

READMe



*Final Project*

Charbel Chdid

Hussam al basha

Contents

[I- Quick demo: 2](#_Toc121330659)

[II- System-Level Architecture Diagram: 2](#_Toc121330660)

[III- Overview: 2](#_Toc121330661)

[IV- Setting up Mosquitto: 3](#_Toc121330662)

[V- Setting up Node-Red: 4](#_Toc121330663)

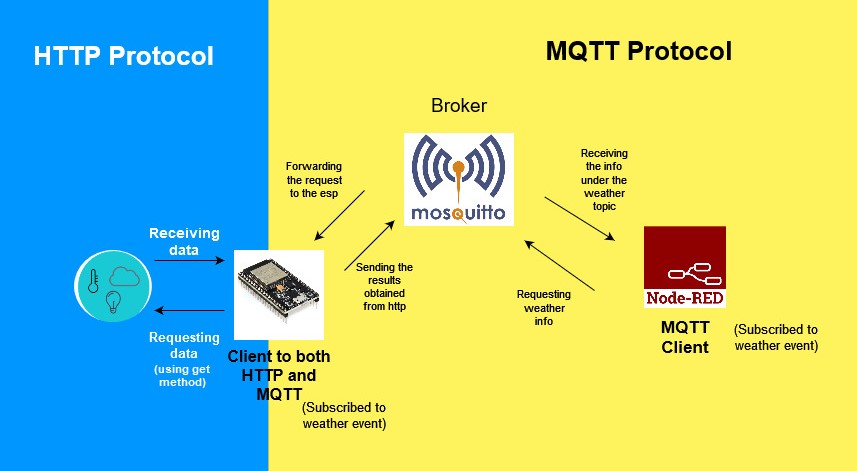
[VI- Setting up the ESP: 5](#_Toc121330664)

# Quick demo:

We did a 1 min demo of the user interface. Hussam posted it on his YouTube channel. You can watch it via the link below.

<https://youtu.be/M_sP4r040yw>

# System-Level Architecture Diagram:



# Overview:

The goal of the project is to display the real time weather and characteristics of any specific city. To do so, we have used the ESP 32 that will be running two protocols: **HTTP** and **MQTT**.

Role of HTTP: The data we are pursuing is stored on the internet and the best way to retrieve it is to use the HTTP protocol by which we can access the data from their URLs. For that, we will be extracting the information from “open weather map” website. The ESP 32 will be acting as client and, using the get method, will be able to read the data and return it as a JSON file. Having the data on hand, the best way to represent it to the user is using the MQTT protocol.

Role of MQTT: In order to process the data to the users, MQTT is the best solution since it offers processing data to user interfaces and it allows filtering the data that are being displayed using subscription protocol. MQTT needs a broker that plays the role of the handler. The broker is responsible for allowing clients to subscribe to different topics and to publish data in these topics. The clients send a message to the broker to subscribe to a certain topic and the broker approves it. Then, when a client publishes some data under a certain topic, all the clients subscribed to that topic will receive the data. In the project we used Node-Red software for the user interface and Mosquitto as the broker. Both the ESP32 and Node-Red are clients to the broker and are subscribed to the same topics. The user will request to get the information by choosing the country, the city and pressing a button. After that, Node-Red will send a message under the country choice topic choosing the country, the city choice topic choosing the city and the weather topic requesting the data. The broker forwards the message to the subscribers (in our case we only have the ESP32), and once the esp receives the message, it will get the info using http, sends them back to the broker under the topic weather, and finally the info will arrive to the Node-Red client and will be displayed to the user.

# Setting up Mosquitto:

As we mentioned earlier, Mosquitto software will play the role of the broker. To prepare mosquitto, we downloaded the software and used some syntax on the command prompt.

