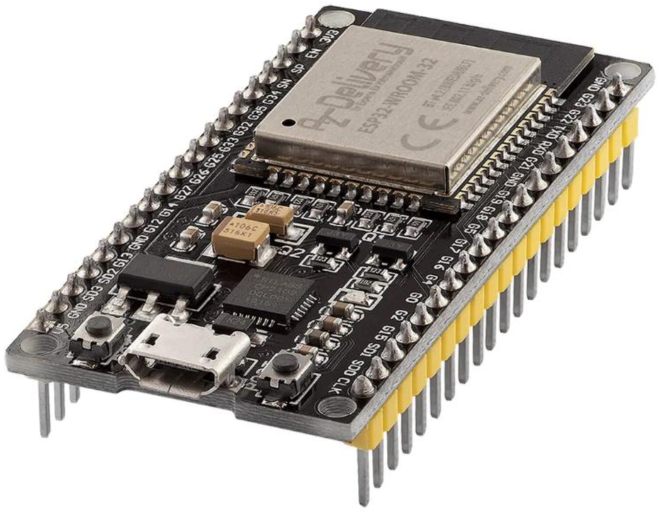
Weather Data with ESP32

American University of Beirut

EECE 425-Embedded and IoT Systems

Fall 2022



Group Members:

Hadil Abdel Samad

Andrew Bejjani

Presented to:

Dr. Mazen Saghir

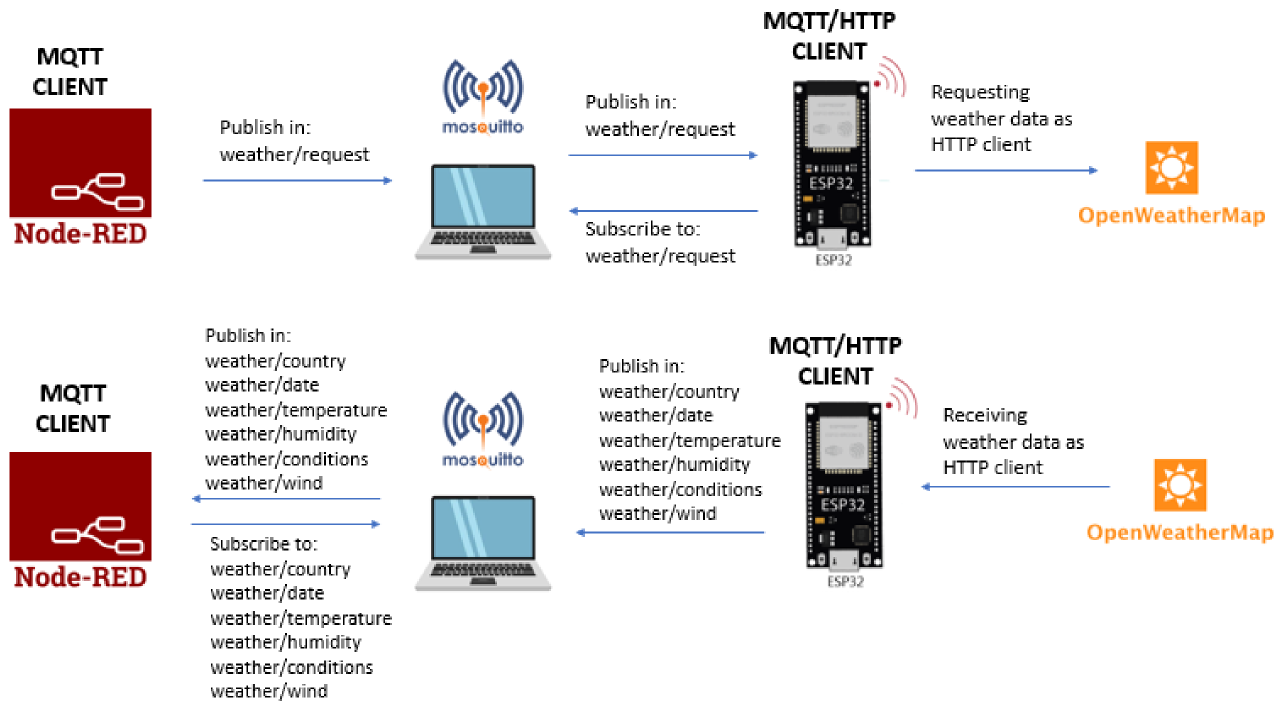
December 7, 2022

Table of Contents

Table of Figures

I. Introduction

In this project, we will be obtaining weather data from the internet for a certain geographical location using the ESP 32. This ESP 32 will be utilized as an HTTP client to receive the weather data and an MQTT client that will publish the data under certain topics. This will be done using a Mosquitto MQTT broker and Node-RED platform that will allow communication between a user interface and the ESP 32



