EECE 425 Project Report

Mohammad El Sheikh and Taha Koleilat

December 7, 2022

1 System-level Architecture

Our developed application presents a Node-Red Client configured as a MQTT Client with a UI that features a Button which when pressed will publish to the MQTT topic **weather/get_weather**. The ESP32 acting as a MQTT Client will subscribe to the MQTT topic **weather/get_weather** and then will act as an HTTP Client to fetch current weather data of a particular region as a JSON object through a REST API that sends an HTTP Request to the OpenWeatherMap server. After retrieving the relevant data, the ESP32 formats the desired parameters into a comma-separated text string and will then publish to **weather/current_weather** where then the Node-Red Client will subscribe to the latter MQTT topic, format the comma-separated text into the needed representation and display in the Node-Red UI using HTML the final weather info along with a related weather icon. The MQTT Broker handles all topics published and subscribed by the MQTT Clients. The MQTT topic **weather/get_weather** specifies a request to the ESP32 to retrieve weather data, while the MQTT topic **weather/current_weather** specifies the extracted and parsed weather variables to be used and displayed in Node-Red. Fig. 1 depicts the overall interconnection of the main components in the system-level architecture.

2 ESP32 HTTP Client

There's 4 main functions in the HTTP Client.

1. wifi_event_handler which handles the different wifi event cases like connecting, losing connection, getting IP, and connection establishment.
2. wifi_connection which handles connecting to the WiFi Access Point (AP) by using the WiFi SSID and Password.
3. client_event_get_handler which handles the HTTP Request and stores the received packets into a char buffer where they are reassembled to be used later
4. rest_get which is the main function which performs the HTTP GET client operations by combining all the components together so that the HTTP Request JSON is received successfully and passed on to the MQTT Client

```
char http_response[1024];
static unsigned int http_index;

static void wifi_event_handler(void *event_handler_arg, esp_event_base_t event_base,
     int32_t event_id, void *event_data)
{
    switch (event_id)
    {
    case WIFI_EVENT_STA_START:
        printf("WiFi connecting ... \n");
        break;
    case WIFI_EVENT_STA_CONNECTED:
        printf("WiFi connected ... \n");
        break;
    case WIFI_EVENT_STA_DISCONNECTED:
        printf("WiFi lost connection ... \n");
        break;
    case IP_EVENT_STA_GOT_IP:
        printf("WiFi got IP ... \n\n");
        break;
    default:
        break;
    }
}

void wifi_connection()
{
    // 1 - Wi-Fi/LwIP Init Phase
    esp_netif_init();                 // TCP/IP initiation           s1.1
    esp_event_loop_create_default(); // event loop                      s1.2
    esp_netif_create_default_wifi_sta(); // WiFi station                s1.3
    wifi_init_config_t wifi_initiation = WIFI_INIT_CONFIG_DEFAULT();
    esp_wifi_init(&wifi_initiation); //                           s1.4
    // 2 - Wi-Fi Configuration Phase
    esp_event_handler_register(WIFI_EVENT, ESP_EVENT_ANY_ID, wifi_event_handler,
        NULL);
    esp_event_handler_register(IP_EVENT, IP_EVENT_STA_GOT_IP, wifi_event_handler,
        NULL);
    wifi_config_t wifi_configuration = {
        .sta = {
            .ssid = "Cheikh",                 //Change to WiFi network name
            .password = "Zc123@MOIM"}};       //Change to WiFi network password
    esp_wifi_set_config(ESP_IF_WIFI_STA, &wifi_configuration);
    // 3 - Wi-Fi Start Phase
    esp_wifi_start();
    // 4- Wi-Fi Connect Phase
    esp_wifi_connect();
}

esp_err_t client_event_get_handler(esp_http_client_event_handle_t evt)
{
    switch (evt->event_id)
    {
    case HTTP_EVENT_ON_DATA:
     if (!esp_http_client_is_chunked_response(evt->client)){
              http_index = http_index + evt->data_len;
              strcat(http_response, (char*)evt->data);
              http_response[http_index] = '\0';
     }
        // printf("HTTP_EVENT_ON_DATA: %.*s\n", evt->data_len, (char *)evt->data);
        break;

    default:
        break;
    }
    return ESP_OK;
}
```

```
void rest_get()
{
    http_index = 81;
    memset(http_response, 81, 1024);
    strcpy(http_response, "");

    esp_http_client_config_t config_get = {
        .url = "http://api.openweathermap.org/data/2.5/weather?q=Beirut,LB&units=
            metric&appid={API_KEY}",
        .method = HTTP_METHOD_GET,
        .cert_pem = NULL,
        .event_handler = client_event_get_handler};

    esp_http_client_handle_t client = esp_http_client_init(&config_get);
    esp_http_client_perform(client);
    esp_http_client_cleanup(client);
}
```

