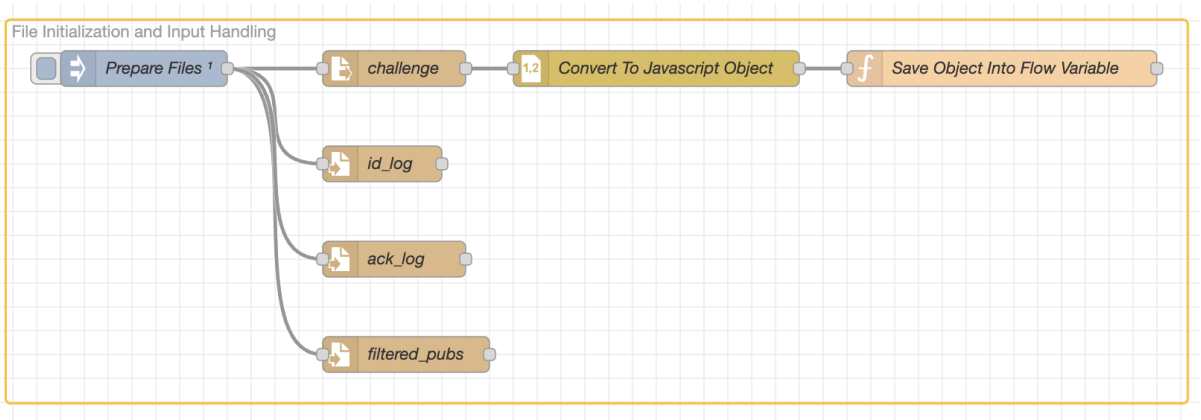
Pianalto Riccardo (Person Code: 10835980)
Pica Mirko (Person Code: 10811404)

Flow Structure And Explanation

The project flow was divided into **three main sections**, each focusing on a specific aspect of the data handling and messaging process.

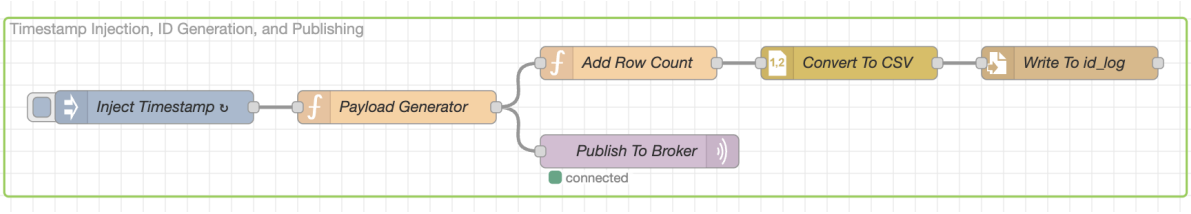
1. File Initialization and Input Handling

The first section of the flow was dedicated to preparing the environment and loading the necessary data. In this stage, **all required files were initialized**. Specifically, the three output CSV files were simply overwritten at the start of the execution to ensure they were empty and ready to store fresh data. For the input, the **challenge3.csv** file containing the data was read. The contents of this file were stored into a **flow variable**, allowing easy access and use across the rest of the flow without having to re-read the file each time it was needed.



2. Timestamp Injection, ID Generation, and Publishing

The second section of the flow introduced the mechanism for generating and publishing data. A **timestamp** was automatically injected at fixed intervals of five seconds. Alongside this, a **random number** was generated within a range of 0 to 30.000, acting as a unique ID for each message.
These generated IDs, paired with their corresponding timestamps, were then published to the MQTT topic **challenge3/id_generator**.
In parallel with the publishing operation, the same data was also recorded in the **id_log.csv** file. Each entry in the log included a **row number**, allowing us to keep track of the sequence in which IDs were created and sent.