MQTT CLIENT
Node-RED
Publish in: weather/request
mosquitto
MQTT/HTTP CLIENT
Publish in: weather/request
Subscribe to: weather/request
ESP32
Requesting weather data as HTTP client
OpenWeatherMap
MQTT CLIENT
Node-RED
Publish in: weather/country weather/date weather/temperature weather/humidity weather/conditions weather/wind
Subscribe to: weather/country weather/date weather/temperature weather/humidity weather/conditions weather/wind
mosquitto
Publish in: weather/country weather/date weather/temperature weather/humidity weather/conditions weather/wind
MQTT/HTTP CLIENT
ESP32
Receiving weather data as HTTP client
OpenWeatherMap

Figure 1: System-Level Architecture Diagram

As an overview, we'll be using Node-RED platform to request data from the MQTT broker that will forward this request to the ESP 32 which will act as an HTTP client to retrieve data. Once data is retrieved, it will act as an MQTT client to publish the data to certain topics as specified in Figure 1 which will allow the Node-RED platform to receive this data.

II. HTTP Client

For us to obtain the weather data, we will generate a certain API from https://home.openweathermap.org. This will allow us to request weather information for a certain city in a specific country based on longitude and latitude. In our case, we will be gathering weather data form Beirut, Lebanon through http://api.openweathermap.org/data/2.5/weather?q=Beirut,LB&units=metric&appid=d0afc4eefe171b18b64c30ee88867366. This link contains weather information formatted in JSON such as temperature, humidity, wind information and date/time of last measurement. We are interested in extracting the city name, country, date and time, temperature, humidity, weather description, and wind conditions. Once we extracted the JSON file, we realized that it is delivered as two packets of data. We want to save it in one buffer, so we use the strncat() function which concatenates the content of the two packets in one buffer of size 1024 for the data to completely fit. In some cases, the wind gust might be available which increases the size of the data to more than 12 characters. In our code, the HTTP_EVENT_ON_DATA case is called twice. Every time it's called, we concatenate the data in this packet to a buffer, hence, instead of printing two packets, we will just print the data once through this buffer. To extract the needed data from this buffer, we used the <cJSON.h> library to be able to parse our data to access each object as a string. We used the cJSON_GetObjectItem() function after defining the root using cJSON_Parse(), to get each object as a string or integer. We created a function that contains the data conversion, for each of the needed conditions and we called them in the http_init().

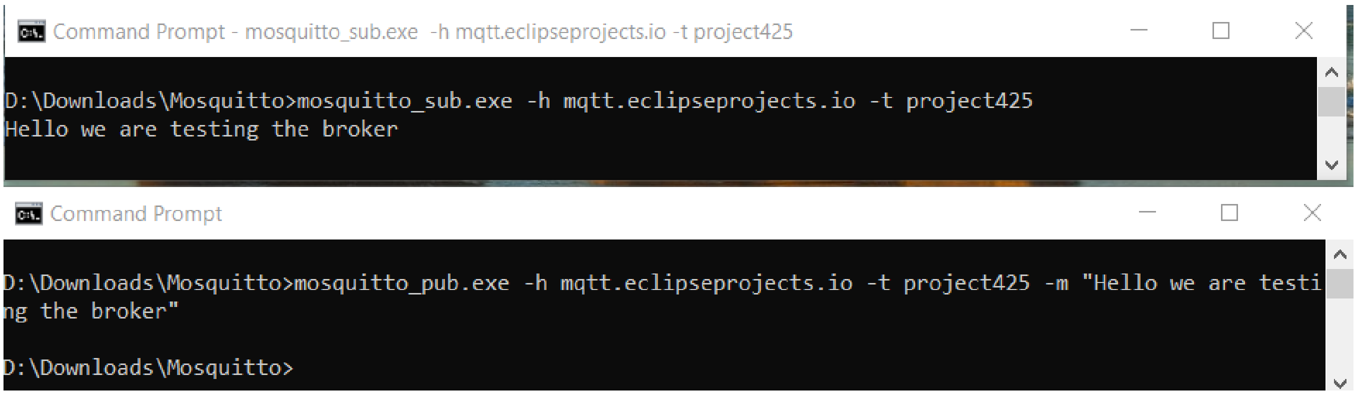
The weatherConditions() function extracts the city and country from the JSON file and stores them in a buffer "loc". The dt() function extracts the timezone, and dt from the JSON file and convert them to an appropriate format that includes the day, month, year, and 12-hour time format. The data is stored in a buffer "date". The temp() and humidity() functions extract the temperature which has a double value and humidity which has an integer value. The temperature is stored in a "temperature" buffer and the humidity is stored in a "hum" buffer. The conditions() function extracts the weather description from the file; however, this data is contained in an object stored in an array. To access the description, we first access the first element in the array and then we get the description which is stored in a "cond" buffer. The wind_direction() gets the wind speed, direction and the gust if found. We extracted the wind speed in m/s, and we converted it to km/hr by multiplying it by 3.6. Now to get the direction, we divide the degree by 45 which will shrink our number to a range between 0 and 8, and then we rounded it. For example, if our number is 1.6, this will allow us to say that the direction is closer to the East than to the Northeast, where East represents a degree of 2 and Northeast represents a degree of 1. For this reason, we rounded the number. Having the number ranged between 0 and 8 will allow us to choose between "N", "NE", "E", "SE", "S", "SW", "W", or "NW". Regarding the wind gust, it might not always be available, hence we checked the size of the wind object, if it was 3 then the gust is present and we proceeded on extracting its data, if not we neglected it. Once we gather the data, we store them in a "wind_data" buffer.

