**Proyecto red IOT**

**Descripción del proyecto**

La idea es crear redes de dispositivos para IOT (Internet of things) en Arduino. Dichos dispositivos no deberán tener ninguna configuración especial para poder conectar a la red, simplemente deberán ser autorizados (emparejados) desde la consola de Domótica. A partir de ahora los llamaremos IOTD (IOT Device)

La consola de Domótica dispondrá de un servidor web al que se conectará su administrador y donde deberá ser capaz de autorizar y vincular nuevos dispositivos de forma sencilla y automática.

Una vez los dispositivos estén vinculados, el administrador podrá enviarles comandos desde la consola en función del tipo de dispositivo o simplemente recibir la información que el mismo genere.

**Estado actual**

En el momento actual, aún no ha comenzado el desarrollo de ninguno de los módulos que forman el proyecto ya que estamos en la fase de recogida de especificaciones y análisis, por lo que todos los puntos del presente documento pueden modificarse.

**Consola Domótica**

Dispositivo ESP32 en modo AP y con SD para almacenar datos. Deberá disponer de un botón para volver a valores de fábrica (RESET pulsado 10 segundos o similar).

Dispondrá de un servidor web en el que el propietario podrá gestionar los dispositivos autorizados, del total que se hayan conectado, también se podrá poner en modo “acoplamiento” o “normal”.

Tras 2 minutos de inactividad, el modo volverá a ser “normal” siempre.

En el modo “acoplamiento”, los IOTD nuevos podrán conectar a la consola con unas credenciales estándar (SSID: IOT\_SERVER – Pass: vincular), y una vez autorizados quedarán registrados en la consola para futuros usos. En ese momento la consola enviará encriptadas al IOTD nuevas credenciales para poder acceder en modo normal (SSID del router wifi, Password y consoleID). El IOTD desencriptará las credenciales y las usará a partir de entonces.

En la consola quedará registrada la autorización del IOTD guardando su MAC Address así como el tipo de dispositivo y otros parámetros.

En el modo “normal”, la consola dispondrá de un SSID único (oculto) y una password única que conocerán los dispositivos vinculados.

Por defecto, la consola tendrá una dirección IP 192.168.1.80 y usuario y contraseña (Admin- Admin), que podrán ser modificados desde cualquier dispositivo que se encuentre en la misma red.

El botón de RESET restablecerá la dirección IP original y limpiará otros datos como configuraciones de conexión a SSID, usuarios y contraseñas y IOTD vinculados.

La consola deberá generar un LOG detallado de todos los cambios realizados en su configuración así como de todas las conexiones o errores que se produzcan. Dicho log podrá ser circular o estar limitado en tamaño o fecha.

El consoleID es un nº al azar generado la primera vez que se usa la consola o al volver a valores de fábrica.

En la consola se definirán todos los modos de trabajo de todos los tipos dedispositivos soportados así como sus parámetros y qué hacer con la información enviada o recibida.

**Dispositivo IOT**

Los IOTD serán dispositivos ESP32 que tendrán un botón para desvincular o resetear configuraciones.

Cuando un IOTD se conecte por primera vez estará en modo “acoplamiento” y permanecerá así hasta que encuentre un dispositivo de tipo Consola. Se validará en él y quedará a la espera de autorización. Si la conexión se interrumpe, seguirá intentando conectar de nuevo.

Si es autorizado, recibirá el SSID y una contraseña del router y obtendrá una IP de la red Wifi en la que esté conectada la consola, que deberá comenzar a utilizar a partir de ese momento como opción principal de conexión. Los valores de SSID y contraseña se almacenarán en la EEPROM del módulo ESP32, juntamente con la “consoleID” a utilizar a partir de entonces.

Una vez que el IOTD esté autorizado y conectado a la consola, deberá comenzar a trabajar en modo normal.

En el momento en que alguien pulse el botón de “Reset”, el dispositivo volverá a modo “acoplamiento”.

**Comunicación entre la consola y los IOTD**

A fin de tener centralizado el acceso a dispositivos, todas las consultas serán realizadas desde cada IOTD.

