IoT Challenge 3

1) Name: Jiaxiang Yi (leader)
Person Code: 10765316

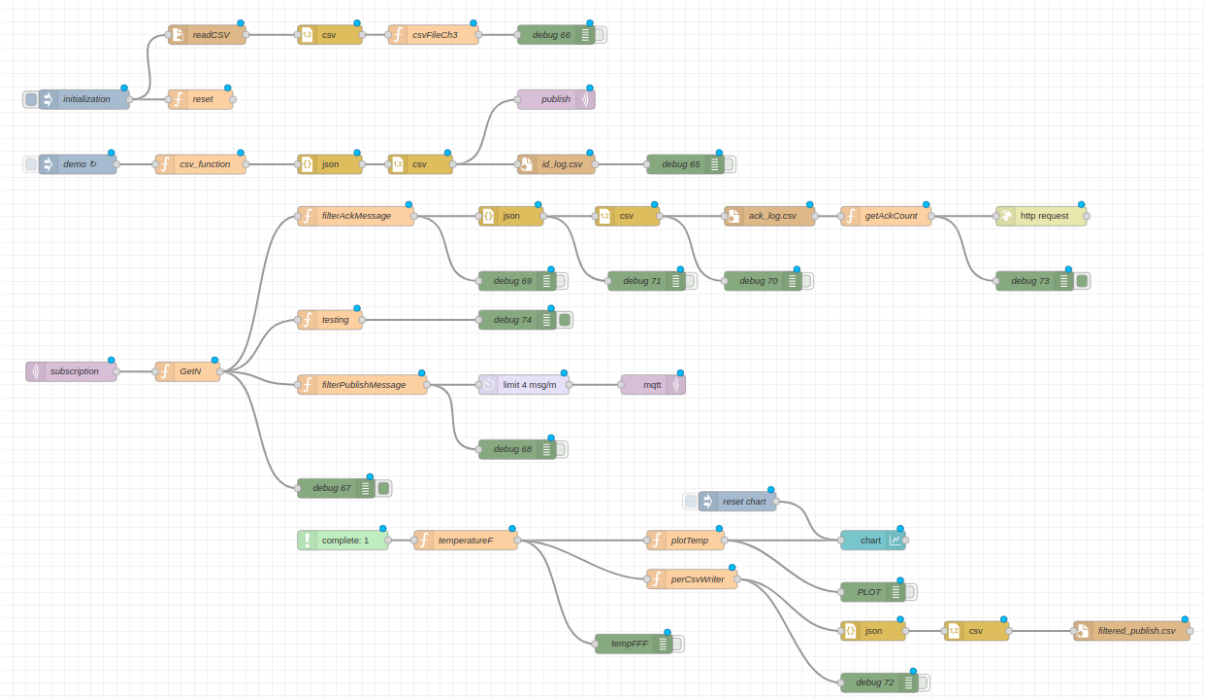
2) Name: Simona Cai
Person Code: 10752734

3) Name: Yang Hao Mao
Person Code: 10705881

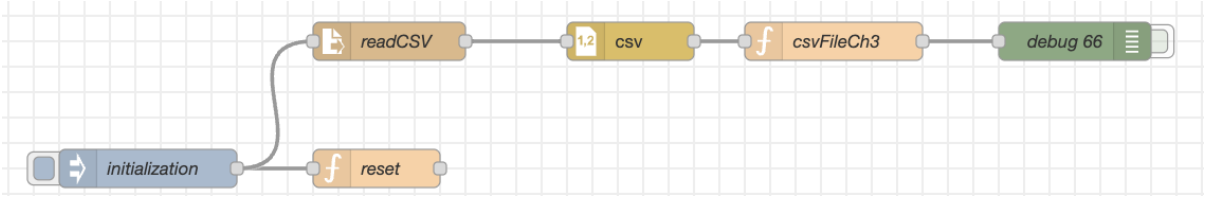
Thingspeak channel ID: https://thingspeak.com/channels/2489627

NOTE: All the files in the flow are referred with a local machine path.