III. MQTT Client

After gathering the data from the HTTP client, we want to publish it to certain topics. However, we will only do so once we receive a request from the user. Therefore, we need to subscribe to a topic where the user will send a request to read, and we publish to other topics where we will transmit this data. Once we connect to the MQTT broker, we will subscribe to a topic called "weather/request" where we will wait for the user’s request. Once the user sends a request, we will visit the case MQTT_EVENT_DATA and publish our data on six different topics. Our choice of six topics will become clear once we describe the user interface. We have previously stored our data in 6 buffers. We will publish our “loc” buffer on topic “weather/country”, the “date” buffer on topic “weather/date”, the “temperature” buffer on topic “weather/temperature”, the “hum” buffer on topic “weather/humidity”, the “cond” buffer on topic “weather/conditions”, and the “wind_data” buffer on topic “weather/wind”.

IV. Mosquitto Broker

We need to install the mosquito broker for us to communicate with the Node-RED client. First, we install it from https://mosquitto.org/. Once installed, we open the command prompt and launch the mosquito. Once launched, we try to test publishing and subscribing to an arbitrary topic. We chose a topic called “project425” and we chose our broker to be “mqtt.eclipseprojects.io” which is widely used in the ESP IDF documentation. Once our broker and topic are ready, we launched the mosquitto_sub.exe application and we subscribed to the chosen broker and topic. Once subscribed, we published a message to the same topic and checked if the message was displayed on the subscribed command prompt.


```
Command Prompt - mosquitto_sub.exe  -h mqtt.eclipseprojects.io -t project425

D:\Downloads\Mosquitto>mosquitto_sub.exe -h mqtt.eclipseprojects.io -t project425
Hello we are testing the broker
```

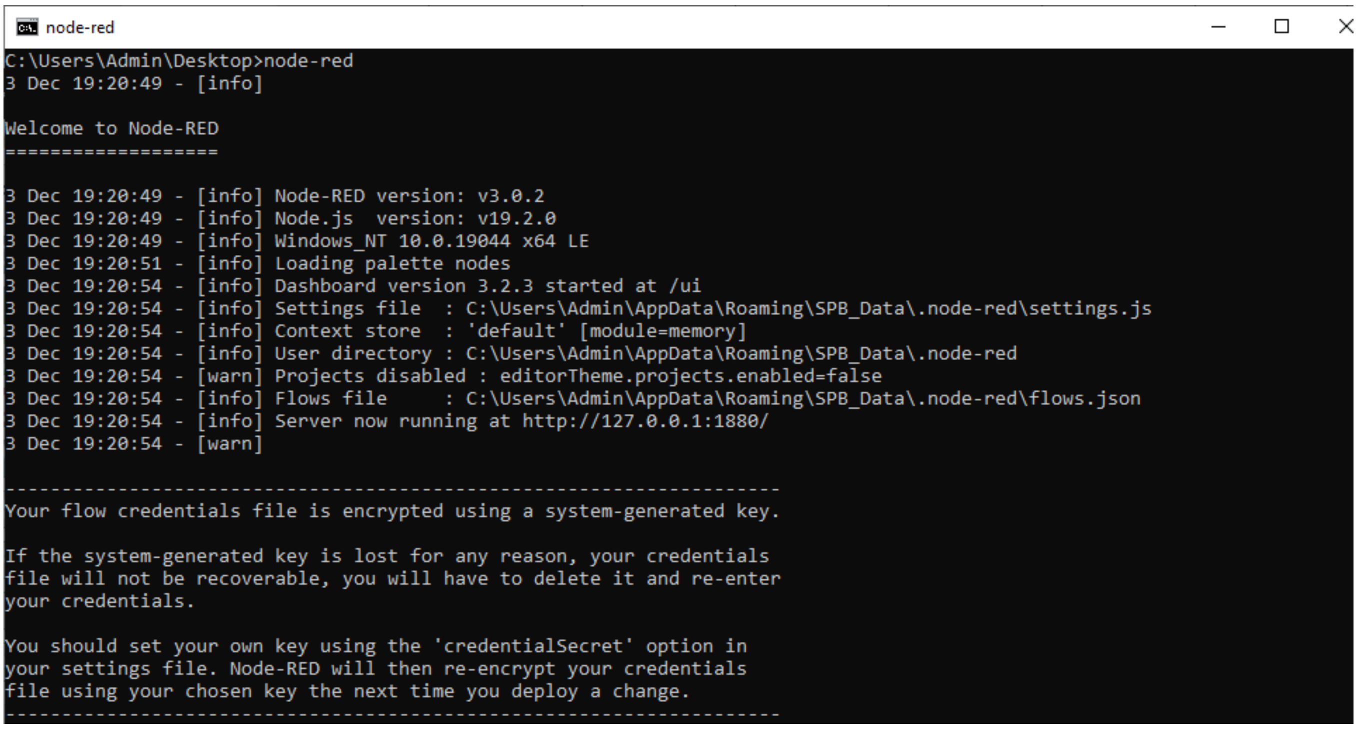
```
Command Prompt

D:\Downloads\Mosquitto>mosquitto_pub.exe -h mqtt.eclipseprojects.io -t project425 -m "Hello we are testi
ng the broker"

D:\Downloads\Mosquitto>
```

Figure 2: Testing the Mosquitto Broker

As we can see in Figure 2, the message "Hello we are testing the broker" was successfully published and read.