Sintaxis:

console\_Send(deviceID, consoleID, parámetro, valor);

Valor=console\_receive(deviceID, consoleID, parámetro);

Ejemplos:

// En este caso estamos enviando una temperatura al servidor, para que la guarde en la BBDD.

Console\_send(234,a4asklsd3,”temperatura”,25);

// Este caso, solicitamos que nos devuelva el valor de una bombilla, para decidir si debemos encenderla

Valor=console\_receive(245, a4asklsd3,status):

**Primer arranque de consola**

Al arrancar por primera vez, se genera un ID único de servidor que se guarda en EEPROM y se queda en modo acoplamiento hasta que se acople y autorice el primer dispositivo IOTD.

En este punto, ya debemos poder configurar el modo de trabajo normal desde la web.

Datos modo acoplamiento:

SSID: IOT\_SERVER

Pass wifi: vincular

IP acoplamiento: 192.168.1.80

Puerto web: 80

Datos modo estándar:

Nombre <default>: Consola\_IOTD

IP <default>: 192.168.1.80

Puerto web <default>: 80

User web <default>: Admin

Password web <default>: Admin

SSID del Wifi destino: <blanco- requerido>

Pass del Wifi destino:<blanco - requerido>

Seguimos en modo acoplamiento, es decir esperando conexiones en el SSID IOT\_SERVER.

Cada ciertos segundos, se mostrarán en la web los dispositivos que intentan conectar y podremos autorizar uno.

Una vez que se haya autorizado alguno, ya se podrá cambiar a modo Normal. Si el usuario no cambia en 2 minutos, el cambio se hará automáticamente.

**Primer arranque de IOTD**

Al arrancar por primera vez, se genera un ID único de IOTD (IOTD\_ID) que se guarda en EEPROM y se queda en bucle intentando acoplarse a una consola. Para hacerlo, utilizará los datos por defecto de acoplamiento.

Datos modo acoplamiento:

SSID: IOT\_SERVER

Pass wifi: vincular

IP acoplamiento: 192.168.1.80

Puerto web: 80

Una vez autorizado, recibiremos los siguientes datos de la consola:

Nombre <default>: Consola\_IOTD

IP <default>: 192.168.1.80

Puerto web <default>: 80

ConsoleID: <xxxxxxxx>

IOTD\_Num: <nº de IOTD registrado en la consola (Siempre incremental y único)>

SSID del Wifi destino: <blanco- requerido>

Pass del Wifi destino:<blanco - requerido>

A partir de ese momento, el IOTD debe conectar al nuevo SSID indicado y empezar a trabajar en modo normal, enviando o pidiendo datos al servidor cada cierto tiempo.

**Acoplamiento de un IOTD a la consola**

Al ser autorizado desde la consola el IOTD y con los datos recibidos, preparará una Key para encriptación/desencriptación y comenzará a trabajar en modo normal, enviando o recibiendo datos cada cierto tiempo.

Key=ConsoleID + IOTD\_ID

Asociados al IOTD existirán algunos campos para cumplimentar información adicional. Por ejemplo:

* Ubicación
* Descripcion

**ENVIO de datos a la consola**

Se usará la función:

Console\_send(IOTD\_Num,Tipo\_ID, Datos)

**Tipo\_ID=**Nº del tipo de ID según lista siguiente. En principio este tipo se usará para hacer una identificación visual del IOTD (mostrar iconos y similar). Este valor es opcional.

1. Switch ON/OFF
2. Temperature

255 - Custom

Datos=String que contiene los valores separados por comas.

**Ejemplo de parámetro “Datos”:**

“sw”:”1” <- En este caso, estaríamos encendiendo una luz.

“lb”:”0”,”temp”:”26,2”,”lr”:”255”,”co2”:”300ppm”

<- ·En este caso sería una sonda múltiple de acuario

“lat”:”50.0467656”,”lon”:”20.0048731” <- En este caso sería un GPS

La función “Console\_Send” realizará internamente una encriptación mediante AES 128b antes de enviar y usará como clave de encriptación la variable Key, que estará compuesta de la suma de dos valores.

**Key=**ConsoleID + IOTD\_ID

La consola identificará que el comando viene de un IOTD\_NUM en concreto y de ahí obtendrá el IOTD\_ID para poder realizar la desencriptación.

