# Flujo código ESP32

El microcontrolador ESP32 se ha programado para que actúe como nodo intermediador entre los dispositivos ESP8266 y el servidor web del huerto docente. El código consiste en una pasarela ESP-Now a MQTT con autoemparejamiento entre dispositivos. A continuación, se explica de forma detallada las partes principales del código y las acciones llevadas a cabo en cada una de ellas.

* Función principal de configuración “void setup()”

Esta función se ejecuta solamente una vez al inicio del dispositivo. Las acciones que implementa son:

* + Inicializa la comunicación en serie. Se inicia la comunicación con el monitor serie a una velocidad de 115200 baudios para la depuración y visualización de mensajes.
  + Establece la conexión WiFi. Antes de realizar la conexión, ejecuta la función “sprintf(ID\_PLACA, "ESP\_%d", ESP.getEfuseMac())”, con el fin de adquirir la dirección MAC del dispositivo y almacenarla en la variable “ID\_PLACA” para utilizarla en el protocolo de comunicación ESP-Now. A continuación, Se llama a la función “conecta\_wifi” para establecer la conexión WiFi utilizando las credenciales proporcionadas (SSID y contraseña).
  + Realiza tareas de configuración. En este punto, se configura:
    - Configuración del Canal WiFi: Se obtiene el número del canal WiFi en el que se encuentra el dispositivo ESP32 y se almacena en la variable “myChannel”.
    - Configuración del Cliente MQTT: Se configura el cliente MQTT (mqtt\_client) para que se conecte al servidor MQTT en la dirección y puerto especificados. También se establece el tamaño del búfer de mensajes a 512 bytes y se registra la función “procesa\_mensaje()” como callback para manejar los mensajes MQTT entrantes.
  + Configura ESP-NOW. Se llama a la función “initESP\_NOW()” para inicializar la comunicación utilizando el protocolo ESP-NOW. En ella se definen otras funciones y callbacks relacionados con ESP-NOW, como addPeer para agregar dispositivos emparejados, OnDataSent para manejar la confirmación de envío y OnDataRecv para procesar los datos recibidos.
* Función “encuentra\_mensaje()”

Se ejecuta antes de la función principal “void loop()”. Busca un mensaje en la cola de mensajes MQTT para un dispositivo específico. Para ello se ha declarado como un iterador “std::list<TmensajeMQTT>::iterator”, esto indica que la función encuentra\_mensaje(),recorre la lista de mensajes MQTT (TmensajeMQTT), buscando y captando mensajes procedente del dispositivo cuya MAC se le pase como argumento(uint8\_t \*\_mac) para poder procesarlos posteriormente.

* Función principal del programa “void loop()”

Es el bucle principal del programa, en esta función se implementa la pasarela ESP-Now a MQTT. Se encarga de mantener en funcionamiento las comunicaciones MQTT y ESP-NOW, gestionando los dispositivos listos para ser emparejados o los ya emparejados y activos, para enviar o revivir mensajes pendientes a través de ambos protocolos de comunicación.

## Documentación y desarrollo del código función “void loop()”

Reconexión al Servidor MQTT: Si el cliente MQTT no está conectado, se llama a la función “conecta\_mqtt()” para intentar conectarse al servidor MQTT. Luego, se permite que el cliente MQTT procese los mensajes entrantes utilizando “mqtt\_client.loop()”.

static unsigned long lastEventTime = millis();

  if (!mqtt\_client.connected()) conecta\_mqtt();

  mqtt\_client.loop(); // Permite que la librería MQTT procese los mensajes

Procesamiento de Dispositivos Listos (readyMACs): Si la cola “readyMACs” no está vacía, significa que hay dispositivos que están listos para recibir mensajes. El programa obtiene la dirección MAC del dispositivo listo desde la cola y busca si hay mensajes en la lista “mensajes\_MQTT” para ese dispositivo. Si encuentra un mensaje, lo envía a través de ESP-NOW y elimina el mensaje de la lista “mensajes\_MQTT”.

  if (!readyMACs.empty()) {

    uint8\_t mac\_addr[6];

    uint8\_t size;

    String mac = readyMACs.front();

    struct\_espnow mensaje\_esp;

    //

    Serial.println("Buscando mensajes para dispositivo "+ mac);