V. Node-RED

For Node-RED, our goal is to install it locally. First, we install Node.js by typing node "--version && npm –version" in the command prompt. After that, we installed Node-RED after executing in the command prompt the following code "npm install -g --unsafe-perm node-red". Once installed, we are now ready to run Node-RED. We do that by typing "node-red" into the location where it was installed.



```
node-red
C:\Users\Admin\Desktop>node-red
3 Dec 19:20:49 - [info]

Welcome to Node-RED
===================

3 Dec 19:20:49 - [info] Node-RED version: v3.0.2
3 Dec 19:20:49 - [info] Node.js  version: v19.2.0
3 Dec 19:20:49 - [info] Windows_NT 10.0.19044 x64 LE
3 Dec 19:20:51 - [info] Loading palette nodes
3 Dec 19:20:54 - [info] Dashboard version 3.2.3 started at /ui
3 Dec 19:20:54 - [info] Settings file  : C:\Users\Admin\AppData\Roaming\SPB_Data\.node-red\settings.js
3 Dec 19:20:54 - [info] Context store  : 'default' [module=memory]
3 Dec 19:20:54 - [info] User directory : C:\Users\Admin\AppData\Roaming\SPB_Data\.node-red
3 Dec 19:20:54 - [warn] Projects disabled : editorTheme.projects.enabled=false
3 Dec 19:20:54 - [info] Flows file     : C:\Users\Admin\AppData\Roaming\SPB_Data\.node-red\flows.json
3 Dec 19:20:54 - [info] Server now running at http://127.0.0.1:1880/
3 Dec 19:20:54 - [warn]

---------------------------------------------------------------------
Your flow credentials file is encrypted using a system-generated key.

If the system-generated key is lost for any reason, your credentials
file will not be recoverable, you will have to delete it and re-enter
your credentials.

You should set your own key using the 'credentialSecret' option in
your settings file. Node-RED will then re-encrypt your credentials
file using your chosen key the next time you deploy a change.
---------------------------------------------------------------------
```

Figure 3: Node-RED Command Prompt Setup

In our case, we typed C:\Users\Admin\Desktop>node-red as shown in Figure 3.

Once we execute that, we can open in our browser the link http://127.0.0.1:1880/ with 127.0.0.1 being the localhost and 1880 being the port number. When opened, the Node-RED will be available for flow design. In our case, since we needed a user interface, we had to install the dashboard in Node-RED by using the "manage palette". After completing the installation of the Dashboard, we can now use the button and the text nodes. We started then with the design of the flow. First, we utilized a button node, which controls the MQTT_PUB node to publish a message on the topic "weather/request" for which the ESP 32 is subscribed to. Once received, the ESP 32 will publish on six different topics the requested information for which we have six nodes, each labeled by the name of the topic it is subscribed to. To display the specific data, we used four text nodes that display text messages for the location, date and time, weather conditions, and wind conditions. We also used two gauge nodes to display numerical values for the temperature and humidity in a user friendly manner.

