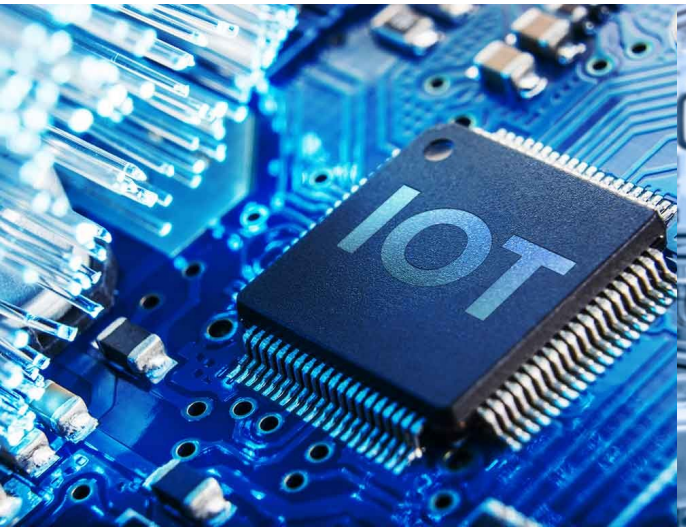
Desarrollo de aplicaciones para IoT



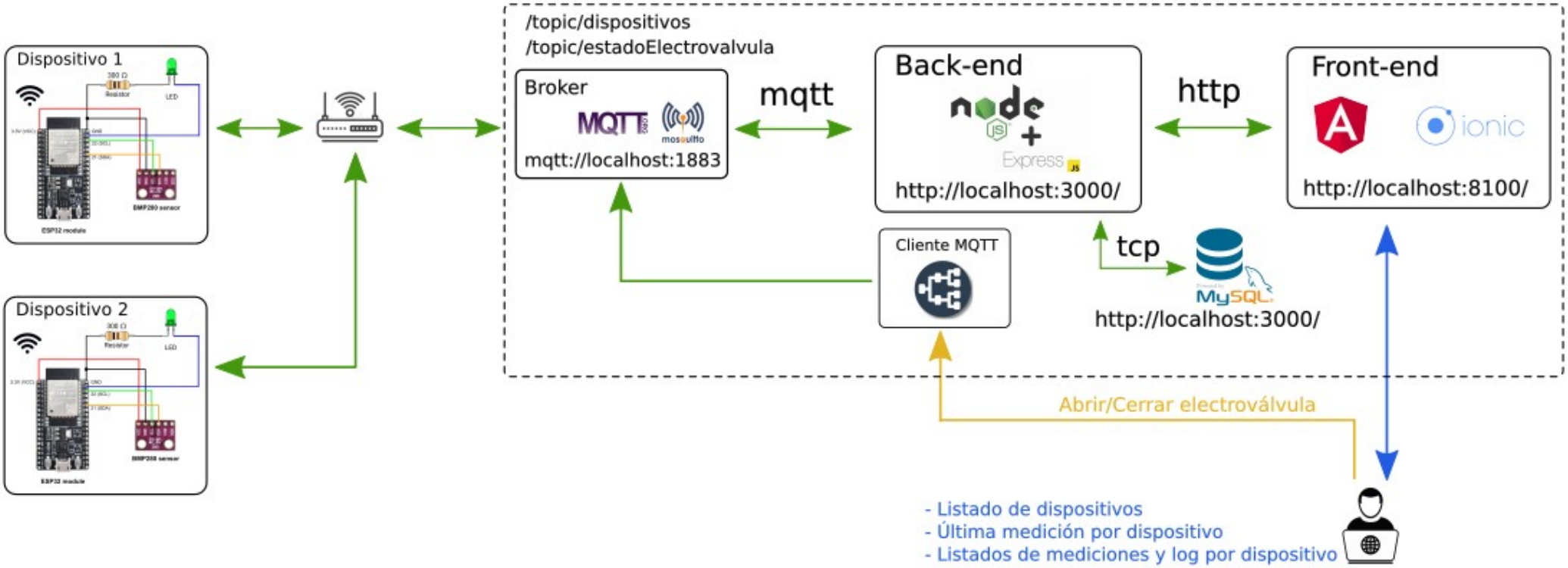
.UBA fiuba
FACULTAD DE INGENIERÍA

Especialización en IoT. 6co

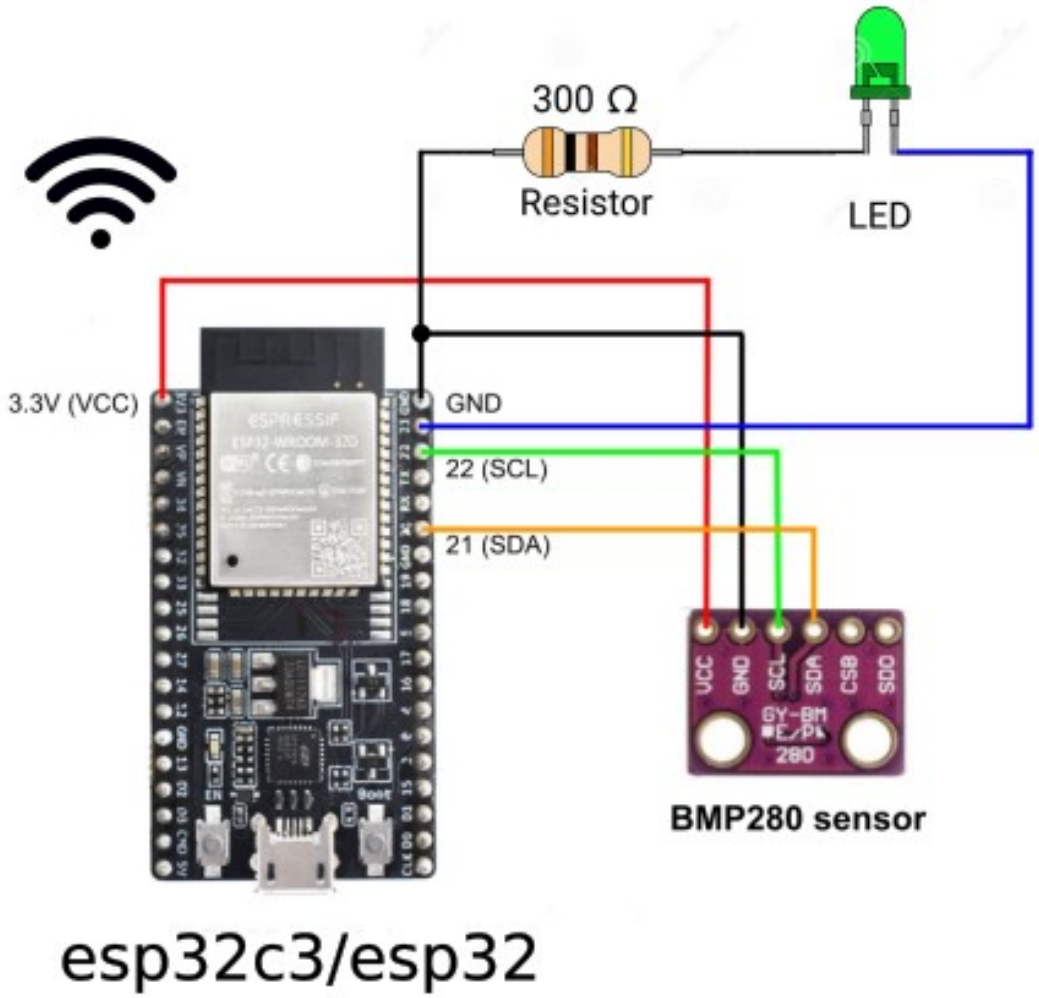
Trabajo integrador

Estudiante: Raúl Romero

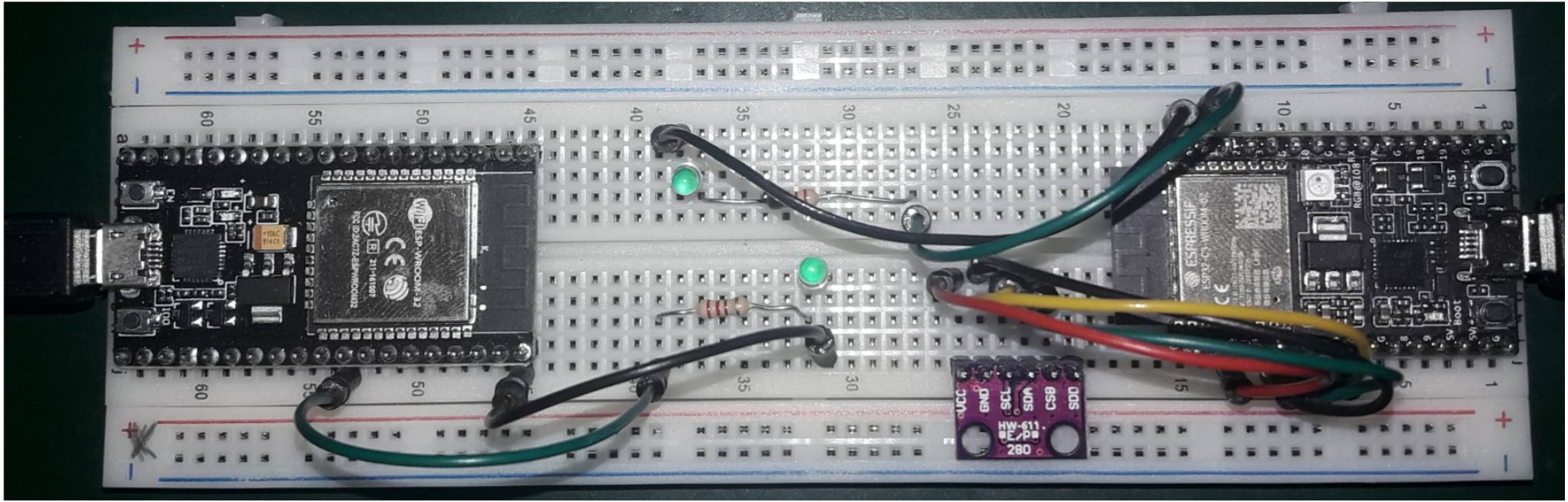
Arquitectura: localhost



Nodo dispositivo



Nodo dispositivo



esp32

esp32c3

Nodo dispositivo: datos de envío


```
{
    “dev_id”:1,
    “temperatura”:23.4,
    “pressure”:100000.02,
    “rssi”:-42,
    “electroState”:0
}
```

- ElectroState 0: Electroválvula Abierta
- ElectroState 1: Electroválvula Cerrada