    HEX2byte(mac\_addr, (char\*)mac.c\_str());

    std::list<TmensajeMQTT>::iterator it= encuentra\_mensaje(mac\_addr);

    if(it==mensajes\_MQTT.end())

    {

      mensaje\_esp.msgType=NODATA;

      size=1;

      readyMACs.pop();

      Serial.println(" > Sin mensajes pendientes");

    }

    else

    {

      mensaje\_esp.msgType=DATA;

      memcpy(mensaje\_esp.payload, it->data, it->len);

      size=it->len+1;

      mensajes\_MQTT.erase(it);

      Serial.println(" > Un mensaje encontrado");

    }

esp\_err\_t result = esp\_now\_send(mac\_addr, (uint8\_t \*) &mensaje\_esp, size);

  }

Envío de Notificación de Emparejamiento (pairMACs): Si la cola “pairMACs” no está vacía, significa que hay dispositivos listos para emparejarse. El programa obtiene la dirección MAC del dispositivo desde la cola y envía un mensaje MQTT de notificación de emparejamiento a través de la función “mqtt\_client.publish()”.

if (!pairMACs.empty())

  {

    uint8\_t mac\_addr[6];

    uint8\_t size;

    String mac = pairMACs.front();

    pairMACs.pop();

    Serial.println("Enviando mensaje MQTT notificación pairing "+ mac);

    sprintf(mensaje\_mqtt, "{\"mac\":\"%s\"}",mac.c\_str());

    mqtt\_client.publish("infind/espnowpairing", mensaje\_mqtt);

  }

Envío de Mensajes Pendientes (cola\_mensajes): Si la cola “cola\_mensajes” no está vacía, significa que hay mensajes en espera para ser enviados. El programa obtiene el primer mensaje de la cola, procesa su contenido y lo envía al servidor MQTT utilizando la función “mqtt\_client.publish()”.

if (!cola\_mensajes.empty())

  {

    TmensajeESP mensaje = cola\_mensajes.front();

    cola\_mensajes.pop();

    deviceMac = byte2HEX(mensaje.mac);

    byte len =  mensaje.len;

    // recibimos el mensaje

    for(int i = 0; i<len ; i++)

    {

       JSON\_serie[i]=mensaje.data[i];

    }

    JSON\_serie[len]='\0'; //fin de cadena de caracteres de C

    Serial.printf("Mensaje len: %d \n payload: %s\n", len, JSON\_serie);

// parece que la deserialización altera la cadena de caracteres original, así que la copiamos antes

    strcpy(JSON\_serie\_bak,JSON\_serie);

    //comprobamos si hay campo topic en el mensaje, para añadirlo al topic de envío

    StaticJsonDocument<512> doc;

    // Deserialize the JSON document

    DeserializationError error = deserializeJson(doc, JSON\_serie, len);

    sprintf(topic\_PUB, "infind/espnow/%s", deviceMac.c\_str());

    // Compruebo si no hubo error

    if (error) {

      Serial.print("Error sintaxis JSON; deserializeJson() failed: ");

      Serial.println(error.c\_str());

      mqtt\_client.publish(topic\_PUB, "{\"error\":\"Error sintaxis JSON; deserializeJson() failed\"}");

    }

    else

    if(doc.containsKey("topic") && doc["topic"].is<const char\*>())  // comprobar si existe el campo/clave que estamos buscando

    {

     String valor = doc["topic"];

     Serial.print("Mensaje OK, topic = ");

     Serial.println(valor);

     sprintf(topic\_PUB, "infind/espnow/%s/%s", deviceMac.c\_str(), valor.c\_str() );

    }

    else // no existe el campo topic

    {

      Serial.println("\"topic\" key not found in JSON");

    }

    sprintf(mensaje\_mqtt, "{\"mac\":\"%s\",\"payload\":%s}",deviceMac.c\_str(),JSON\_serie\_bak);

    // publica el mensaje recibido

    mqtt\_client.publish(topic\_PUB, mensaje\_mqtt);

    Serial.printf("Mensaje from: %s, publicado en MQTT\n", deviceMac.c\_str());

    Serial.printf("       topic: %s\n", topic\_PUB);

    Serial.printf("     payload: %s\n\n", mensaje\_mqtt);

}