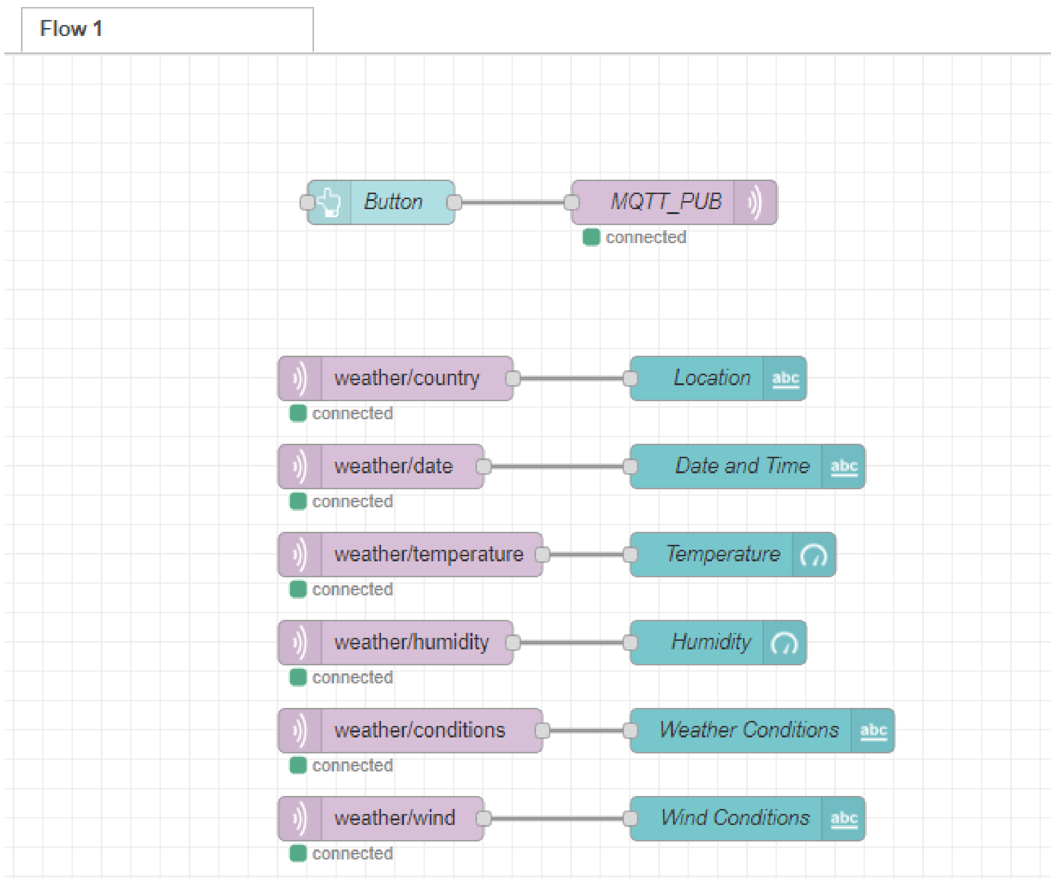
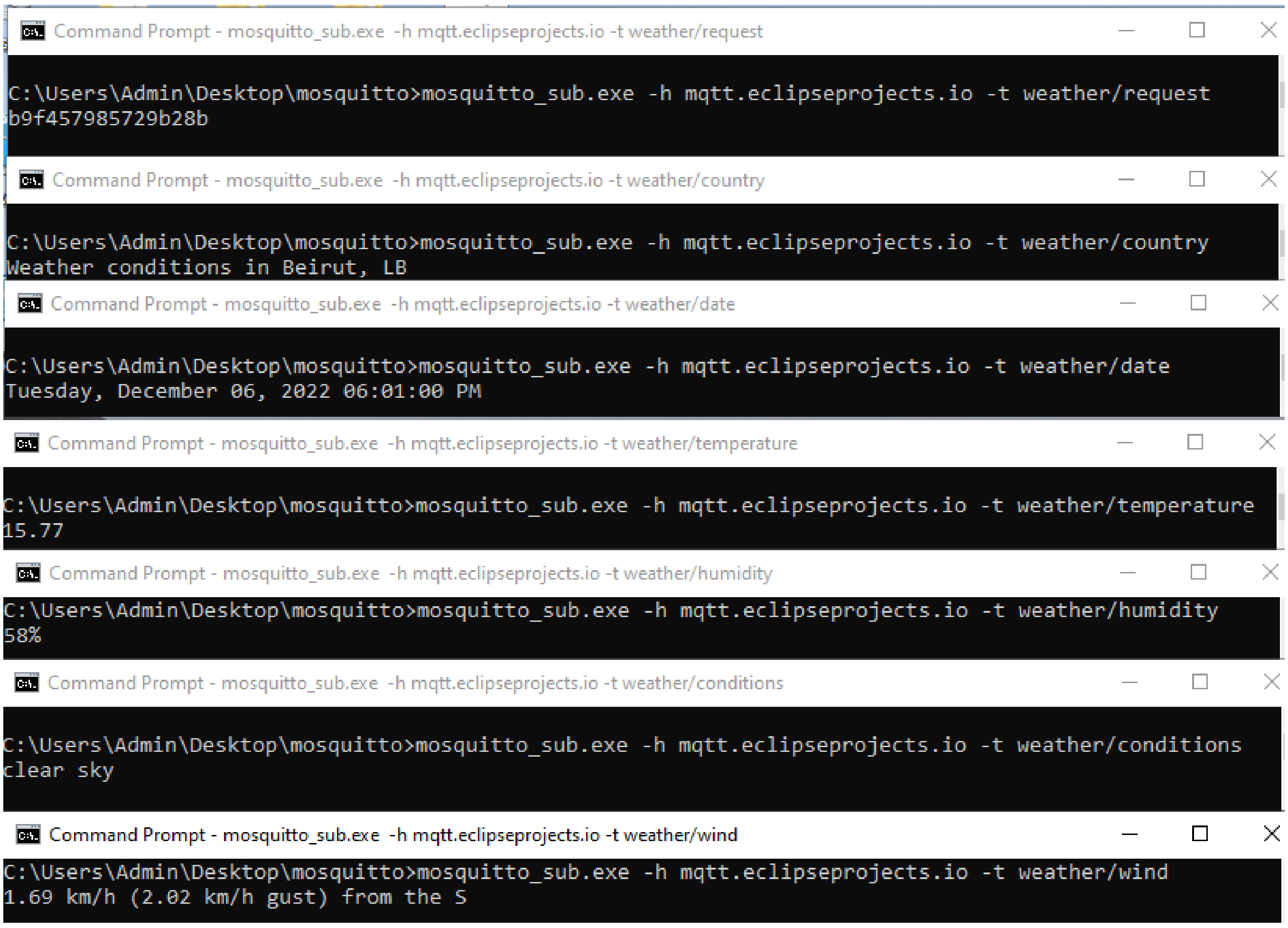


Figure 4: Node-RED Flow Chart

Our final result will be a connected flow chart as shown in Figure 4.