**RECEPCION de datos desde la consola**

Se usará la función:

Valor=Console\_receive(IOTD\_Num,Parametros)

Parametros =String que contiene los valores separados por comas.

YA DESENCRIPTADOS.

Ejemplo de “Parametros”:

“sw” <- En este caso, pedimos información sobre el estado deseado para una bombilla.

“texto” <- En este caso, si fuese en la sonda del acuario, pediríamos sólo el valor de texto a mostrar en pantalla.

“luz1”,”luz2”,”luz3” <- en este caso pediremos los estados de 3 luces.

“lb”:”0”,”temp”:”26,2”,”lr”:”255”,”co2”:”300ppm”

<- ·En este caso sería una sonda múltiple de acuario

“lat”:”50.0467656”,”lon”:”20.0048731” <- En este caso sería un GPS

La función “Console\_Receive” realizará internamente una desencriptación mediante AES 128b al recibir los datos y usará como clave de encriptación la variable Key, que estará compuesta de la suma de dos valores.

**Key=**ConsoleID + IOTD\_ID

El IOTD\_ID ya poseerá el CONSOLE\_ID, que obtuvo durante el emparejamiento para poder realizar la desencriptación.

**NOTA IMPORTANTE:** Para inicializar un IOTD será recomendable enviar un comando “Console\_Send” con todos los parámetros iniciales, de esa manera, al realizar un “Console\_Receive” la consola podrá devolver como mínimo el último valor registrado.

Si un valor no está definido en la BBDD de la consola, se devolverá NULL.

Ejemplo de código para consola para ponerse en modo “AP”:

#include <WiFi.h>

const char \*ssid = "MyESP32AP";

const char \*password = "testpassword";

void setup() {

  Serial.begin(115200);

WiFi.softAP(ssid, password);

  Serial.println();

  Serial.print("IP address: ");

  Serial.println(WiFi.softAPIP());

}

void loop() {}

Ejemplo de código para cliente para ponerse en modo “Conexión a wifi”

#include <WiFi.h>

const char\* ssid = "yourNetworkName";

const char\* password =  "yourNetworkPass";

void setup() {

  Serial.begin(115200);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL\_CONNECTED) {

    delay(500);

    Serial.println("Connecting to WiFi..");

  }

  Serial.println("Connected to the WiFi network");

}

void loop() {}

Servidor web en espera de conexión

#include <WiFi.h>

**const** **char**\* ssid = "tenda";

**const** **char**\* password = "phong707";

/\* create a server and listen on port 8088 \*/

WiFiServer **server**(**8088**);

**void** **setup**()

{

Serial.begin(**115200**);

Serial.print("Connecting to ");

Serial.println(ssid);

/\* connecting to WiFi \*/

WiFi.begin(ssid, password);

/\*wait until ESP32 connect to WiFi\*/

**while** (WiFi.status() != WL\_CONNECTED) {

delay(**500**);

Serial.print(".");

}

Serial.println("");

Serial.println("WiFi connected with IP address: ");

Serial.println(WiFi.localIP());

/\* start Server \*/

server.begin();

}

**void** **loop**(){

/\* listen for client \*/

WiFiClient client = server.available();

**uint8\_t** data[**30**];

**if** (client) {

Serial.println("new client");

/\* check client is connected \*/

**while** (client.connected()) {

**if** (client.available()) {

**int** len = client.read(data, **30**);

**if**(len < **30**){

data[len] = '\0';

}**else** {

data[**30**] = '\0';

}

Serial.print("client sent: ");

Serial.println((**char** \*)data);

}

}

}

}

Ejemplo de servidor web multi-instancia y de creación de páginas web.