Listing 1: Code Listing for the ESP32 HTTP Client

3 ESP32 MQTT Client

There's 5 main functions in the MQTT Client.

1. convert_wind_speed which takes a double value of the wind speed in m/s from the Weather API as an input and converts it to km/h
2. convert_wind_direction which takes a double value of the wind direction in degrees fetched from the API and converts it to one of 8 characters that represents the direction like 'N' or 'SE'
3. mqtt_event_handler_cb which tells the MQTT Client what to do in different case events and thus subscribes to the desired topic so that it calls the HTTP Client to get the JSON data, parse the data using cJSON library, and publish it to the specified topic as a comma-separated text string.
4. mqtt_event_handler which packages the MQTT Client from the overall components so that it performs the coded functions
5. mqtt_app_start which is the main function that runs the MQTT Client by specifying the Broker IP Address and Port that the ESP32 MQTT Client will communicate with.

```
char http_response[1024];

static const char *TAG = "MQTT_TCP";

double convert_wind_speed(double wind_speed_meters_per_second)
{
    double wind_speed_km_per_h = wind_speed_meters_per_second * 3.6;
    return wind_speed_km_per_h;
}

char* convert_wind_direction(double wind_deg)
{
    if (wind_deg >= 337.5 || wind_deg < 22.5)
    { return "N";}
    else if (wind_deg >= 22.5 && wind_deg < 67.5)
    { return "NE";}
    else if (wind_deg >= 67.5 && wind_deg < 112.5)
    { return "E";}
    else if (wind_deg >= 112.5 && wind_deg < 157.5)
    { return "SE";}
    else if (wind_deg >= 157.5 && wind_deg < 202.5)
    { return "S";}
    else if (wind_deg >= 202.5 && wind_deg < 247.5)
    { return "SW";}
    else if (wind_deg >= 247.5 && wind_deg < 292.5)
    { return "W";}
    else if (wind_deg >= 292.5 && wind_deg < 337.5)
    { return "NW";}
    else
    { return "N/A";}
}

void rest_get();

static esp_err_t mqtt_event_handler_cb(esp_mqtt_event_handle_t event)
{
    esp_mqtt_client_handle_t client = event->client;
    switch (event->event_id)
    {
    case MQTT_EVENT_CONNECTED:
        ESP_LOGI(TAG, "MQTT_EVENT_CONNECTED");
        esp_mqtt_client_subscribe(client, "weather/get_weather", 0);
        break;
    case MQTT_EVENT_DISCONNECTED:
        ESP_LOGI(TAG, "MQTT_EVENT_DISCONNECTED");
        break;
    case MQTT_EVENT_SUBSCRIBED:
        ESP_LOGI(TAG, "MQTT_EVENT_SUBSCRIBED, msg_id=%d", event->msg_id);
        break;
    case MQTT_EVENT_UNSUBSCRIBED:
        ESP_LOGI(TAG, "MQTT_EVENT_UNSUBSCRIBED, msg_id=%d", event->msg_id);
        break;
    case MQTT_EVENT_PUBLISHED:
        ESP_LOGI(TAG, "MQTT_EVENT_PUBLISHED, msg_id=%d", event->msg_id);
        break;
    case MQTT_EVENT_DATA:
        ESP_LOGI(TAG, "MQTT_EVENT_DATA");
        printf("\nTOPIC=%.*s\r\n", event->topic_len, event->topic);
        printf("DATA=%.*s\r\n", event->data_len, event->data);

        rest_get();

        cJSON *root = cJSON_Parse(http_response);

        cJSON *_main = cJSON_GetObjectItem(root,"main");
        cJSON *_weather = cJSON_GetObjectItem(root,"weather");
        cJSON *_sys= cJSON_GetObjectItem(root,"sys");
        cJSON *_wind = cJSON_GetObjectItem(root,"wind");