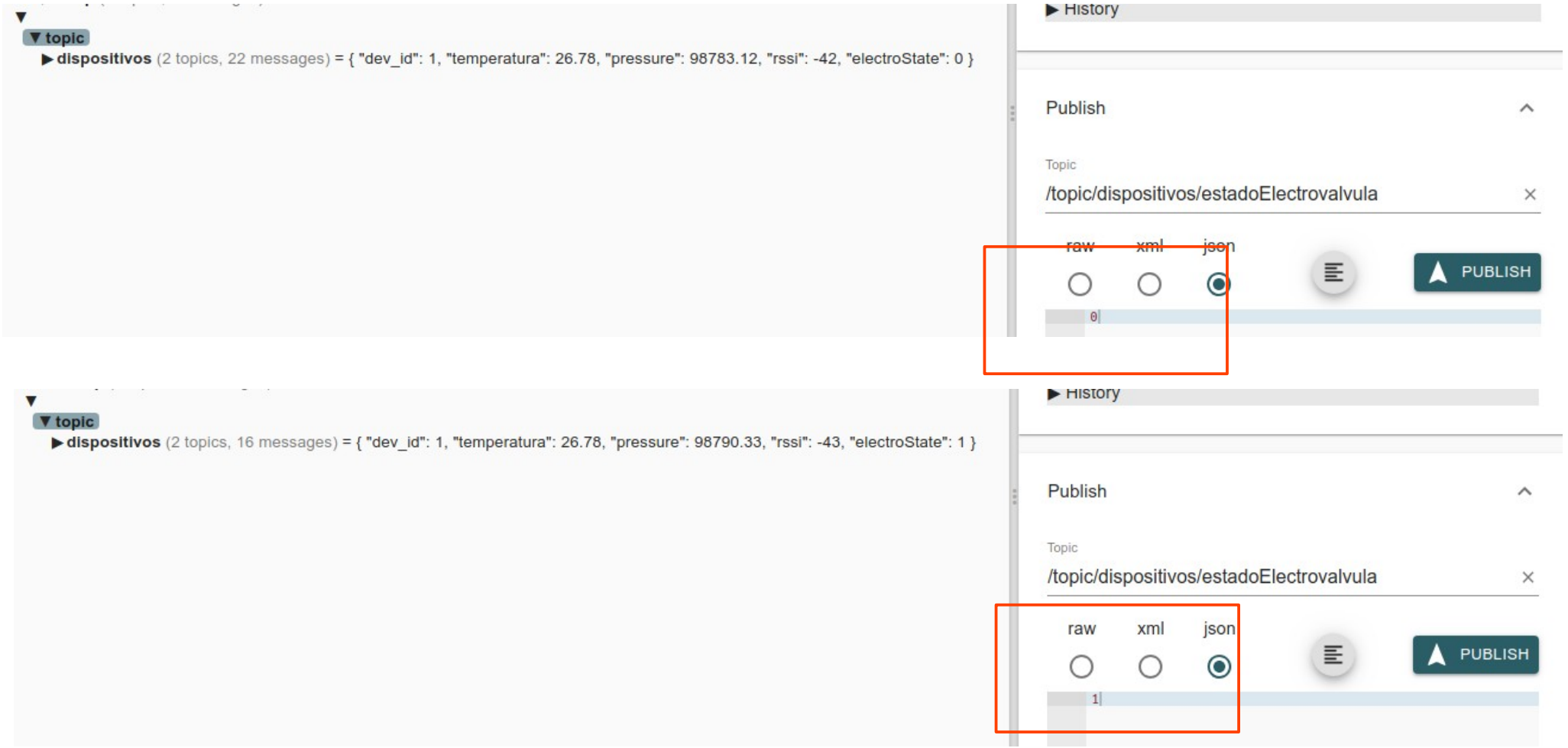
Nodo dispositivo: monitor

```
Queue should have been empty!
Lectura del sensor-> Pressure: 98783.12 Pa, Temperature: 26.78 C
Lectura desde Queue-> Pressure: 98784.46 Pa, Temperature: 26.77 C
Could not receive from the queue.
I (159195) MQTT MODULE: : MQTT_EVENT_DATA:
TOPIC=/topic/dispositivos/estadoElectrovalvula
DATA=0
Activar electroválvula=0

I (159205) MQTT MODULE: : RSSI: -42
I (159205) Datos a enviar: : T 26.77 - Pa 98784.46 - RSSI -42 - electroEstate 0
I (159215) JSON enviado:: { "dev_id": 1, "temperatura": 26.77, "pressure": 98784.46, "rssi": -42, "electroState": 0 }
I (159225) JSON enviado:: /topic/dispositivos
I (159235) MQTT MODULE: : sent publish successful, msg_id=58464
I (159405) MQTT MODULE: : MQTT_EVENT_PUBLISHED, msg_id=58464
```

Nodo dispositivo: monitor

```
Lectura del sensor-> Pressure: 98787.83 Pa, Temperature: 26.77 C
Lectura desde Queue-> Pressure: 98783.40 Pa, Temperature: 26.77 C
Queue should have been empty!
Activar electroválvula=1

I (119105) MQTT MODULE: : RSSI: -42