#include <WiFi.h>

#include <FS.h>

#include <AsyncTCP.h>

#include <ESPAsyncWebServer.h>

const char\* ssid = "yourNetworkName";

const char\* password =  "yourNetworkPassword";

AsyncWebServer server1(80);

AsyncWebServer server2(81);

AsyncWebServer server3(82);

void setup(){

  Serial.begin(115200);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL\_CONNECTED) {

    delay(1000);

    Serial.println("Connecting to WiFi..");

  }

  Serial.println(WiFi.localIP());

  server1.on("/hello", HTTP\_GET, [](AsyncWebServerRequest \*request){

    request->send(200, "text/plain", "Hello from server 1");

  });

  server2.on("/hello", HTTP\_GET, [](AsyncWebServerRequest \*request){

    request->send(200, "text/plain", "Hello from server 2");

  });

  server3.on("/test", HTTP\_GET, [](AsyncWebServerRequest \*request){

    request->send(200, "text/plain", "Hello from server 3");

  });

  server1.begin();

  server2.begin();

  server3.begin();

}

void loop(){}

Ejemplo de Encriptacion/desencriptacion AES 128

//Cargamos las librerias necesarias para AES  
#include <AESLib.h>  
  
void setup()  
{  
  Serial.begin(9600); //Inicializamos el puerto Serie.  
    encriptar("top secret",128);  //Lo pongo aquí para que se ejecute una única vez  
  //desencriptar(cifrado,128);  
}  
  
void loop()  
{  
  
}  
  
  
String encriptar(String var, int bits)   
{   
  //Variable que se quiere encriptar y bits para encriptación (128, 256, etc)  
  //Función que devolverá encriptada el texto que contenga la variable  
  //de entrada como parámetro.  
  
  
  //El IV es un número normalmente aleatorio que se puede compartir  
  //y sirve para que un mismo texto encriptado no aparezca siempre  
  //de la misma forma, así es mas complicado desenciptarlo buscando  
  //patrones repetitivos.  
  
  
  uint8\_t key[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15};  
  //char data[] = "0123456789012345"; //16 chars == 16 bytes  
  
  
  // Largo del array (con un caracter extra para el caracter NULL)  
  int var\_len = var.length() + 1;  
  
  // Prarara array de carateres (el buffer)  
  char char\_array[var\_len];  
  
  // Convertimos la variable a array y lo volcamos en char\_array  
  var.toCharArray(char\_array, var\_len);  
    
  switch (bits)  
  {  
      
    case (128):  
      aes128\_enc\_single(key, char\_array);  
    break;  
      
      
    case (256):  
      aes256\_enc\_single(key, char\_array);  
    break;  
      
      
    default:  
      
      Serial.println(F("No se dió un parámtro correcto para bits de encriptación"));  
    break;  
  }  
      
      
    
  Serial.print("Encriptado:");  
  Serial.println(char\_array);  
  return (char\_array); //Se devuelve el texto encriptado  
  
}  
  
String desencriptar(String var, int bits)   
{   
  //Variable que se quiere desencriptar y bits que se usaron para encriptarlo (128, 256, etc)  
  //Función que devolverá desencriptada el texto que contenga la variable  
  //de entrada como parámetro.  
  
  
  //El IV es un número normalmente aleatorio que se puede compartir  
  //y sirve para que un mismo texto encriptado no aparezca siempre  
  //de la misma forma, así es mas complicado desenciptarlo buscando  
  //patrones repetitivos.  
  
  
  uint8\_t key[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15};  
  //char data[] = "0123456789012345"; //16 chars == 16 bytes  
  
  
  // Largo del array (con un caracter extra para el caracter NULL)  
  int var\_len = var.length() + 1;  
  
  // Prarara array de carateres (el buffer)  
  char char\_array[var\_len];  
  
  // Convertimos la variable a array y lo volcamos en char\_array  
  var.toCharArray(char\_array, var\_len);  
    
    switch (bits)  
  {  
      
    case (128):  
      aes128\_dec\_single(key, char\_array);  
    break;  
      
      
    case (256):  
      aes256\_dec\_single(key, char\_array);  
    break;  
      
      
    default:  
      
      Serial.println(F("No se dió un parámtro correcto para bits de desencriptación"));  
    break;  
  }  
  
  Serial.print("Desencriptado:");  
  Serial.println(char\_array);  
  
  return (char\_array); //Se devuelve el texto desencriptado  
  
}

Ejemplo de escritura/lectura en la EEPROM de Arduino (0<addr<255):

<https://www.arduino.cc/en/Tutorial/EEPROMWrite>

EEPROM.write(addr, val);

Val=EEPROM.read(addr);