        cJSON *_index = cJSON_GetArrayItem(_weather, 0);
        cJSON *wind_gust=NULL; //Placeholder so the ESP doesn't freak out
        if (cJSON_GetObjectItem(_wind,"gust")){
            wind_gust = cJSON_GetObjectItem(_wind,"gust");
        }
        char *city = cJSON_GetObjectItem(root,"name")->valuestring;
        char *country_code = cJSON_GetObjectItem(_sys,"country")->valuestring;
        int dt = cJSON_GetObjectItem(root, "dt")->valueint;
        int timezone = cJSON_GetObjectItem(root, "timezone")->valueint;
        double temp = cJSON_GetObjectItem(_main, "temp")->valuedouble;
        int humidity = cJSON_GetObjectItem(_main, "humidity")->valueint;
        char *description = cJSON_GetObjectItem(_index, "description")->valuestring;
        double wind_speed = cJSON_GetObjectItem(_wind, "speed")->valuedouble;
        double wind_deg = cJSON_GetObjectItem(_wind, "deg")->valuedouble;

        time_t epoch = dt + timezone;
        char *wind_direction = convert_wind_direction(wind_deg);
        wind_speed = convert_wind_speed(wind_speed);

        struct tm ts;
        char current_time[80];
        ts = *localtime(&epoch);
        strftime(current_time , sizeof ( current_time ), "%A, %B %d, %Y %I:%M:%S %p",
             &ts);


        if (wind_gust){

        int length = snprintf(NULL, 0,"%s,%s,%s,%.1f,%d,%s,%.1f,%s,%.1f", city,
            country_code, current_time, temp, humidity, description, wind_speed,
            wind_direction,wind_gust->valuedouble);
        char str[length+1];
        sprintf(str, "%s,%s,%s,%.1f,%d,%s,%.1f,%s,%.1f", city, country_code,
            current_time, temp, humidity, description, wind_speed, wind_direction,
            wind_gust->valuedouble);
        esp_mqtt_client_publish(client, "weather/current_weather", str, 0, 1, 0);
        }
        else{
            int length = snprintf(NULL, 0,"%s,%s,%s,%.1f,%d,%s,%.1f,%s", city,
                country_code, current_time, temp, humidity, description, wind_speed,
                wind_direction);
            char str[length+1];
            sprintf(str, "%s,%s,%s,%.1f,%d,%s,%.1f,%s", city, country_code,
                current_time, temp, humidity, description, wind_speed, wind_direction
                );
            esp_mqtt_client_publish(client, "weather/current_weather", str, 0, 1, 0);
        }
        break;
    case MQTT_EVENT_ERROR:
        ESP_LOGI(TAG, "MQTT_EVENT_ERROR");
        break;
    default:
        ESP_LOGI(TAG, "Other event id:%d", event->event_id);
        break;
    }
    return ESP_OK;
}

static void mqtt_event_handler(void *handler_args, esp_event_base_t base, int32_t
    event_id, void *event_data)
```

```
{
    ESP_LOGD(TAG, "Event dispatched from event loop base=%s, event_id=%d", base,
        event_id);
    mqtt_event_handler_cb(event_data);
}

void mqtt_app_start(void)
{
    esp_mqtt_client_config_t mqtt_cfg = {
        .uri = "mqtt://local_ip_address:1883", //Change to mqtt://(device's IPv4
            address):1883
    };
    esp_mqtt_client_handle_t client = esp_mqtt_client_init(&mqtt_cfg);
    esp_mqtt_client_register_event(client, ESP_EVENT_ANY_ID, mqtt_event_handler,
        client);
    esp_mqtt_client_start(client);
}
```