VI. Connected Implementation

Being done with the flow, we opened the user interface by typing http://127.0.0.1:1880/ui in our browser. Once we press the button, this will trigger a series of events. First, the Node-RED will send a message to the topic “weather/request”. Once the MQTT client receives this message, it will trigger the HTTP client to fetch the appropriate weather information. This information will be stored in six different buffers for six different topics. Once the data is fetched, the MQTT client will publish the data gathered under the six topics “weather/country”, “weather/date”, “weather/temperature”, “weather/humidity”, “weather/conditions”, and “weather/wind”.


Command Prompt - mosquitto_sub.exe -h mqtt.eclipseprojects.io -t weather/request

```
C:\Users\Admin\Desktop\mosquitto>mosquitto_sub.exe -h mqtt.eclipseprojects.io -t weather/request
b9f457985729b28b
```

Command Prompt - mosquitto_sub.exe -h mqtt.eclipseprojects.io -t weather/country

```
C:\Users\Admin\Desktop\mosquitto>mosquitto_sub.exe -h mqtt.eclipseprojects.io -t weather/country
Weather conditions in Beirut, LB
```

Command Prompt - mosquitto_sub.exe -h mqtt.eclipseprojects.io -t weather/date

```
C:\Users\Admin\Desktop\mosquitto>mosquitto_sub.exe -h mqtt.eclipseprojects.io -t weather/date
Tuesday, December 06, 2022 06:01:00 PM
```

Command Prompt - mosquitto_sub.exe -h mqtt.eclipseprojects.io -t weather/temperature

```
C:\Users\Admin\Desktop\mosquitto>mosquitto_sub.exe -h mqtt.eclipseprojects.io -t weather/temperature
15.77
```

Command Prompt - mosquitto_sub.exe -h mqtt.eclipseprojects.io -t weather/humidity

```
C:\Users\Admin\Desktop\mosquitto>mosquitto_sub.exe -h mqtt.eclipseprojects.io -t weather/humidity
58%
```

Command Prompt - mosquitto_sub.exe -h mqtt.eclipseprojects.io -t weather/conditions

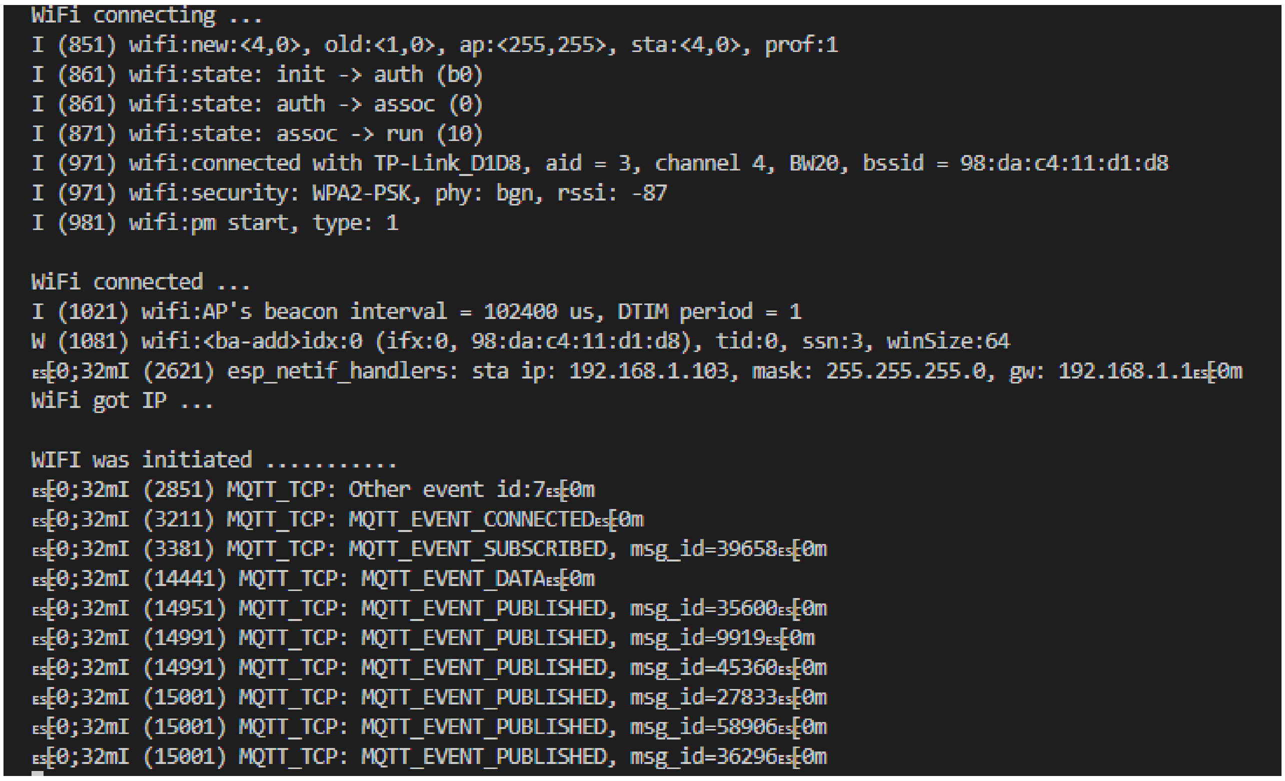
```
C:\Users\Admin\Desktop\mosquitto>mosquitto_sub.exe -h mqtt.eclipseprojects.io -t weather/conditions
clear sky
```