I (119115) Datos a enviar: : T 26.77 - Pa 98783.40 - RSSI -42 - electroEstate 1
I (119125) JSON enviado::  { "dev_id": 1, "temperatura": 26.77, "pressure": 98783.40, "rssi": -42, "electroState": 1 }
I (119135) JSON enviado::  /topic/dispositivos
I (119135) MQTT MODULE: : sent publish successful, msg_id=57464
I (119165) MQTT MODULE: : MQTT_EVENT_PUBLISHED, msg_id=57464
```

Cliente MQTT



Firmware nodo: main.c

```
void app_main(void)
{
    ESP_ERROR_CHECK( nvs_flash_init() );

    ESP_ERROR_CHECK(i2cdev_init());

    xQueue = xQueueCreate( 1, sizeof( Data_t ) );
    xQueue1 = xQueueCreate( 1, sizeof( int) );
    xQueue2 = xQueueCreate( 1, sizeof( int) );

    initialise_wifi();

    initialize_sntp();

    while (!time_sinc_ok) vTaskDelay(100 * 1);

    xTaskCreate(mqtt_app_main_task, "mqtt_app_task", 4096 * 8, NULL, 3, NULL);

    xTaskCreate(led_task, "led_task", 4096 * 8, NULL, 3, NULL);

    xTaskCreate(publicar_temperatura_task, "temp_pub_task", 4096 * 8, NULL, 3, NULL);

    xTaskCreatePinnedToCore(bmp280_test, "bmp280_test", configMINIMAL_STACK_SIZE * 8,

}
```