Listing 2: Code Listing for the ESP32 MQTT Client

4 Mosquitto broker

4.1 Setup

After installing Mosquitto on the local computer, we go to the directory to which Mosquitto installed in. We edit the configuration file to run the broker on the local IP Address of the PC used and port 1883, as well as configure the broker to accept connections from other clients:

listener 1883 Local_IP_Address
allow_anonymous true

We then run the Broker using the new configuration settings in the Command Terminal by running the command:

mosquitto -v -c mosquitto.conf

A Mosquitto Broker is then setup on Local_IP_Address and port 1883. (i.e. mqtt://Local_IP_Address:1883)

4.2 Testing

To test the Broke, we open two additional command terminal windows (one to publish an MQTT topic and another to subscribe to the topic). From the first terminal, we use the command:

mosquitto_pub -h Local_IP_Address -t topic/test -m ”hello” -r -i client -d

which publishes the topic/test topic containing a ”hello” message. The other MQTT client then subcribes to topic/test by running the command:

mosquitto_sub -h Local_IP_Address -t topic/test -d

and in the terminal we see the ”hello” message. Thus, we have configured and tested the MQTT Broker successfully.

5 Node-RED Client/UI

5.1 Setup and Configuration

After installing Node-Red, we run in the terminal the command:

node-red

which boots the Node-Red environment locally on 127.0.0.1:1880.

5.2 Flow code in graphical form

From Fig. 2, the Node-Red Flow code is showcased in graphical form. The Flow code shows 5 nodes:

1. *Retrieve Weather* which is the button which carries a "Retrieve Weather Data" message to the *MQTT Out* node
2. *MQTT Out* which publishes the message to the MQTT topic **weather/get_weather**
3. *MQTT In* which subscribes to the **weather/current_weather** MQTT topic that has the weather data published by the ESP32.
4. *format text* which extracts the weather parameters by splitting them according to the commas and then updates the message payload to be sent to the *template* node
5. *template* which shows the message payload according to a specific format through HTML

5.3 UI design

As shown in Fig. 3 and Fig. 4, the user interfaces shows the weather data as required with a FontAwesome weather icon that describes how the weather feels like currently in a particular region. We can also see a "RETRIEVE WEATHER" Button which triggers the UI to update the weather data by having the ESP32 fetch new data before publishing to the corresponding MQTT topic.

6 Contributions

The work was divided fairly and each member dedicated much of their time and contributed equally to this project. Mohammad El Sheikh worked on the ESP32 MQTT Client code as well as setting the Mosquitto Broker and integrating all the components together to be tested out on the ESP32 and Local Area Network. Taha Koleilat worked on the ESP32 HTTP Client code, along with setting up the Node-Red Client with the flow code and UI to display the final result.