Node-red Flow



Detailed-1:



- Initialization: at beginning of the deploy dictates the readCSV to read the file
- readCSV: read the input file challenge3.csv
- csv: parser the input csv file
- csvFileCh3: see below

- reset: see below
-

Reset Function:

```
global.set("Status", "UNGOING");
global.set("ackCount", 0);
global.set("rowNum", 1)
global.set("msgNum", 0);
global.set("rowNumTemp", 1);
```

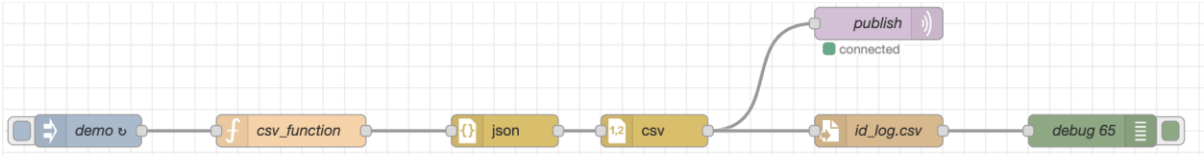
This function is used to initialize the global variables required for operation.

CsvFileCh3:

```
var dataArray = msg.payload;
global.set("myData",dataArray);

return msg;
```

This stores CSV data from msg.payload into "myData".

Detailde-2:



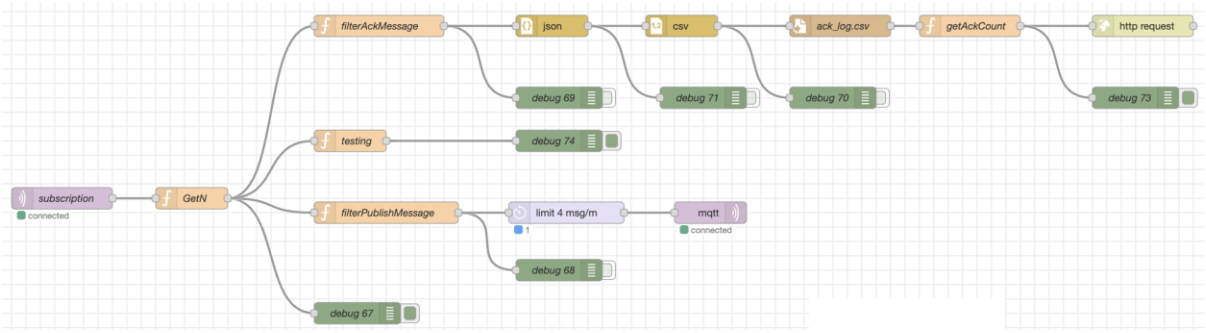
- Demo: after injected the node “initialization”, repeatedly send with a rate of 1 message every 5 seconds.
- Csv_function: see below
- Id_log.csv: read/write the input file id_log.csv
- Publish: publish the various Publish Messages Payload to the topic “challenge/id_generator”

Csv_function

```
let rowNum = global.get("rowNum") || 1;

const id = Math.floor(Math.random() * 50001);
const timestamp = Math.floor(Date.now() / 1000);

msg.payload = JSON.stringify({
    "No.": rowNum,
    "ID": id,
    "TIMESTAMP": timestamp
});

rowNum++;
global.set("rowNum", rowNum);

return msg;
```

Each message randomly generates number (id) between 0 and 50000, and the time in generation with format UNIX timestamp, and the row number incrementally

Detailed-3:



- Subscription: Subscribed to the topic challenge3/id_generator in the local broker (localhost, port 1884)
- GetN: see below.
- FileAckMessage: see below.
- Ack_log.csv: read/write the input file ack_log.csv
- GetAckCount: see below.
- Http request: SEND the value of the global ACK counter to thingspeak channel through HTTP API
- Testing: (not necessary) a counter for monitoring how many messages received
- FilterPublishMessage: see below
- Mqtt: publish the various Publish Messages Payload to the respectively topics dynamically
-