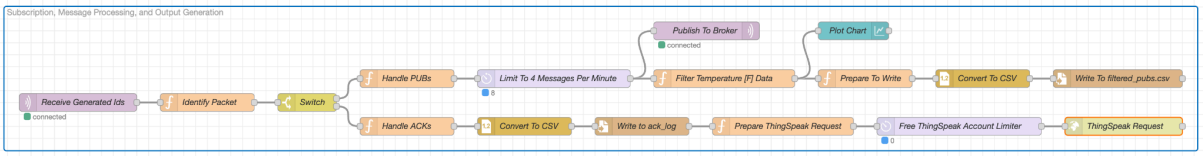


3. Subscription, Message Processing, and Output Generation

The final and most complex section of the flow involved subscribing to the MQTT channel and processing incoming messages. Each time a message was received, a counter was incremented. This counter was used to enforce a **processing limit**: once **80 messages** had been handled, the system would stop further processing to meet the specification.
Upon receiving a message, the flow computed a **modulo operation** using the received ID, which allowed us to retrieve the corresponding packet from the original CSV input. The packet was then analyzed to determine its type (for clarity all this was done in the *Identify Packet* node, other nodes should be clear).
Only two packet types were of interest: **publish and ack**. All other packet types were ignored.
Depending on the type, the processing was split into two branches using a switch node:

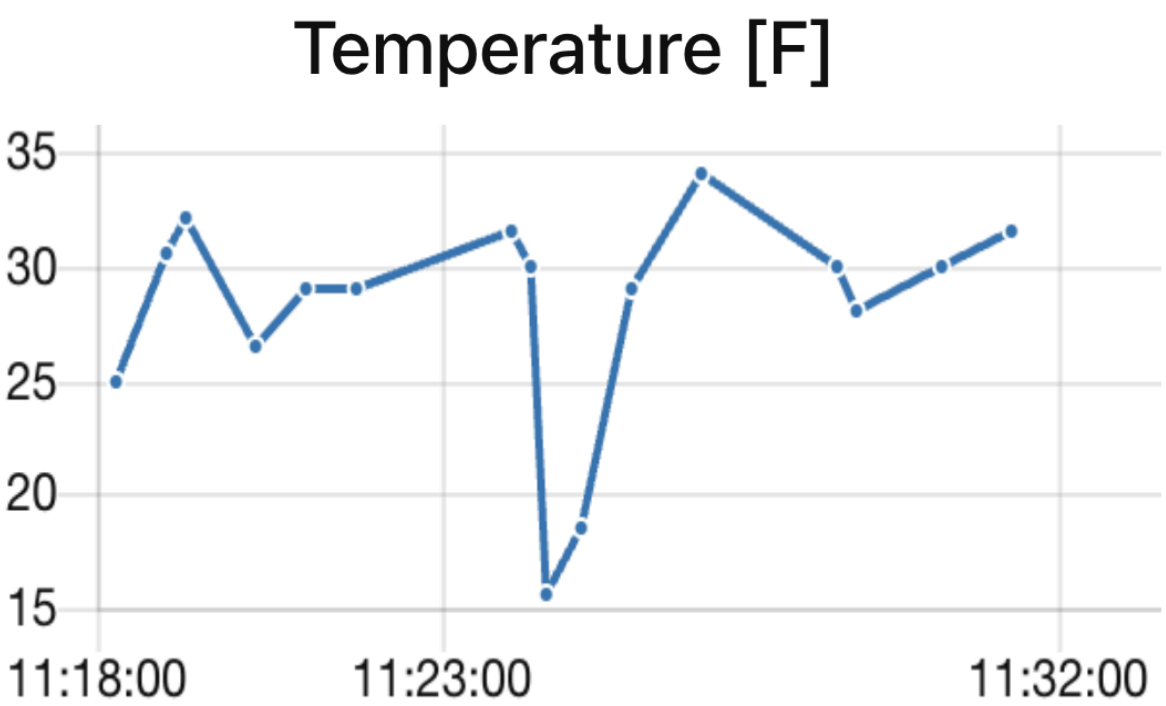
- **Acknowledgment (Ack) Branch:**
  In the case of an ack packet, the relevant data was prepared and logged into the **ack_log.csv** file. Then, a request was formatted and sent via **HTTP GET** to the ThingSpeak service (we added a rate limiter to bypass the free account limit).

- **Publish Branch:**
  If the packet was of type publish, a more complex handling followed. First, the topics and all included payloads were extracted and **sent one by one** (since a packet could contain multiple publish messages). Each payload was then passed through a **rate limiter** to control the frequency of outgoing messages. After rate limiting, each payload was **published to** its designated **MQTT** topic. In addition to publishing, the flow also **checked each payload for temperature** data expressed in Fahrenheit. If such data was found, the payload was forwarded to a **Node-RED chart** for live visualization. At the same time, these filtered messages were stored in the **filtered_pubs.csv**.



Charts

Node-RED chart



ThingSpeak chart

We put this to show how the graph was after the run (because now it's full of other runs and is not clear on the public channel).