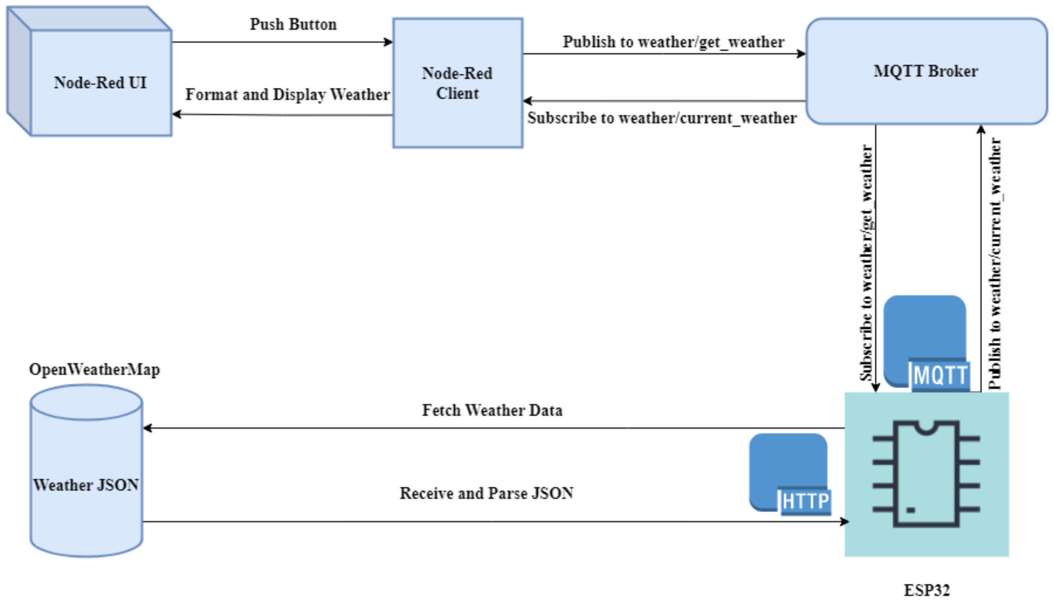

Node-Red UI
Push Button
Format and Display Weather
Node-Red Client
Publish to weather/get_weather
Subscribe to weather/current_weather
MQTT Broker
Subscribe to weather/get_weather
Publish to weather/current_weather
MQTT
OpenWeatherMap
Weather JSON
Fetch Weather Data
Receive and Parse JSON
HTTP
ESP32

Figure 1: A System-Level Architecture diagram depicting the main components of the application.

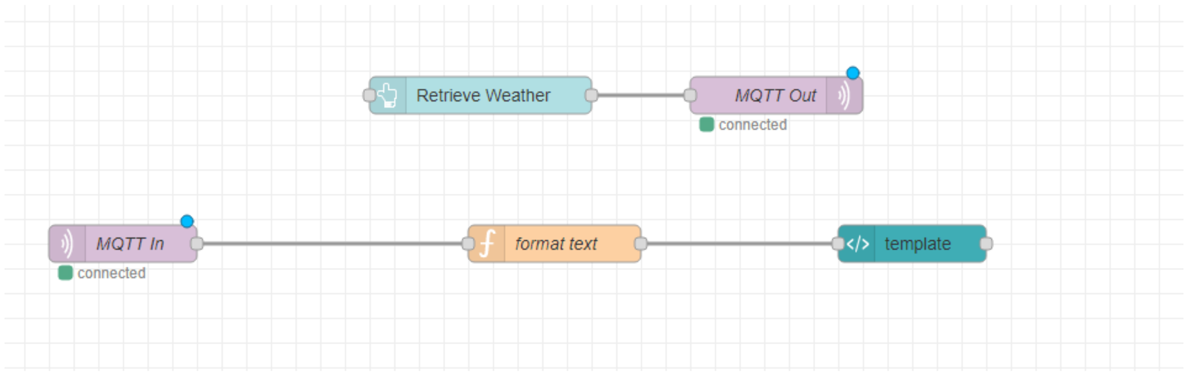
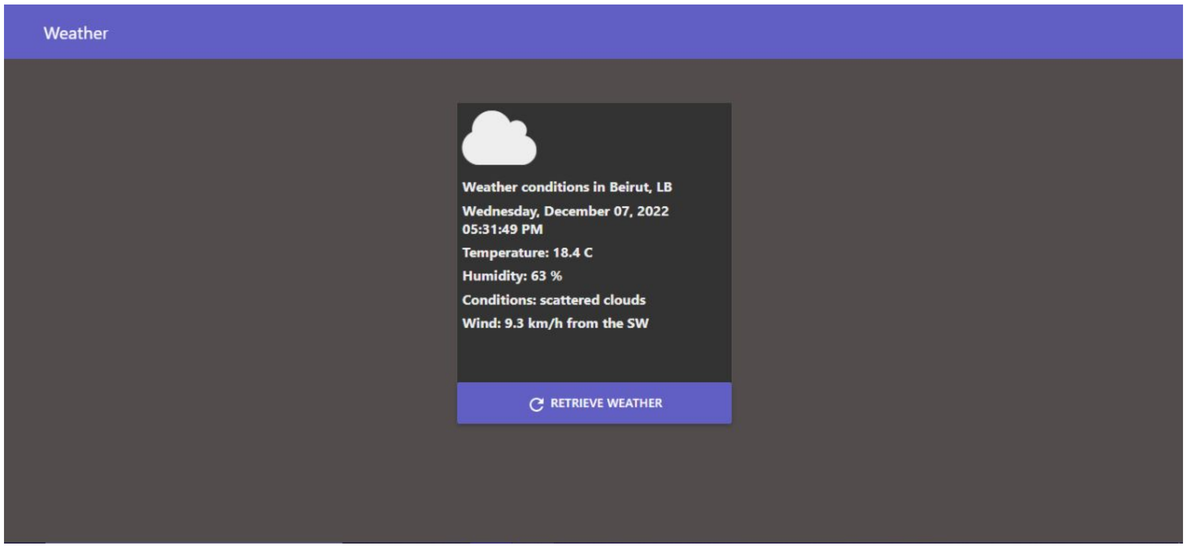