GetN

```
let msgNum = global.get("msgNum") || 0;

if(msgNum >= 80){
    global.set("Status","DONE");
    return null;
}

var parts = msg.payload.split(',');
var id = parts[1];
var dataArray = global.get("myData");
var desiredID =id % 7711;

var matchingObject = dataArray.filter(function(item){
    return parseInt(item["No."])===desiredID;
});
msg.payload = matchingObject;

msgNum++;
global.set("msgNum", msgNum);


return msg;
```

First of all,

it will check the status of flow if the node didn't receive more than 80 messages it will compute the N value, so basically it splits the string "msg.playload" by commas and storing the resulting parts in an array called "parts", after got the id value it goes to search the message with same ID in "myData" which contains the information of csv file.
msgNum is a counter to count the number of receiving messages.

FileAckMessage

```
if(global.get("Status")==="DONE"){
    return;
}
var dataArray = global.get("myData");
var ackCount = global.get("ackCount") || 0;
var matchingObject = msg.payload;
var noValue = msg.payload[0]["No."];

if(matchingObject && matchingObject[0].Info.includes("Ack")){
    global.set("matchObj",matchingObject);
    var ackRegex = /(Connect|Publish|Subscribe|Unsubscribe)\sAck/;
    var msgTypeMatch = matchingObject[0].Info.match(ackRegex);
    var msgType = msgTypeMatch ? msgTypeMatch[0] : null;
    msg.payload = JSON.stringify({
        "TIMESTAMP": Math.floor(Date.now() / 1000),
        "SUB_ID":noValue,
        "MSG_TYPE":msgType
    });
    global.set("ackCount", ackCount +1);
    return msg;
}else{
    global.set("matchObj",null);
    return null;
}
```

This function checks if a global variable named "Status" is set to "DONE". If it is, the function stops execution. Otherwise, it gets data from another global variable named "myData". It also gets the current value of the global variable "ackCount" or defaults to 0 if it's not set. Then, it examines the payload received in the message. If the payload contains an object and the value of its "Info" property includes the string "Ack", it sets another global variable named "matchObj" to the received payload. It then uses a regular expression to extract the type of the acknowledgment message from the "Info" property.

Next, it creates a JSON object with the current timestamp, message ID, and acknowledgment message type. This JSON object is added to the message payload.
Finally, it increments the "ackCount" variable by 1 and returns the modified message. If the payload doesn't have an acknowledgment message, it sets "matchObj" to null and returns null.

GetAckCount

```
msg.payload = global.get("ackCount");
return msg;
```

The code updates the payload with the value "ackCount".

FilterPublishMessage

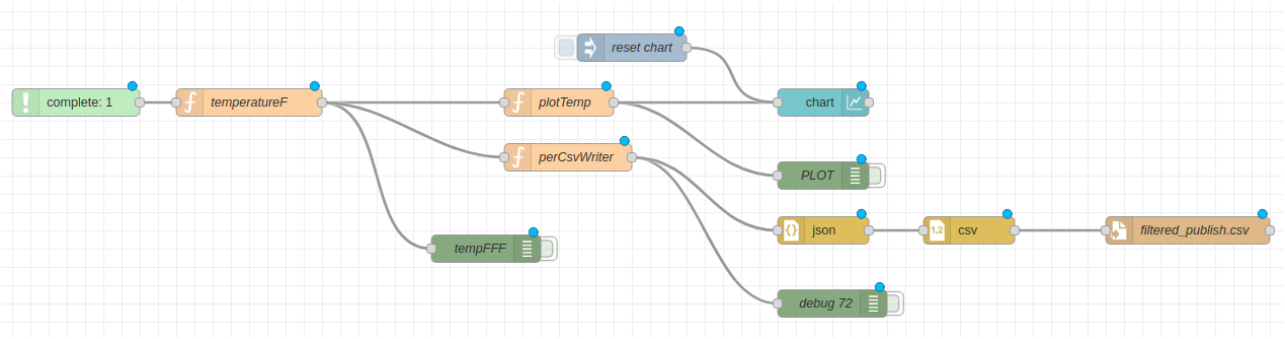
```
if(global.get("Status")==="DONE"){
    return;
}
var matchingObject = msg.payload;
var dataArray = global.get("myData");
var noValue = msg.payload[0]["No."];

//version 1.1
if(matchingObject && matchingObject[0].Info.includes("Publish Message")){

    global.set("matchObj",matchingObject);
    var topics = matchingObject[0].Info.match(/\[([^)]+)\]/g);
   var payload = matchingObject[0].Payload||"";

   topics.forEach(function(topic) {
        var publishMsg = {
            payload: {
                "timestamp": new Date().toISOString(),
                "id": noValue,
                "payload": payload
            },
            topic: topic
        };

        // Sending the publish message to the next node
        node.send(publishMsg);
    });
}else{
    global.set("matchObj",null);
    return null;
}
```