Firmware nodo: bmp280_task

```
while (1)
{
    vTaskDelay(pdMS_TO_TICKS(500));
    if (bmp280_read_float(&dev, &temperature, &pressure, &humidity) != ESP_OK)
    {
        printf("Temperature/pressure reading failed\n");
        continue;
    }
    //cargamos los datos de presión y temperatura a la cola
    send_dataSensor_queue.temperature=temperature;
    send_dataSensor_queue.pressure=pressure;
    xStatus = xQueueSendToBack( xQueue, &(send_dataSensor_queue), 300);
    if( xStatus != pdPASS )
    {
    /* The send operation could not complete because the queue was full -
    this must be an error as the queue should never contain more than
    one item! */
    printf( "Could not send to the queue.\n" );
    }

    printf("Lectura del sensor-> Pressure: %.2f Pa, Temperature: %.2f C", pressure, temperature);
    if (bme280p)
    printf(", Humidity: %.2f\n", humidity);
    else
    printf("\n");
}
```

Firmware nodo: publicar_temperatura_task

```
while(1)
{

    if (mqtt_client_connected == true) {
    // INICIO CICLO DE LECTURAS y publicaciones.
    vTaskDelay(TEMP_PUBLISH_INTERVAL_SECONDS * 1000 / portTICK_PERIOD_MS);

    if( uxQueueMessagesWaiting( xQueue ) != 0 )
    {
    // verificamos si se recibieron datos en la cola desde bmp280_test
    printf( "Queue should have been empty!\n" );
    }

    xStatus = xQueueReceive( xQueue, &receive_dataSensor_queue, xTicksToWait );

    if( xStatus == pdPASS )
    {
    /* Data was successfully received from the queue, print out the received value. */
    temp=receive_dataSensor_queue.temperature;
    press=receive_dataSensor_queue.pressure;
    printf( "Lectura desde Queue-> Pressure: %.2f Pa, Temperature: %.2f C", press,temp);
    printf("\n");
    }
    else
    {
    /* Data was not received */
    printf( "Could not receive from the queue.\n" );
    }
```

Firmware nodo: mqtt_event_handler

```
case MQTT_EVENT_DATA:
    ESP_LOGI(TAG, "MQTT_EVENT_DATA: ");
    printf("TOPIC=%.*s\r\n", event->topic_len, event->topic);
    printf("DATA=%.*s\r\n", event->data_len, event->data);
    /*****************************/
    char *pcStringToSend;
    char *estado="1";
    pcStringToSend=(char *)event->data;
    int electroState=0;
    if(strncmp(pcStringToSend, estado,1)==0)
    {
        electroState=1;
    }else
    {
        electroState=0;
    }
    xStatus = xQueueSendToBack( xQueue1, &electroState, 300);
        if( xStatus != pdPASS )
        {
        /* The send operation could not complete because the queue was full -
        this must be an error as the queue should never contain more than
        one item! */
        printf( "Could not send to the queue.\n" );
        }
    /*****************************/
```

Firmware nodo: led_task

```
void led_task(void *pvParameters)
{
    BaseType_t xStatus;
    static uint8_t s_led_state = 0;
    const TickType_t xTicksToWait = pdMS_TO_TICKS( 1000 );
    gpio_reset_pin(BLINK_GPIO);
    /* Set the GPIO as a push/pull output */
    gpio_set_direction(BLINK_GPIO, GPIO_MODE_OUTPUT);

    while (1) {
    /* Set the GPIO level according to the state (LOW or HIGH)*/
     if( uxQueueMessagesWaiting( xQueue2 ) != 0 )
        {
        // verificamos si se recibieron datos en la cola desde bmp280_test
        printf( "Queue should have been empty!\n" );
        }

        xStatus = xQueueReceive( xQueue2, &s_led_state, xTicksToWait );

        if( xStatus == pdPASS )
        {
        /* Data was successfully received from the queue, print out the received value. */

        printf("\n");
        }
        else
        {
        /* Data was not received */
        printf( "Could not receive from the queue.\n" );
        }
            gpio_set_level(BLINK_GPIO, s_led_state);
    }
}
```