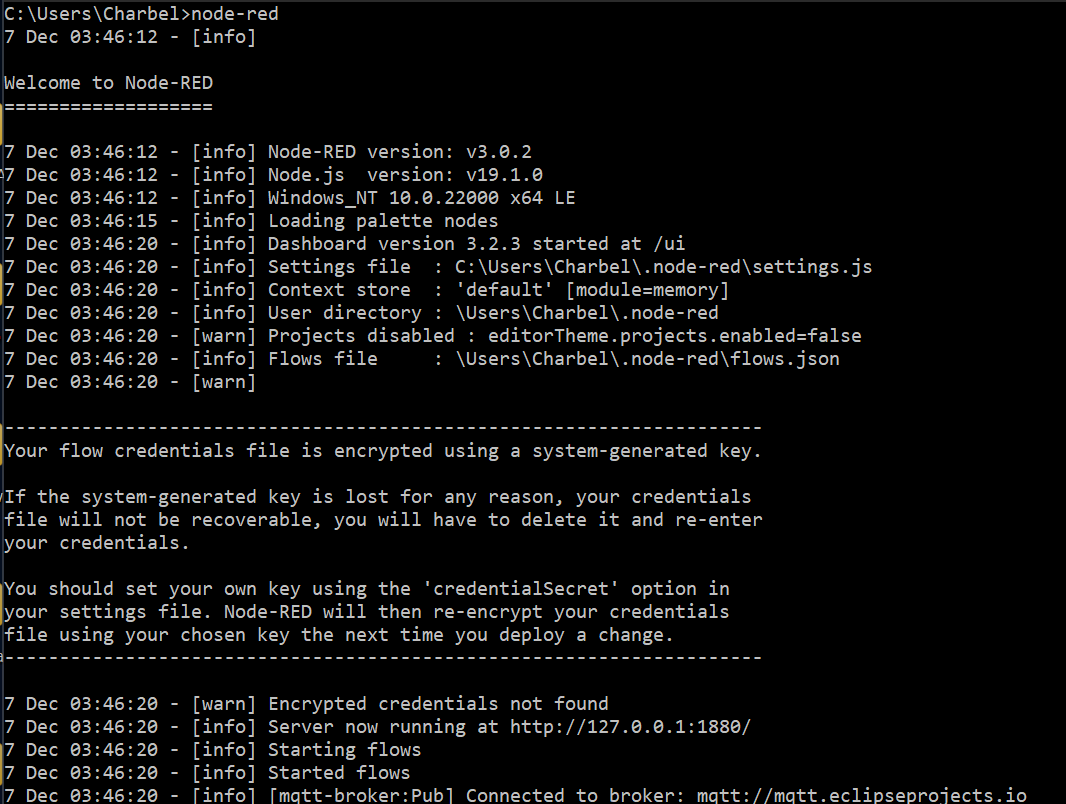
To launch mosquitto, we opened its directory and used mosquitto -v syntax as follows:A screenshot of a computer

Description automatically generated with medium confidence

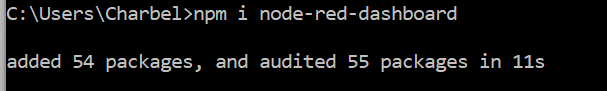
In order to track any message that has been sent under the weather topic, we subscribed to the broker under the same topic using the command prompt by entering the following syntax: mosquitto\_sub.exe -h mqtt.eclipseprojects.io -t weather.

Mosquitto is now ready.