It will begin by checking the status of the flow. Then, it will search for the object based on the "No." extracted from the message payload.

If the matched message is a publish message, it will be stored in a global variable for further use.

If the message contains multiple topics, it will iterate through them using `forEach` and use `node.send()` to send them separately.

Detaild-4:



- Complete: it will be monitoring the mqtt node.
- TemperatureF: see below.
- PlotTemperatureFilter:see below.
- PlotTemp: see below.
- PerCsvWriter: see below.
- Filtered_publish.csv: read/write the input file filterd_publish.csv

TemperatureF

```
var matchingObject = global.get("matchObj");
var payload;
if(matchingObject){
    if (matchingObject[0].Payload == null) {
    payload = "";
    } else {
        payload = matchingObject[0].Payload;
        var data = JSON.parse(payload);
        if (data.type === "temperature" && data.unit === "F") {
            msg.payload = matchingObject;
            return msg;
        } else{
            return null;
        }
    }
}else{
    return null;
}
```

After previous node found the matched message this node function will check if the Publish Message contains in the payload a temperature in Fahrenheit if it's true then this message will be transmitted to the next node for calculation of its value which is going to plot on the chart

PlotTemp

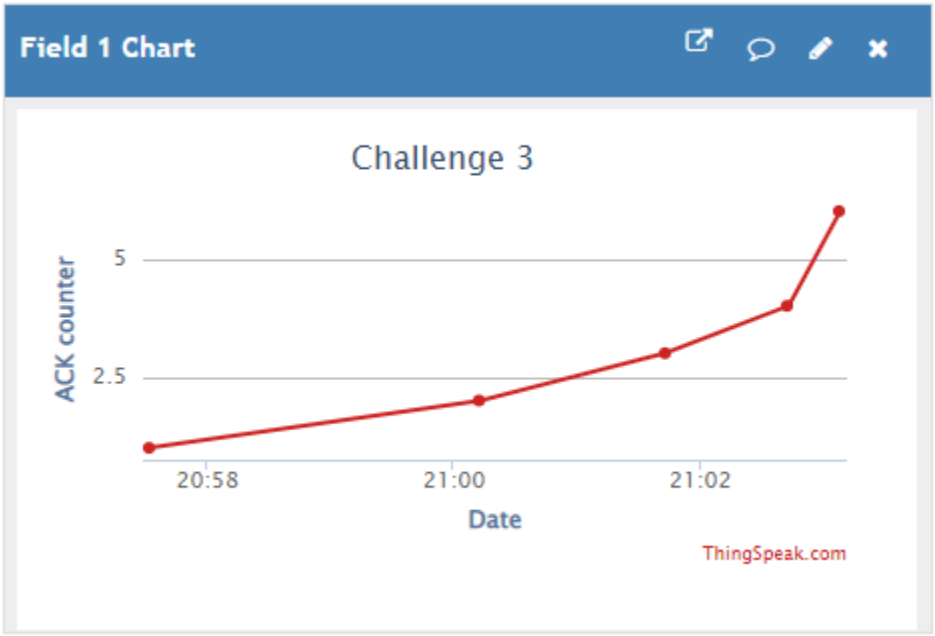
```
var dataObj=msg.payload;
var data = JSON.parse(dataObj[0].Payload);

if(data){
    let tempRange = data.range;
    let tempValue=(tempRange[0]+tempRange[1])/2;
    msg.payload=tempValue;
    return msg;

}else{
    return;
}
```