Command Prompt - mosquitto_sub.exe -h mqtt.eclipseprojects.io -t weather/wind

```
C:\Users\Admin\Desktop\mosquitto>mosquitto_sub.exe -h mqtt.eclipseprojects.io -t weather/wind
1.69 km/h (2.02 km/h gust) from the S
```

Figure 5: Subscribing to the topics where data will be displayed

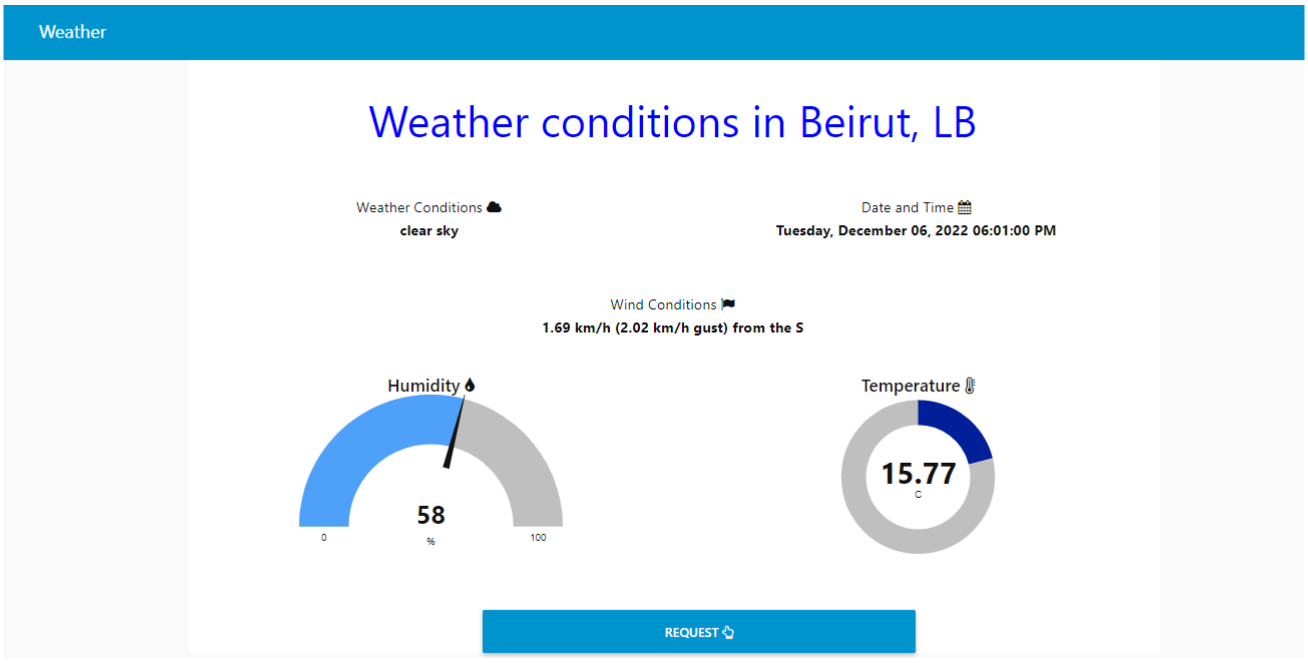
We can see in figure 5 the data that is sent on the "weather/request" topic, for which the user is requesting the weather data and the data that is sent on the six topics, where the user will receive the data. We notice that the data published under the topic "weather/request" is dummy since we know that we only have one button that the user is pressing. So once we receive data, regardless of the data, we will know that the user is requesting information. If multiple buttons were present, then we'll have to decode the actual message being sent from Node-RED, but this is not needed here. For further clarification, we chose to publish on six topics for the ease in designing the user interface which will be shown later on. The subscriptions we did here in the command prompt are just to visualize how the data we are sending is indeed correct and not to fall in any traps. This is supported in Figure 6 when we check the serial monitor for the ESP 32.



```
WiFi connecting ...
I (851) wifi:new:<4,0>, old:<1,0>, ap:<255,255>, sta:<4,0>, prof:1
I (861) wifi:state: init -> auth (b0)
I (861) wifi:state: auth -> assoc (0)
I (871) wifi:state: assoc -> run (10)
I (971) wifi:connected with TP-Link_D1D8, aid = 3, channel 4, BW20, bssid = 98:da:c4:11:d1:d8
I (971) wifi:security: WPA2-PSK, phy: bgn, rssi: -87
I (981) wifi:pm start, type: 1

WiFi connected ...
I (1021) wifi:AP's beacon interval = 102400 us, DTIM period = 1
W (1081) wifi:<ba-add>idx:0 (ifx:0, 98:da:c4:11:d1:d8), tid:0, ssn:3, winSize:64
ESC[0;32mI (2621) esp_netif_handlers: sta ip: 192.168.1.103, mask: 255.255.255.0, gw: 192.168.1.1ESC[0m
WiFi got IP ...

WIFI was initiated ...........
ESC[0;32mI (2851) MQTT_TCP: Other event id:7ESC[0m
ESC[0;32mI (3211) MQTT_TCP: MQTT_EVENT_CONNECTEDESC[0m
ESC[0;32mI (3381) MQTT_TCP: MQTT_EVENT_SUBSCRIBED, msg_id=39658ESC[0m
ESC[0;32mI (14441) MQTT_TCP: MQTT_EVENT_DATAESC[0m
ESC[0;32mI (14951) MQTT_TCP: MQTT_EVENT_PUBLISHED, msg_id=35600ESC[0m
ESC[0;32mI (14991) MQTT_TCP: MQTT_EVENT_PUBLISHED, msg_id=9919ESC[0m
ESC[0;32mI (14991) MQTT_TCP: MQTT_EVENT_PUBLISHED, msg_id=45360ESC[0m
ESC[0;32mI (15001) MQTT_TCP: MQTT_EVENT_PUBLISHED, msg_id=27833ESC[0m
ESC[0;32mI (15001) MQTT_TCP: MQTT_EVENT_PUBLISHED, msg_id=58906ESC[0m
ESC[0;32mI (15001) MQTT_TCP: MQTT_EVENT_PUBLISHED, msg_id=36296ESC[0m
```