Figure 2: The Node-Red Flow code in Graphical Form



Weather conditions in Beirut, LB

Wednesday, December 07, 2022
05:31:49 PM

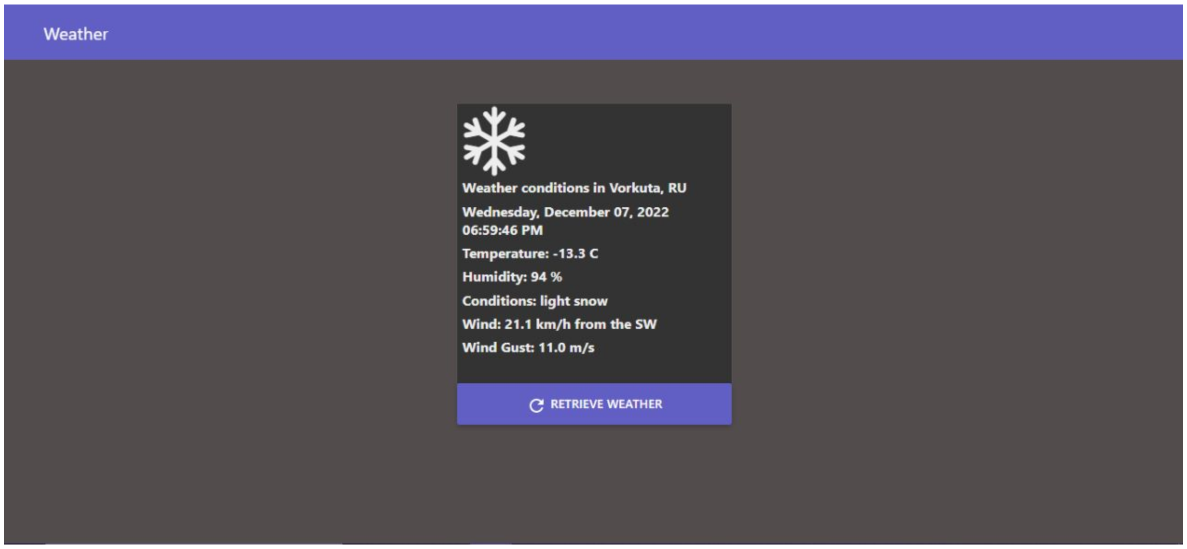
Temperature: 18.4 C

Humidity: 63 %

Conditions: scattered clouds

Wind: 9.3 km/h from the SW

Figure 3: The Node-Red UI design showcasing the displayed weather data in Beirut, LB along with a Button



Weather conditions in Vorkuta, RU

Wednesday, December 07, 2022
06:59:46 PM

Temperature: -13.3 C

Humidity: 94 %

Conditions: light snow

Wind: 21.1 km/h from the SW

Wind Gust: 11.0 m/s

Figure 4: The Node-Red UI design showcasing the displayed weather in Vorkuta, RU with the gust value