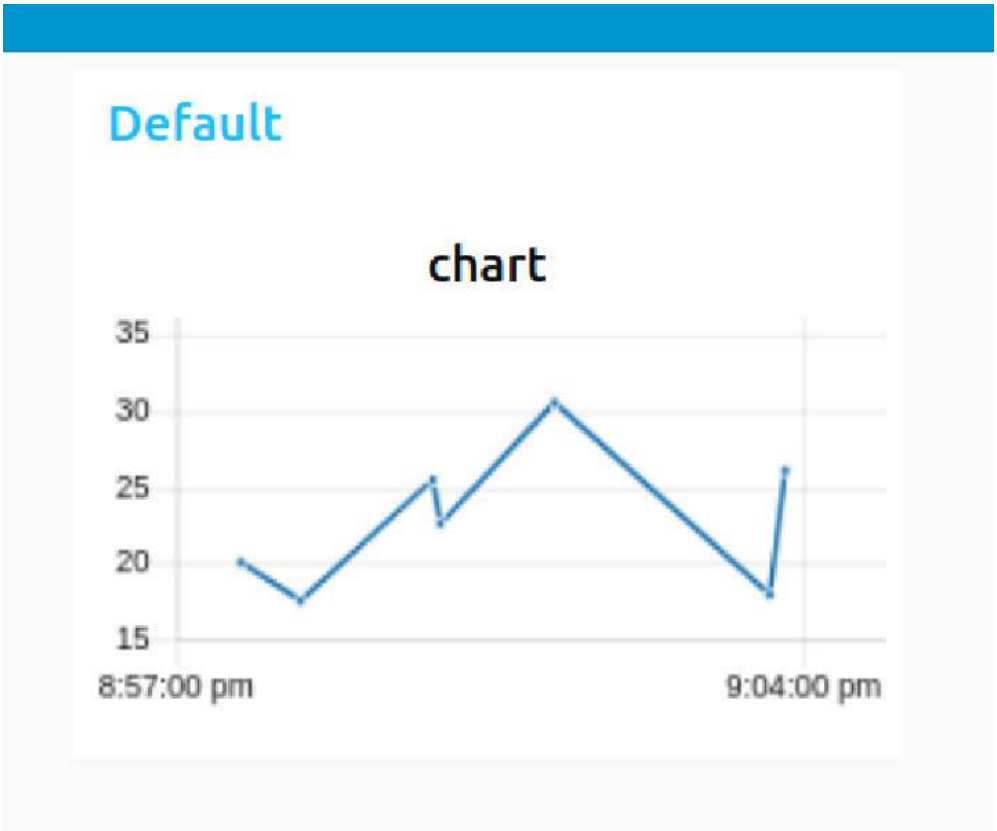
This code extracts and parses JSON data from a message. It checks if the data exists and calculates the average value if it does. The calculated value is then added to the message and returned.

PerCsvWriter

```
let rowNumTemp = flow.get("rowNumTemp") || 1;
var dataObj=msg.payload;
var data = JSON.parse(dataObj[0].Payload);

var long = data.long;
var range = data.range;
var lat = data.lat;
var type=data.type;
var unit = data.unit;
var description = data.description;

msg.payload = JSON.stringify({
    "No.": rowNumTemp,
    "LONG": long,
    "RANGE": range,
    "LAT":lat,
    "TYPE":type,
    "UNIT":unit,
    "DESCRIPTION":description,

});

rowNumTemp++;
global.set("rowNumTemp", rowNumTemp);

return msg;
```

This code retrieves the current value of the "rowNumTemp" global variable or defaults to 1 if it's not set. Then it parses the payload data into a JSON object. Next, it extracts specific properties from the JSON object and assigns them to variables. After that, it constructs a new JSON object containing the extracted properties along with an incremented row number. Finally, it updates the "rowNumTemp" variable and returns the modified message.


Field 1 Chart
Challenge 3
ACK counter
5
2.5
20:58
21:00
21:02
Date
ThingSpeak.com


Default
chart
35
30
25
20
15
8:57:00 pm
9:04:00 pm