Figure 6: ESP 32 Serial Monitor

We can distinguish how once we connect to the MQTT client, we visit the cases MQTT_EVENT_CONNECTED and MQTT_EVENT_SUBSCRIBED. Once the user presses the button on Node-RED, this will trigger the case MQTT_EVENT_DATA where the MQTT client will publish the data and this triggers MQTT_EVENT_PUBLISHED six times since we publish on six topics. As a final result, we will receive the data on the Node-RED UI as shown in Figure 7.



Weather

Weather conditions in Beirut, LB

Weather Conditions
clear sky

Date and Time
Tuesday, December 06, 2022 06:01:00 PM

Wind Conditions
1.69 km/h (2.02 km/h gust) from the S

Humidity
58 %

Temperature
15.77 C

Figure 7: User Interface on Node-RED

VII. Distribution of Work

The work was equally divided between both team members Andrew Bejjani and Hadil Abdel Samad. First of all, both members worked on combining both packets into one buffer and finding the appropriate JSON library to use to parse the data since this involved a lot of research. Once found, both members worked on the weatherConditions() function to test the library. Once it worked, Andrew worked on the dt() and conditions() functions and Hadil worked on the humidity(), temp(), and wind_direction() functions. Once that was finalized, Andrew worked on the mosquito broker to test how to publish and subscribe to arbitrary topics on the command prompt. He also took the responsibility of combining both of the HTTP and MQTT codes into one code and subscribing and publishing to certain topics. On the other hand, Hadil worked on setting up the Node-RED platform and correctly building a flow chart that subscribes and publishes to certain defined topics. She also worked on the user-interface and made sure that it properly displays the data correctly and neatly. As for the report, it was a mutual effort.

VIII. References

C library function - strftime(). n.d. tutorialspoint. <https://www.tutorialspoint.com/c_standard_library/c_function_strftime.htm>.

cJSON Library Functions. 9 January 2017. GitHub. <https://github.com/pycom/esp-idf-2.0/blob/master/components/json/include/cJSON.h>.

ESP32 MQTT Client Publish and Subscribe using ESP-IDF. y October 2022. ESP 32 ESP-IDF. <https://esp32tutorials.com/esp32-mqtt-client-publish-subscribe-esp-idf/>.

ESP-MQTT. n.d. Espressif. <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/protocols/mqtt.html>.

Steve. *Using The Mosquitto_pub and Mosquitto_sub MQTT Client Tools- Examples*. 12 May 2022. <http://www.steves-internet-guide.com/mosquitto_pub-sub-clients/>.

The Icons. n.d. Font Awesome 4. <https://fontawesome.com/v4/icons/>.