# Setting up Node-Red:



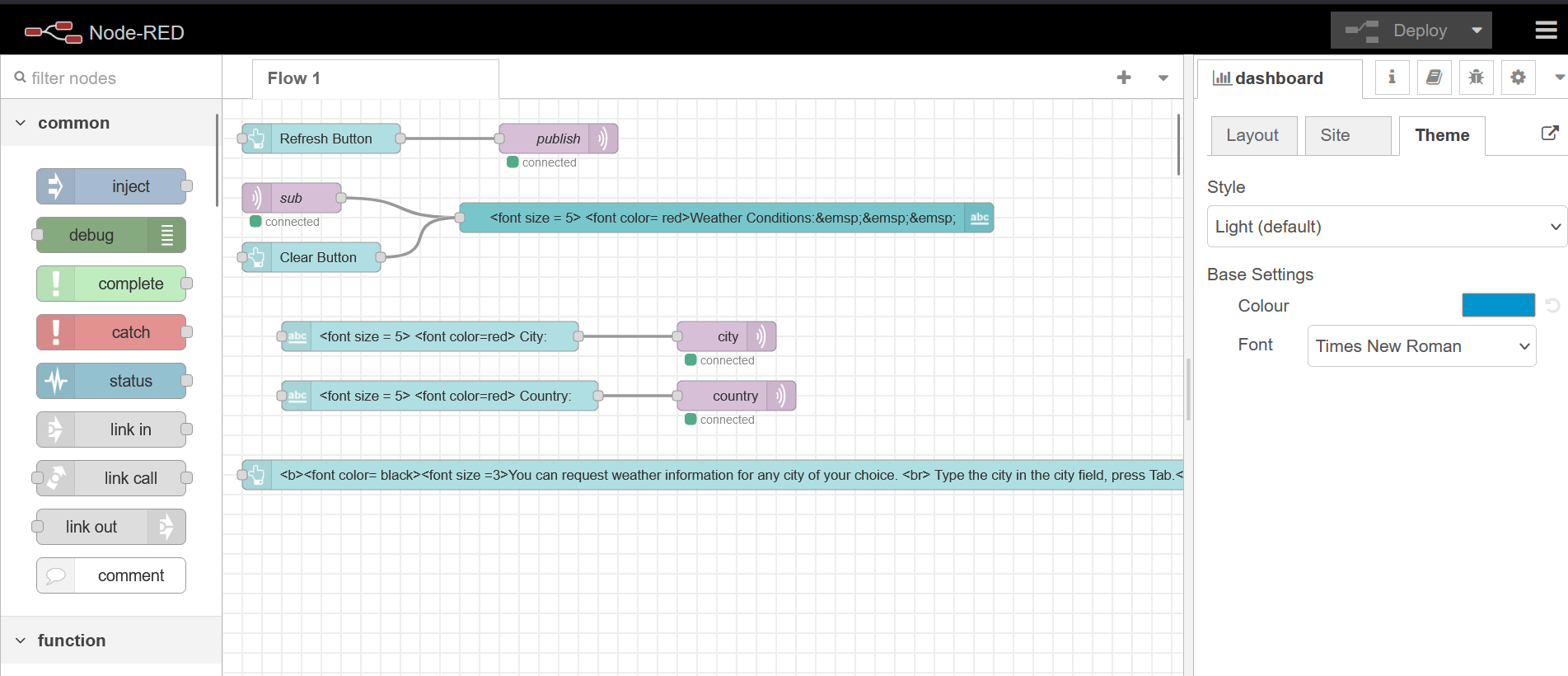
1. Launch:

Setting up Node-Red needs first downloading the software to use its commands on command prompts. After that Node-red can be simply launched using the syntax: node-red. In order to get more features for UI and to see the dashboard, we downloaded some packages:

Now to use Node-red and to setup the user interface, we need to copy the URL of the server and paste it in the browser. In this case our URL is: 127.0.0.1:1880/.

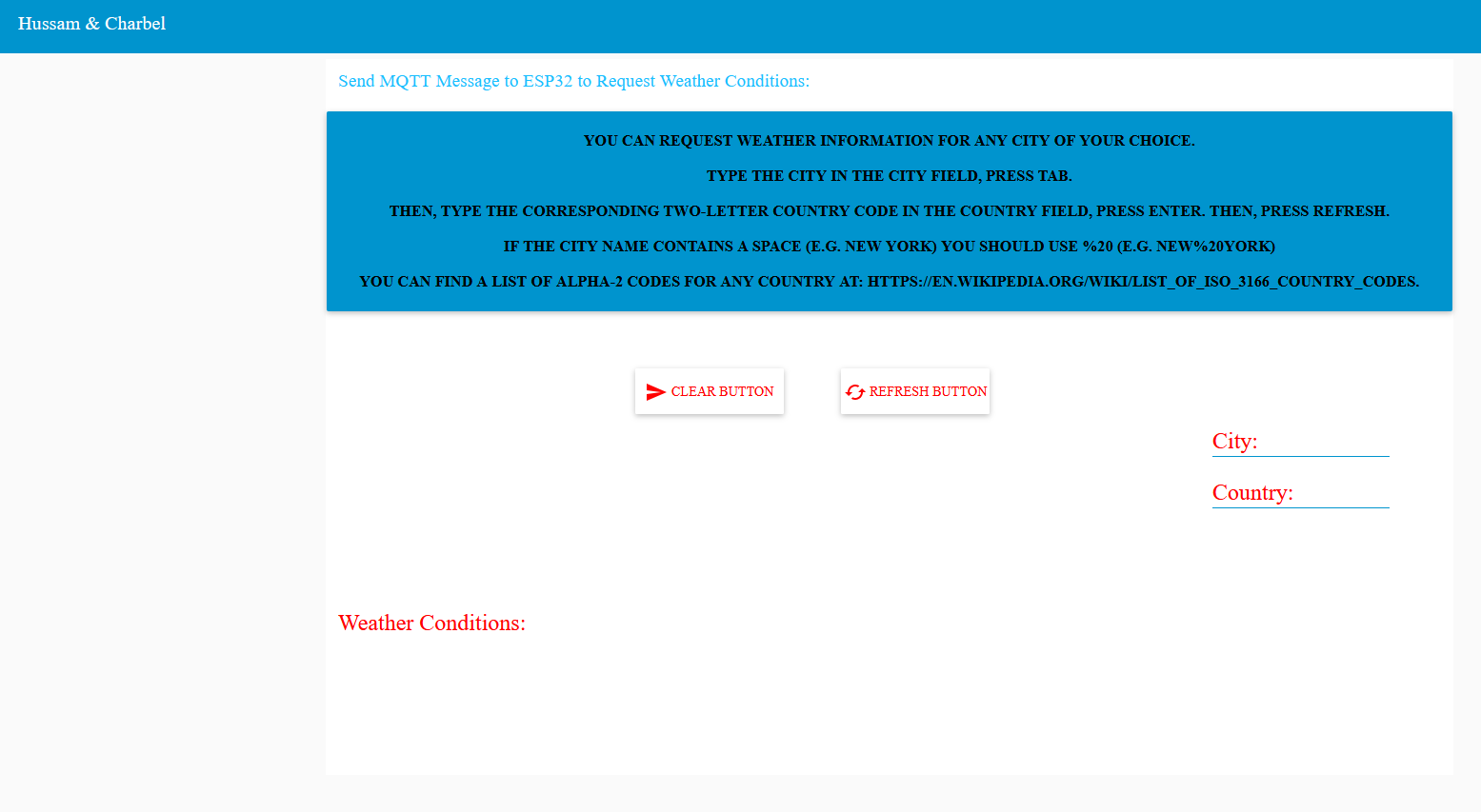
1. UI:

The figure below shows the components used for the user interface:



We first used two text boxes each connected to an MQTT out box. The user in these text boxes should choose a country and a city. Node-Red sends two messages to the broker under the topics country and city and then are processed to the esp. Then we used a refresh button connected to another MQTT out that will send a message to the ESP under the topic weather to publish the data and thus Node-Red will receive the data sent by the ESP from the broker under the same topic. Once received, it will be displayed in a text box. We designed the text format and added some icons from the message sent from the ESP.

Finally, we edited the UI design using the dashboard layout editor and that’s how it looked like without having any data:



To test if Node-Red is behaving as client, pressed the button from the UI and observed if the message was received in the command prompt where we subscribed to the broker and to the same topic as follows: 

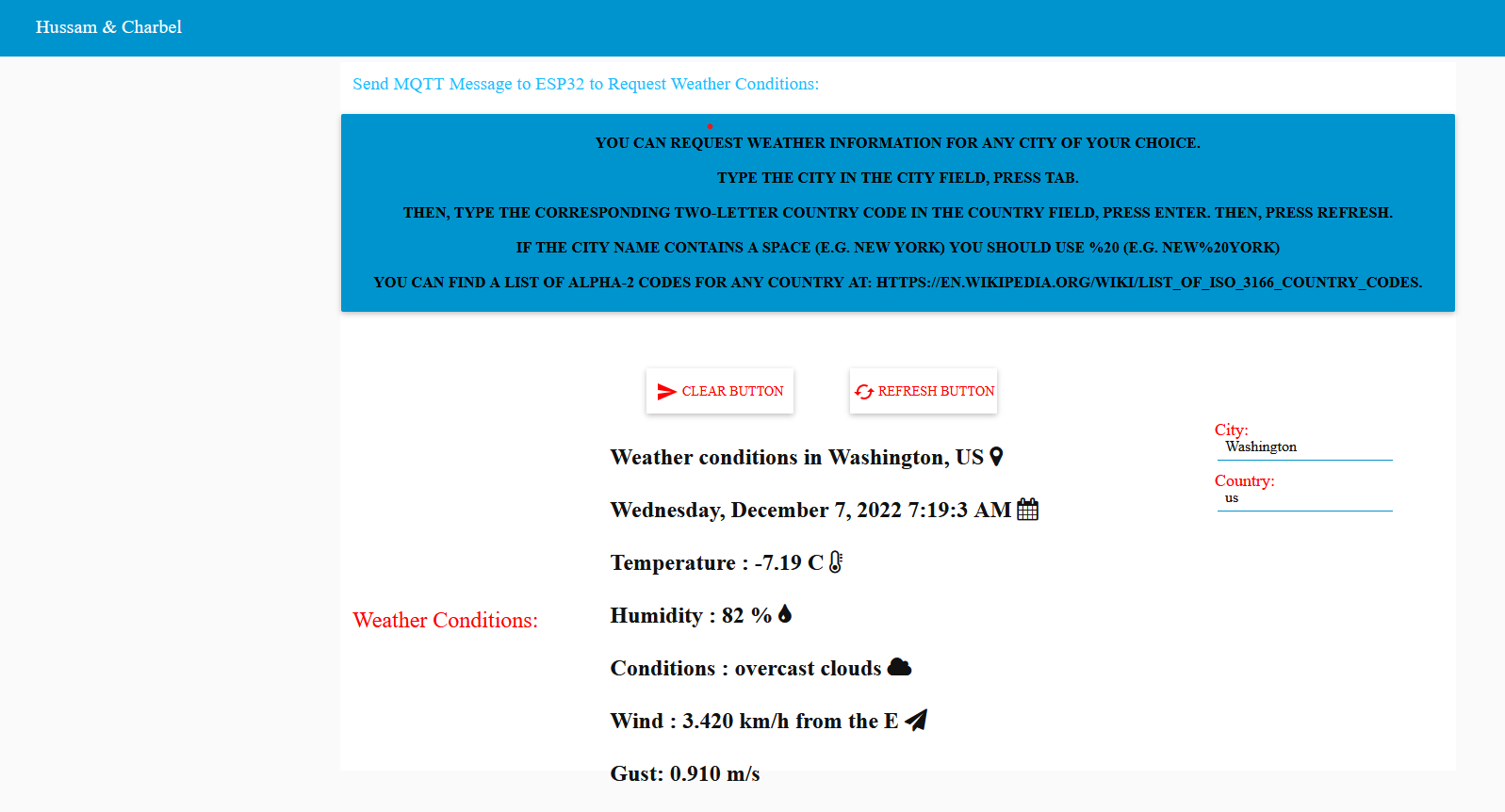
Node-Red is now ready.

# Setting up the ESP:

Concerning the ESP, we wrote a code that first, connects the ESP to a Wi-Fi, initializes MQTT and then waits for a message from the broker. Once a message has been received, the ESP will check under which topic the message has been sent. If the topic is weather, a function containing the http operation will be called. If the topic is either country or city, the ESP will call a function that will change the link from which we are extracting the data in the http protocol and will adjust it to the corresponding values.

In the HTTP protocol, we send a request to get the data from the weather website (as mentioned before). The data is received as a JSON string. In order to fetch the needed information from the string, we used the cJSON library that parses this string and copy its elements into a data structure that is similar to a hash-map. We took the needed info and combined them in a single string. Then, we edited this string to have a format that offers a good display in Node-Red, and we also included Icons. The esp will then publish them to the broker under the same topic received (weather).

Once we sent the message, we can directly see the data displayed on Node-Red with the format that we designed.

The picture below shows how we the user will receive the data: