Habitación Robohealth

Contenido

Acceso Habitación

Ilustración 1: Habitación construida

Se ha introducido una cámara y dado acceso en IP pública al controlador Vera, para prácticas en remoto:

Vera IP Pública: http://138.100.100.124
Cámara Habitación: http://138.100.100.142

- Usuario: *admin*
- Password: *rh_cam*

Control inalámbrico de la cama articulada

La cama articulada tiene un mando conectado mediante cable que permite subir y bajar tanto el cabecero como el pie de la cama. Para controlarla también de manera inalámbrica se ha diseñado e incorporado un sistema compuesto por relés y una placa basada en un Shield ESP32 que ejerce de pequeño servidor web. Éste espera mediante Wifi las órdenes de bajar o subir tanto la parte superior de la cama como la inferior, sin quitar la misma funcionalidad al mando original que posee.

En primer lugar se desmontó el mando de la cama y se realizó un esquema de su funcionamiento (Ilustraciones 2 y 3):

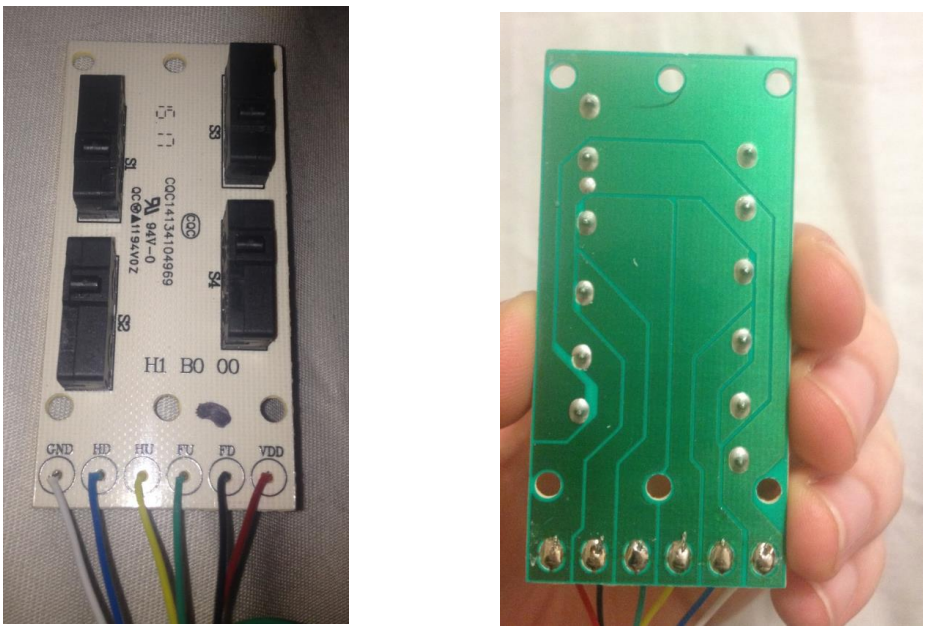

Ilustración 2 : Interior del mando

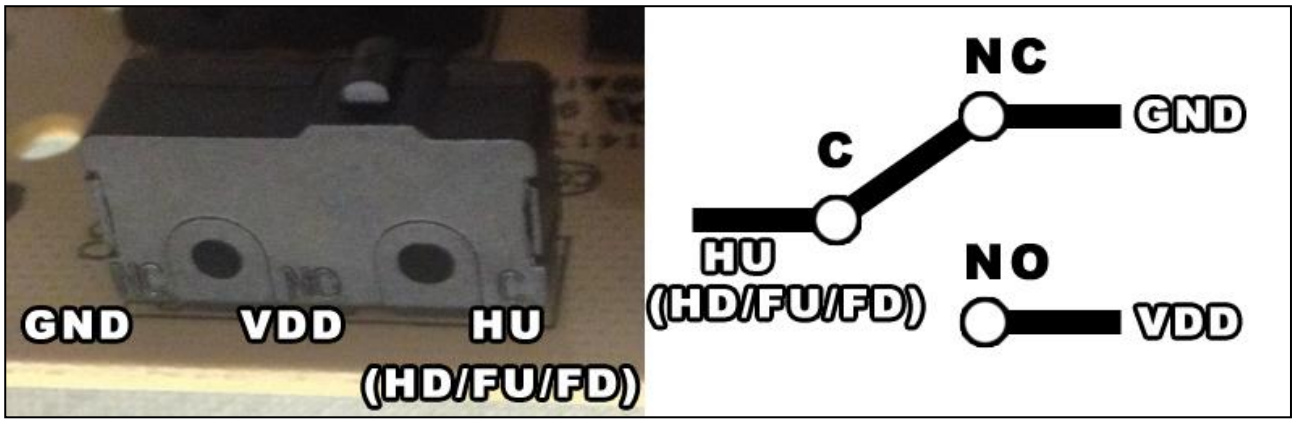

Ilustración 3: Esquema Switch

A continuación se diseñó el sistema objetivo mediante el que se controlarán los motores de la cama. Dicho sistema estará colocado entre la clavija hembra del inferior de la cama y la clavija macho del extremo del mando original. Mediante 4 relés se podrán controlar los motores, simulando el acto de presionar botones en el mando. Estos relés estarán controlados por el Shield ESP32, que tiene capacidad de comunicación Wifi y Bluetooth. Como indica el siguiente esquema (Ilustración 4), si los relés no están activados (motor moviéndose debido a una orden Wifi), el mando mantiene su funcionalidad intacta.

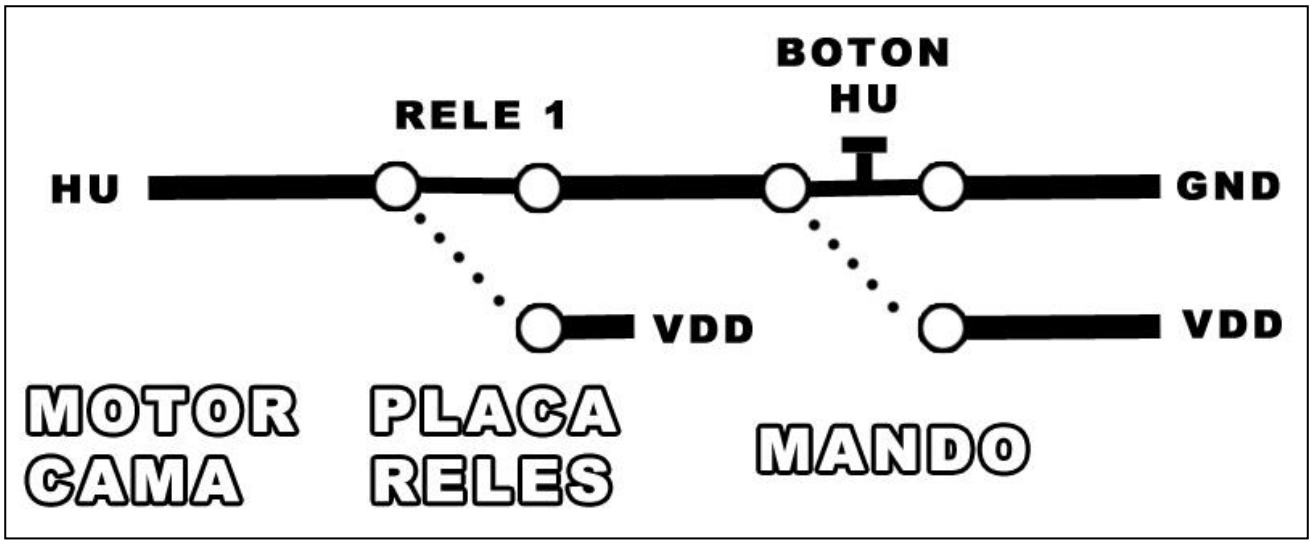

Ilustración 4: Esquema cableado para la función Head Up

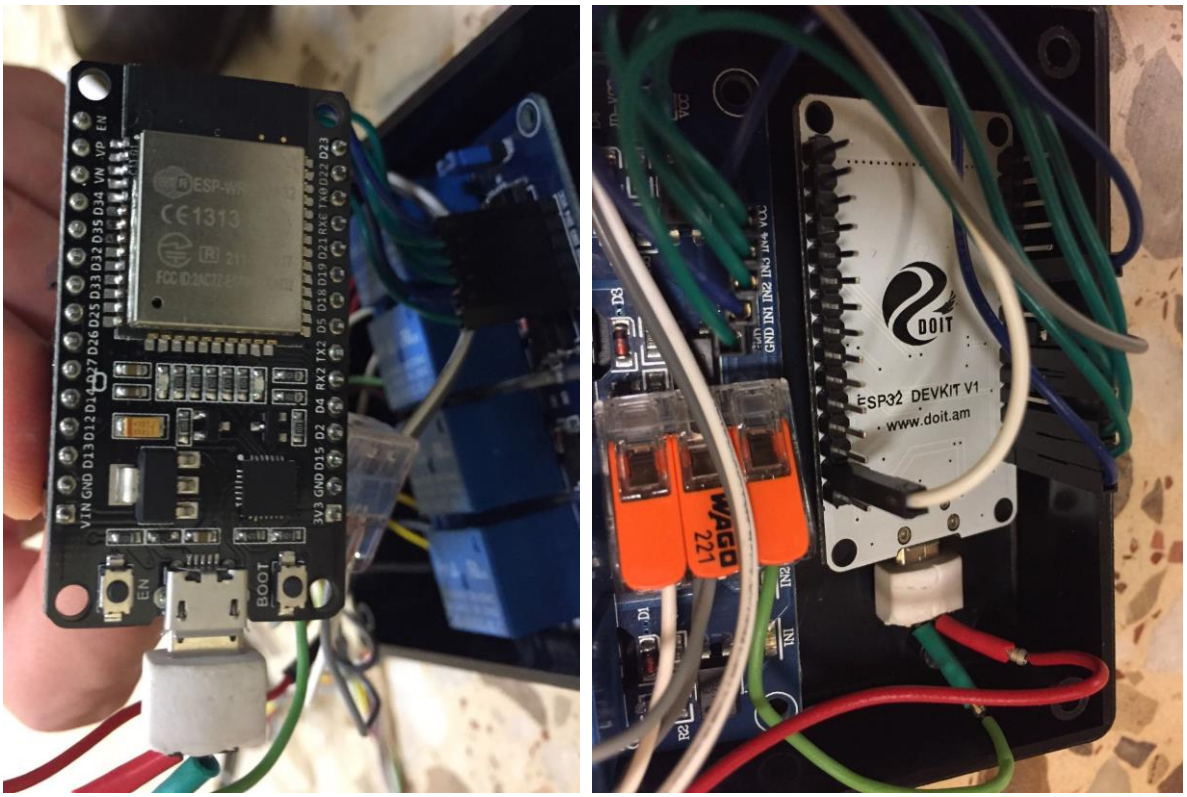

Ilustración 5: Shield ESP32 Devkit V1 y conexión con placa de 4 relés

La placa ESP32 se alimenta gracias a un regulador DC-DC (OKI-78SR) que aprovecha la alimentación de los motores de la cama. También se ha incluido a modo de test un led rojo que indica si la placa ha conseguido conectarse al Wifi “siemens” para poder funcionar, en cuyo caso el led permanecerá encendido.

En el Anexo del final de este documento se encuentra el código listo para introducirlo en el Shield ESP32 mediante el IDE de Arduino.

Interfaz Node-RED

Se ha realizado también un prototipo de interfaz de usuario híbrida que incluye la visualización y control de los dispositivos ZWave, así como otros de diferente protocolo como 6LoWPAN o Wifi. La interfaz estará basada en la herramienta de programación Node-RED, que permite unificar dispositivos físicos, APIs y servicios online de forma sencilla e intuitiva. Dispone de un editor accesible mediante navegador y la programación basada en Node.js se realiza mediante nodos interconectados. El sistema que ejecuta Node-RED (preinstalado en Raspbian GNU/Linux 9 "stretch") es una Raspberry Pi 3 Model B.

Para que Node-RED se ejecute automáticamente desde el encendido, basta con ordenar en la terminal:

```
sudo systemctl enable nodered.service
```

➔ Link de interés: https://nodered.org/docs/hardware/raspberrypi

La Raspberry está por defecto conectada a la red Wifi "Siemens" del laboratorio de robótica donde está situada la habitación de hospital. Dentro de la red, su dirección IP es: 192.168.1.144

Si tenemos conectado un ordenador a la misma red, tendremos acceso al Node-RED de la Raspberry, al cual se accede por defecto en el puerto 1880. Por ello, si entramos desde nuestro navegador a 192.168.1.144:1880 (ACTUALIZACIÓN: temporalmente se ha puesto la ip pública 138.100.100.140:1880) podemos ver y editar los flujos de esta herramienta, y si añadimos /ui al final, entraremos en la interfaz de usuario que hemos configurado: 192.168.1.144:1880/ui (138.100.100.140:1880/ui)

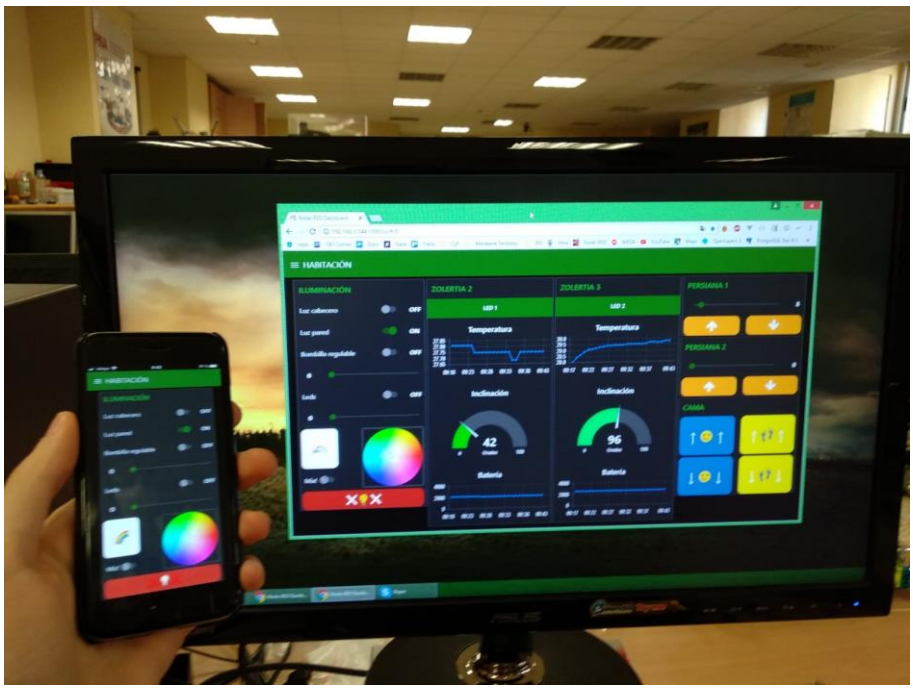

Ilustración 4: Interfaz de usuario accesible desde diferentes dispositivos

Para proteger los flujos programados, se ha establecido una contraseña que proteja el modo edición:

usuario: *admin*
contraseña: *roboraspy*

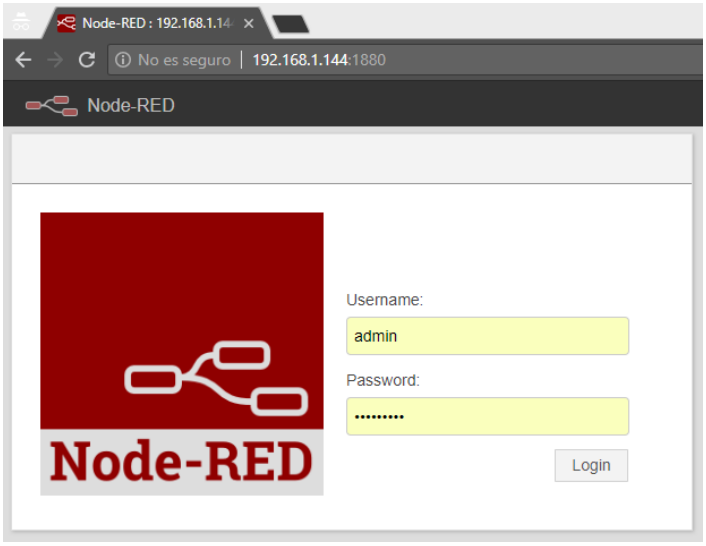

Ilustración 5: Entrada a la configuración de flujos de Node-RED

Se han desarrollado cinco secciones o flujos de Node-RED para este proyecto:

Manejo Raspberry Pi

- El primero de ellos corresponde al **manejo de la Raspberry**, en concreto a la monitorización de su CPU y temperatura, así como la opción de apagado remoto de la placa.

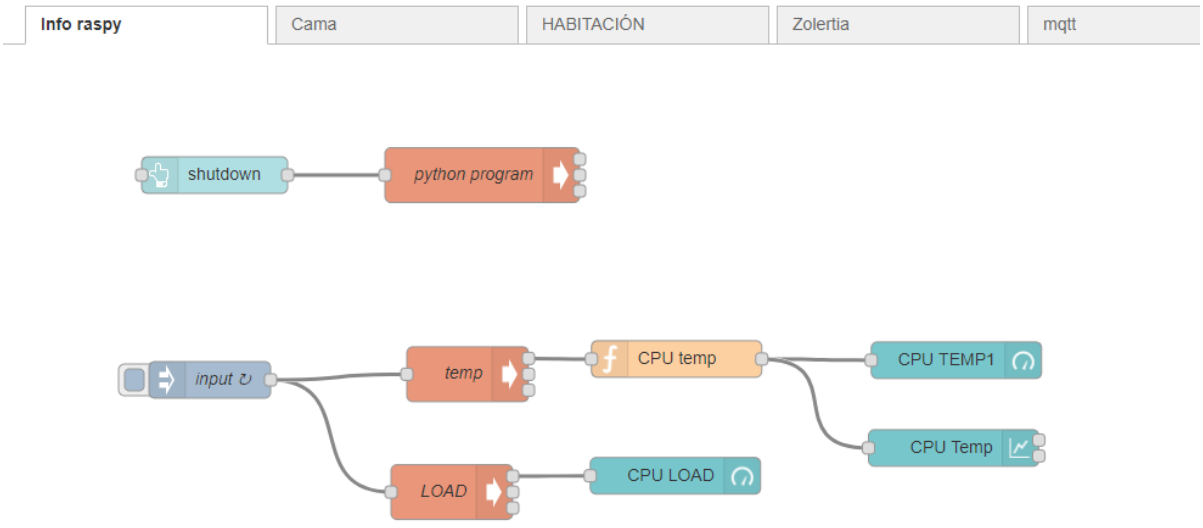

Ilustración 6: Flujo info raspy

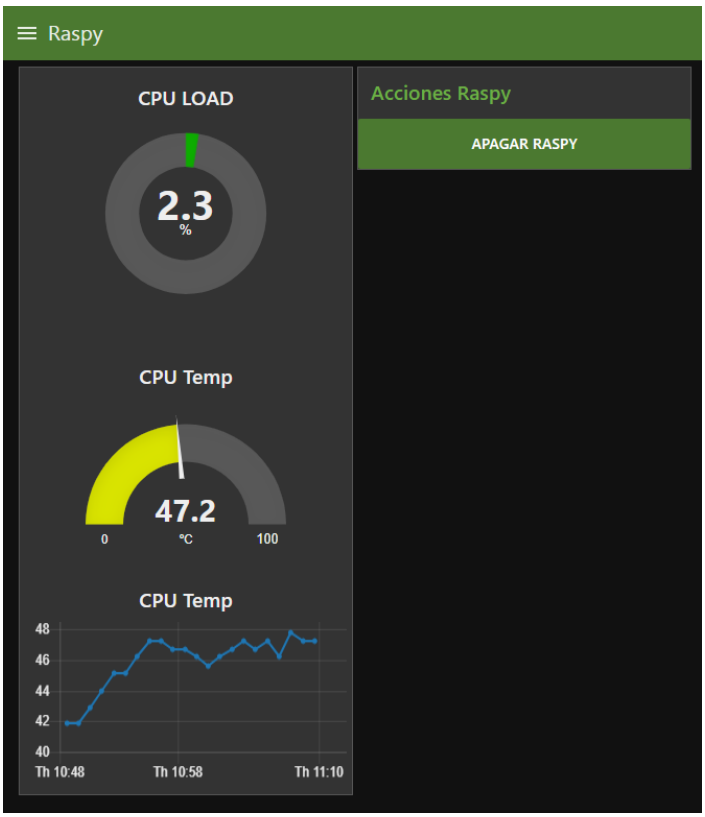

Ilustración 7: UI info raspy

Control ZWave

- El segundo controla los **dispositivos físicos de los que se compone la instalación ZWave**, así como la visualización de sus datos correspondientes. La instalación consta de una serie de sensores y actuadores unificados y comandados por un VeraLite Smart Controller, uno de los controladores domóticos más potentes del mercado y utilizado a su vez por su sencillez de conexión prácticamente Plug & Play.

En este flujo se obtienen datos de los dispositivos de iluminación, persianas y demás sensores para mostrar su estado con la tasa de refresco que se estime adecuada, y también permite controlarlos mediante la interfaz desarrollada. Se ha partido del Trabajo Fin de Máster realizado por Pablo Vicente García, mejorando la interactividad de los botones de la interfaz con el estado real de los elementos en la habitación, rediseñando los pulsadores con plantillas html y Angular, así como añadiendo las persianas y la tira de leds RGB instalada en la habitación. Aparte de estar también controlados por Z-Wave, se han incorporado pulsadores físicos dentro de la habitación para su manejo.

Para ordenar el encendido, apagado o valor concreto a un elemento Z-Wave de la habitación, se hará uso de peticiones GET a las direcciones proporcionadas por el controlador Vera. Por ejemplo, para encender la bombilla del cabecero (Device = 12), se le especifica al Vera una acción con Target Value = 1 (ON):

```
http://138.100.100.124:3480/data_request?id=action&output_format=xml&DeviceNum=12&serviceId=urn:upnp-org:serviceId:SwitchPower1&action=SetTarget&newTargetValue=1
```

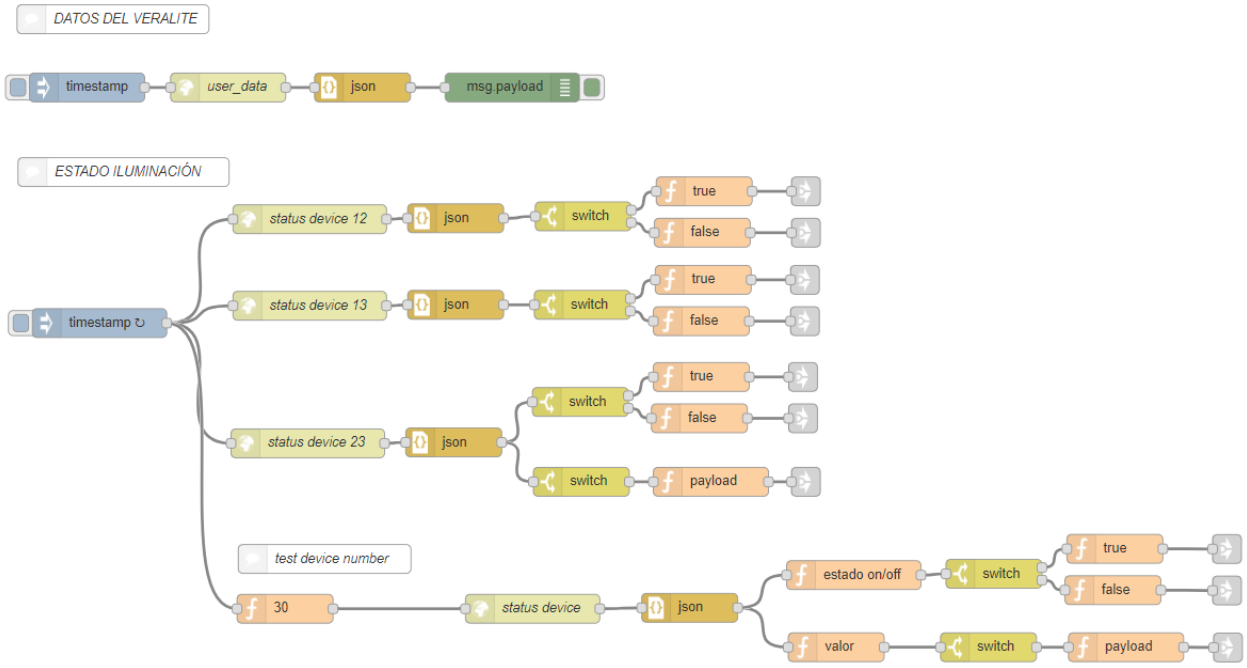

Ilustración 8: Flujo habitación -> obtención de estados

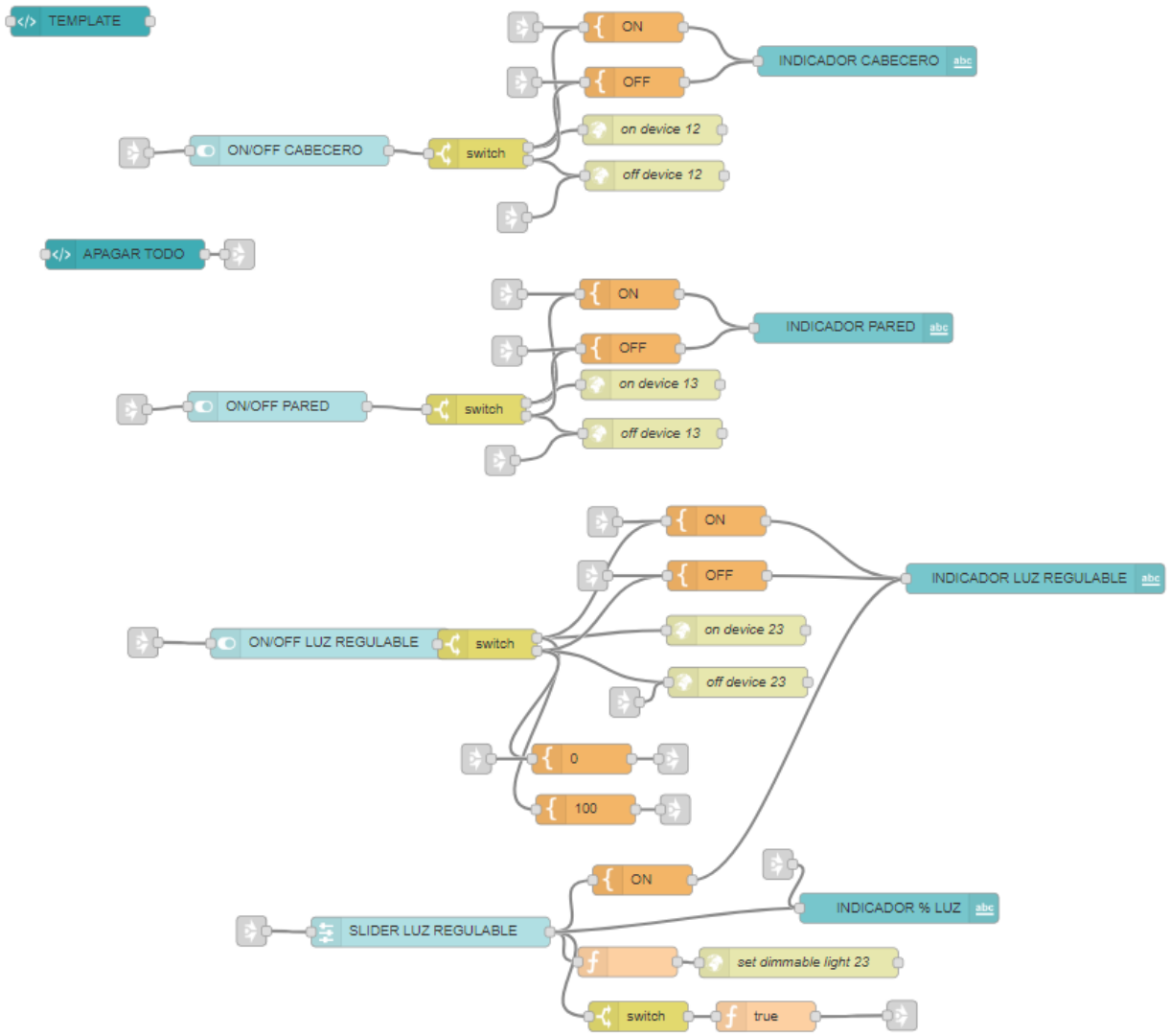

Ilustración 9: Flujo habitación -> control de estados

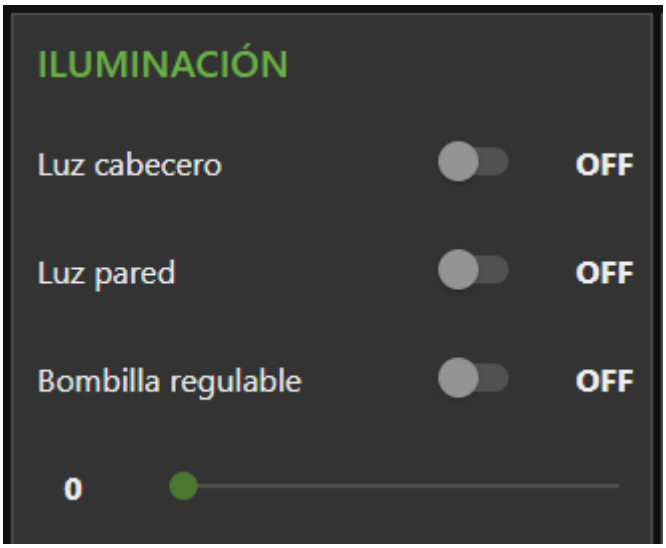

Ilustración 10: UI iluminación

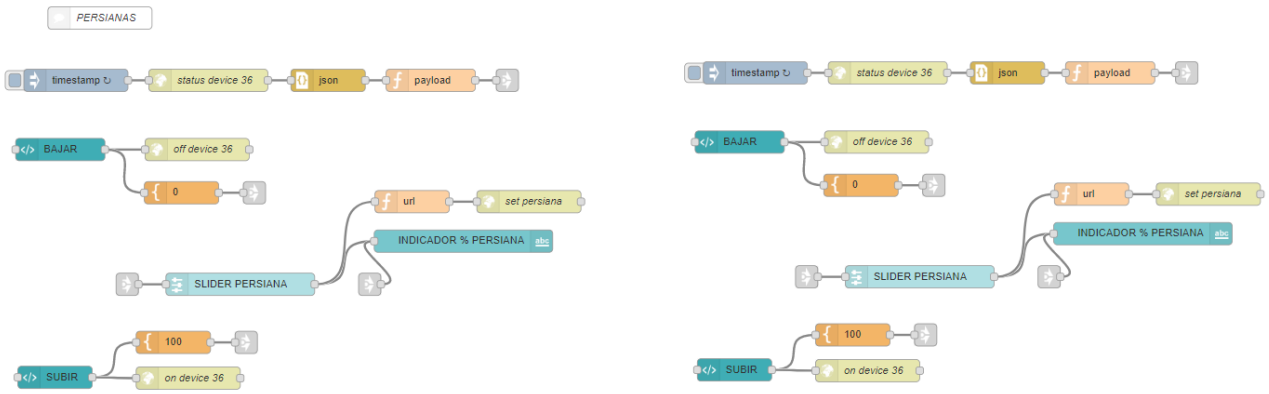

Ilustración 11: Flujo habitación -> Control de persianas

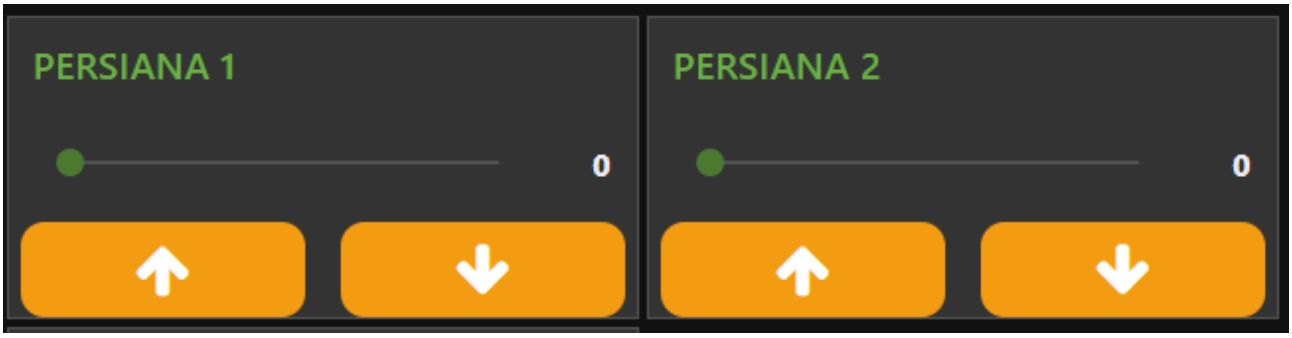

Ilustración 12: UI persianas

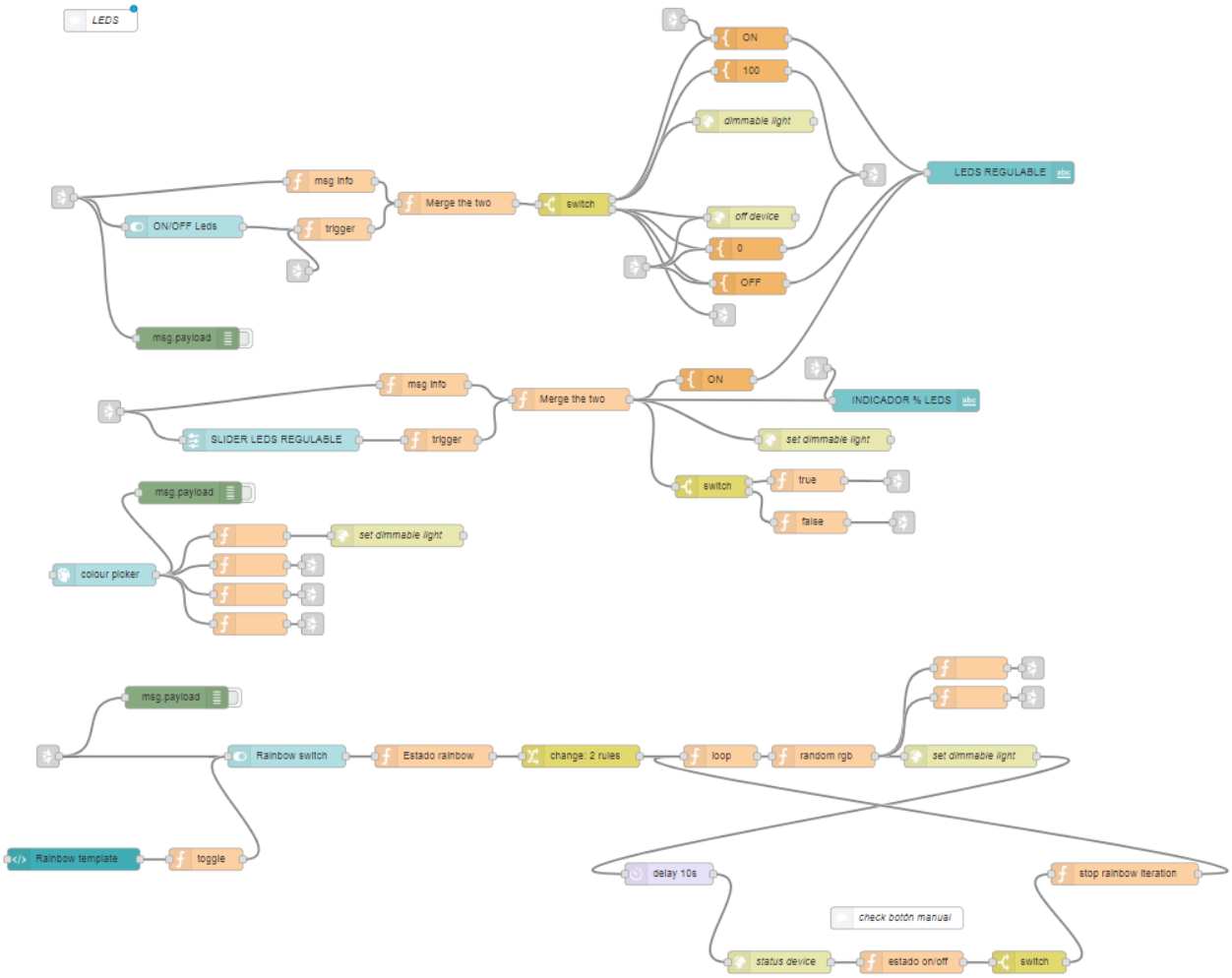

Ilustración 13: Flujo habitación -> Control de leds RGB y botón Rainbow

Ilustración 14: UI Leds RGB

Actuadores cama articulada

- El tercer flujo se compone de los **actuadores para la cama articulada**. Para controlarla de manera inalámbrica se ha diseñado e incorporado un sistema compuesto por relés y una placa basada en un Shield ESP32 que ejerce de pequeño servidor web, como se describió anteriormente. Así, de manera parecida a los dispositivos Z-Wave, ordenamos mediante peticiones GET la subida o bajada del cabecero o los pies de la cama.

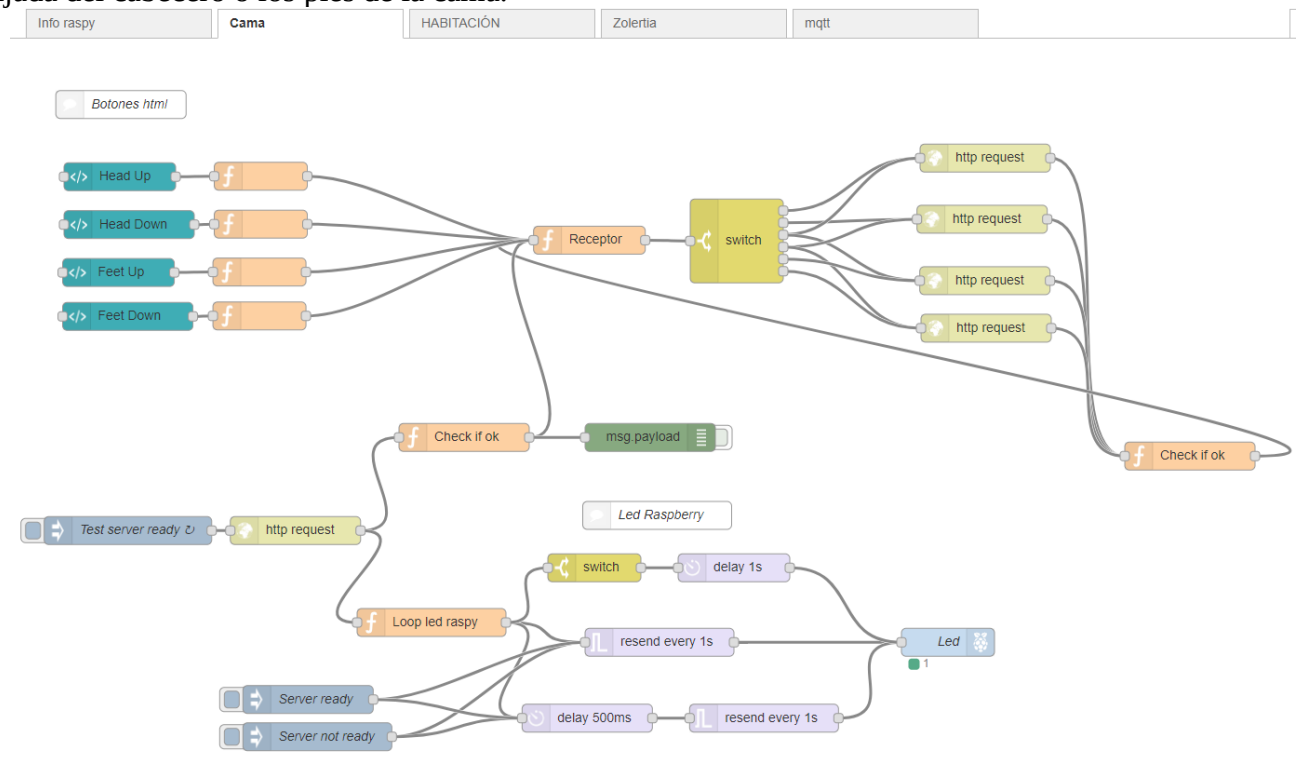

Ilustración 15: Flujo Cama

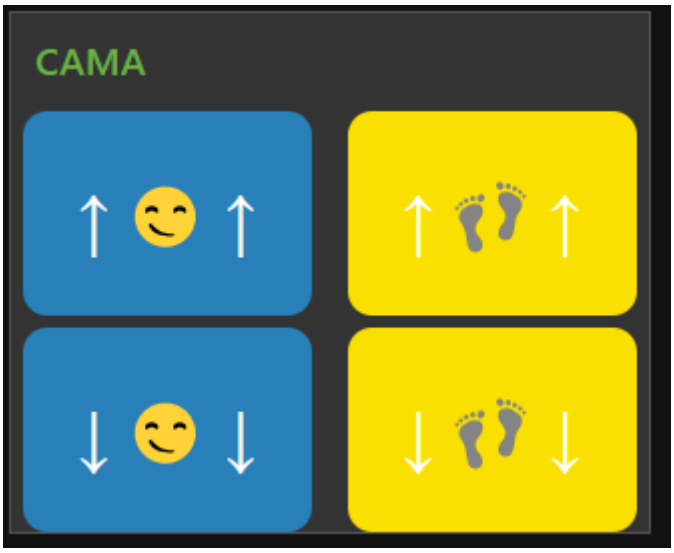

Ilustración 16: UI Cama

Dispositivos Zolertia

- El cuarto incorpora los **dispositivos Zolertia** con los que se miden datos como la **temperatura o la inclinación de la cama** gracias a los sensores que incorporan. Estos módulos se comunican entre sí mediante 6LoWPAN/IPv6. Este protocolo permite a pequeños dispositivos manejar pilas IPv6, simplificando los dominios de red y haciendo así más sencilla la distribución y mantenimiento de dichas redes, pudiendo utilizar en una capa superior protocolos como UDP, TCP, HTTP, MQTT, CoAP y otros. Dos de estos módulos Zolertia forman parte de una red sensórica, mientras que el tercero (conectado a la Raspberry) ejerce de *Border Router* ó encargado de conectar la malla a internet.

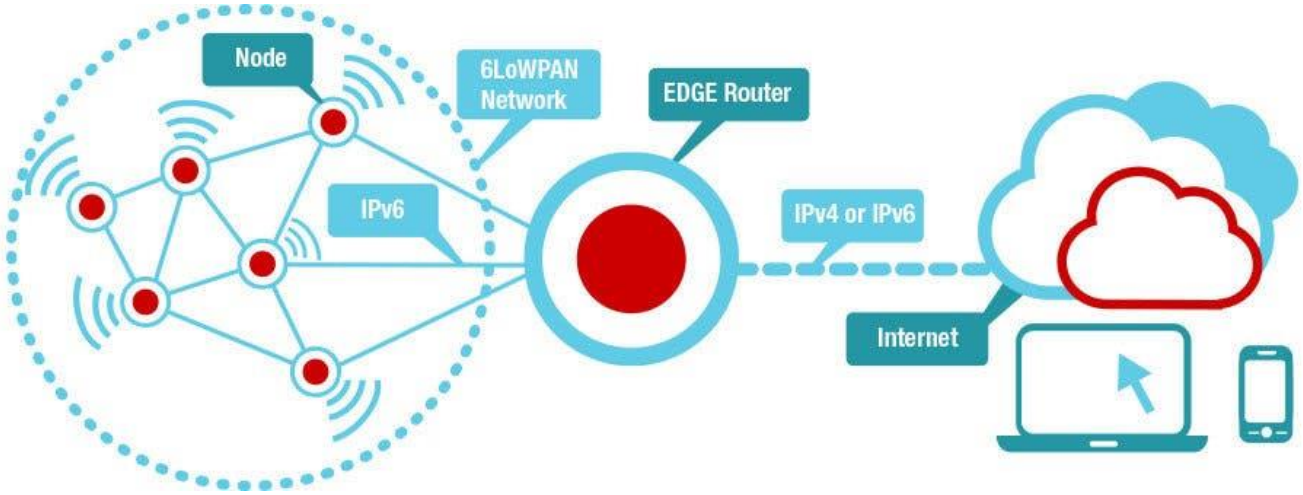

Ilustración 17: Red 6LOWPAN utilizada

Para la programación y conexión de las motas Zolertia, resultaron de ayuda 3 documentos:

- Documento de Juan Fernando Tenorio Galván (alumno del Grado en Ingenieria en Sistemas Computacionales sobre Red con "Motas Zolertia Z1" bajo los protocolos 6LoWPAN, CoAP y MQTT)
- Ejemplo de una red Zolertia Z1 y Raspberry: https://www.hackster.io/4348/zolertia-ipv6-6lowpan-ubidots-bcde84
- Documentación detallada sobre Internet of Things, en particular sobre IPv6 y Zolertia: https://github.com/marcozennaro/IPv6-WSN-book/blob/master/Releases/IoT%20in%20five%20days%20-%20v1.1%2020160627.pdf

La mota nº 1 tiene cargado el código de /home/pi/contiki/examples/ipv6/rpl-border-router/ y ejercerá por ello de Border Router. Para que esté operativo automáticamente, insertaremos en la Raspberry una tarea cron que lanzará el comando de conexión cada vez que se encienda.

```
crontab -e
```

```
@reboot make -C /home/pi/contiki/examples/ipv6/rpl-border-router/ connect-router
MOTES=/dev/ttyUSB0 &
```

Nota: Antes de encender la Raspberry, la mota 1 debe estar conectada a un puerto USB de la misma.

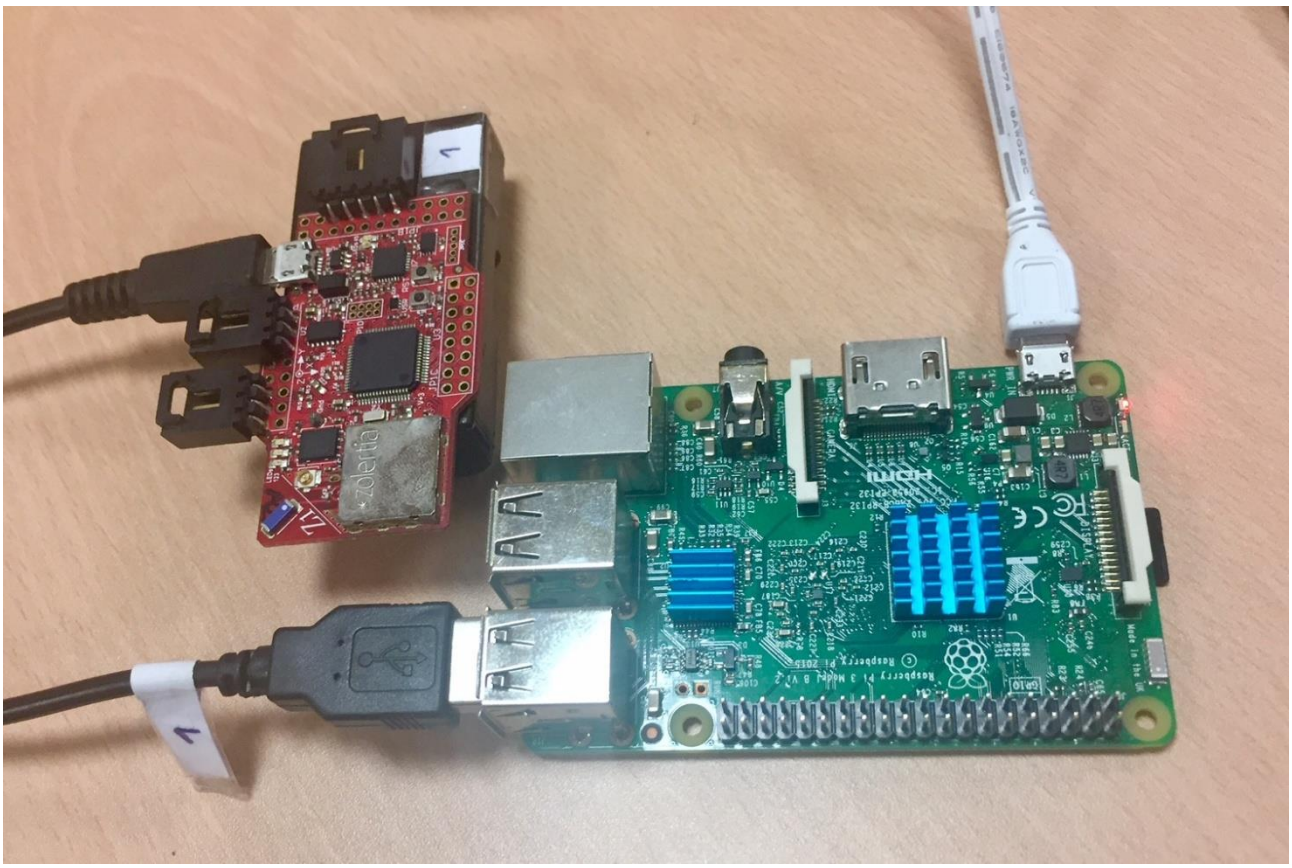

Ilustración 18: Mota 1 Zolertia conectada a USB de Raspberry Pi

Las motas 2 y 3 tienen el código /home/user/contiki/examples/er-rest-example, y con ellas sólo es necesario proporcionarles alimentación (pila, cable usb) para que comiencen a funcionar, buscando a la mota 1 y conectándose a ella. Si al inicio no se iluminan la 2 o 3, basta con pulsar el botón RST (Ilustración 19)

Ilustración 19: Mota 1 colocada en cabecero; Ubicación botón Reset

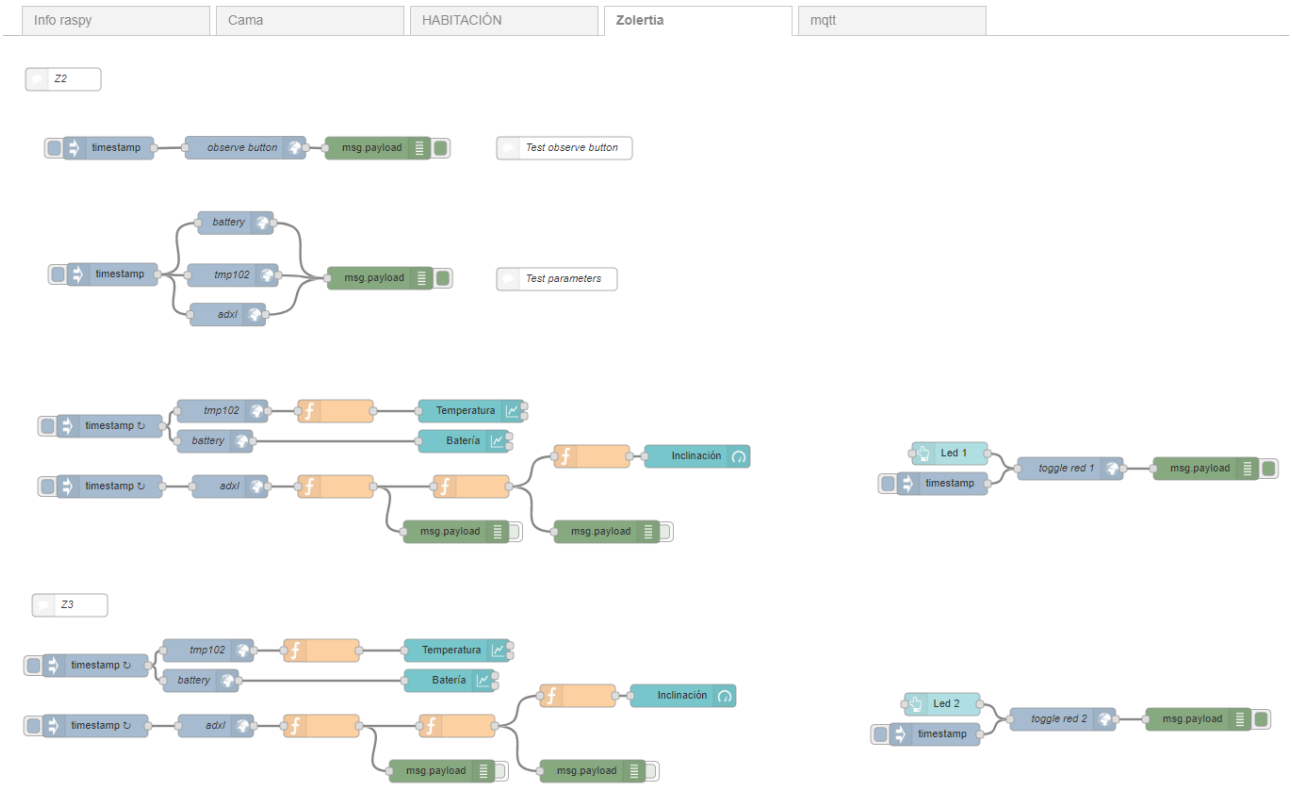

Ilustración 20: Flujo Zolertia

La frecuencia con la que se actualizan los datos en la interfaz se puede cambiar en el editor de Node-RED. Actualmente se hace cada 5 segundos la inclinación y cada 10 minutos la temperatura y batería.

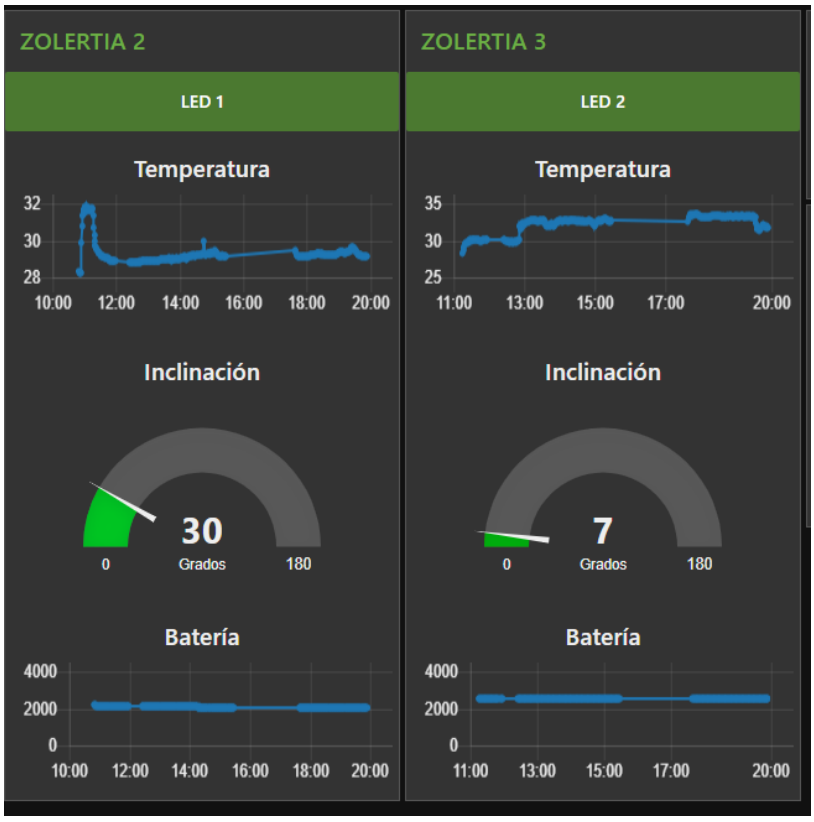

Ilustración 21: UI Zolertia

Suscripción y publicación MQTT

- El último flujo de nodos corresponde a la **gestión de mensajes MQTT**. Éste se encarga de monitorizar todos los dispositivos de la habitación y enviar a los suscriptores del topic: *Robohealth/room/devices* los cambios de estado que ocurran en cada elemento. El objetivo es que desarrollos actuales como la app Windows puedan suscribirse a sus mensajes y obtengan así la información de la sala, liberando a la tablet de realizar dicha tarea en peticiones periódicas.

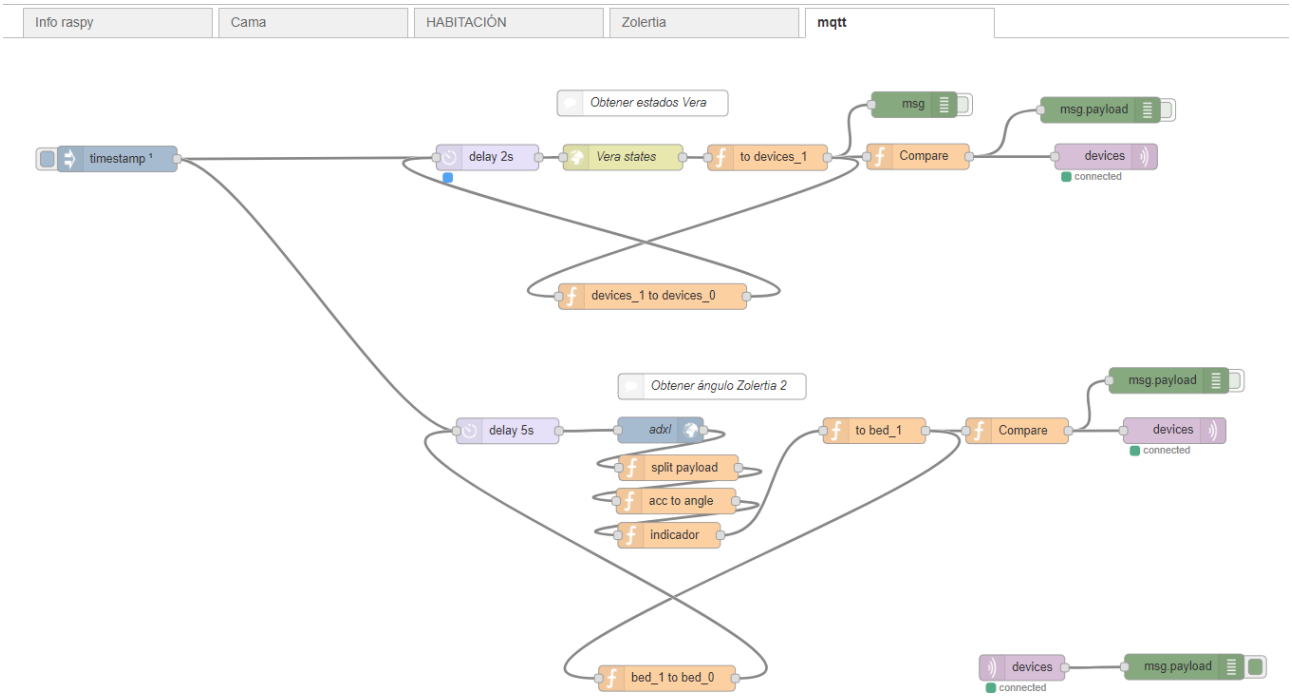

Ilustración 22: Flujo MQTT

Actualización: Aparte de escuchar, se ha añadido la capacidad de dar órdenes mediante MQTT, por lo que también se permite cambiar el estado de las bombillas y persianas. Bastaría con publicar un mensaje al topic *Robohealth/room/devices*:

- Bombillas: {"id":12, status":"1"}
- Bombilla y persianas regulables: {"id":23,"level":"15","status":"1"}

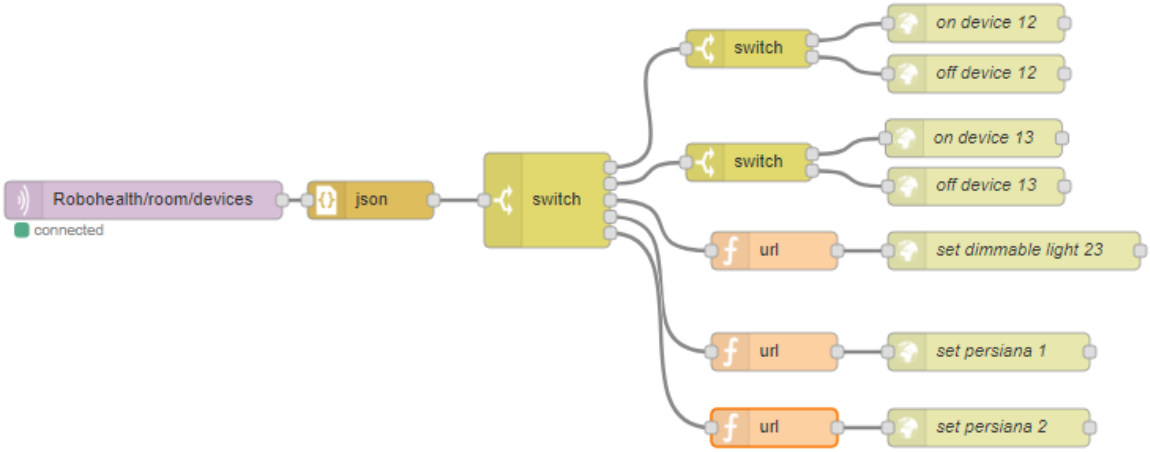

Ilustración 23: Flujo MQTT -> Actuador

Este sistema MQTT de suscripción y publicación se puede comprobar mediante cualquier programa (MQTTfx) o app (MQTTool). Se precisa indicar la ip del host, junto con el puerto estándar 1883 y la autenticación de usuario:

- Username: pi
- Password: roboraspy

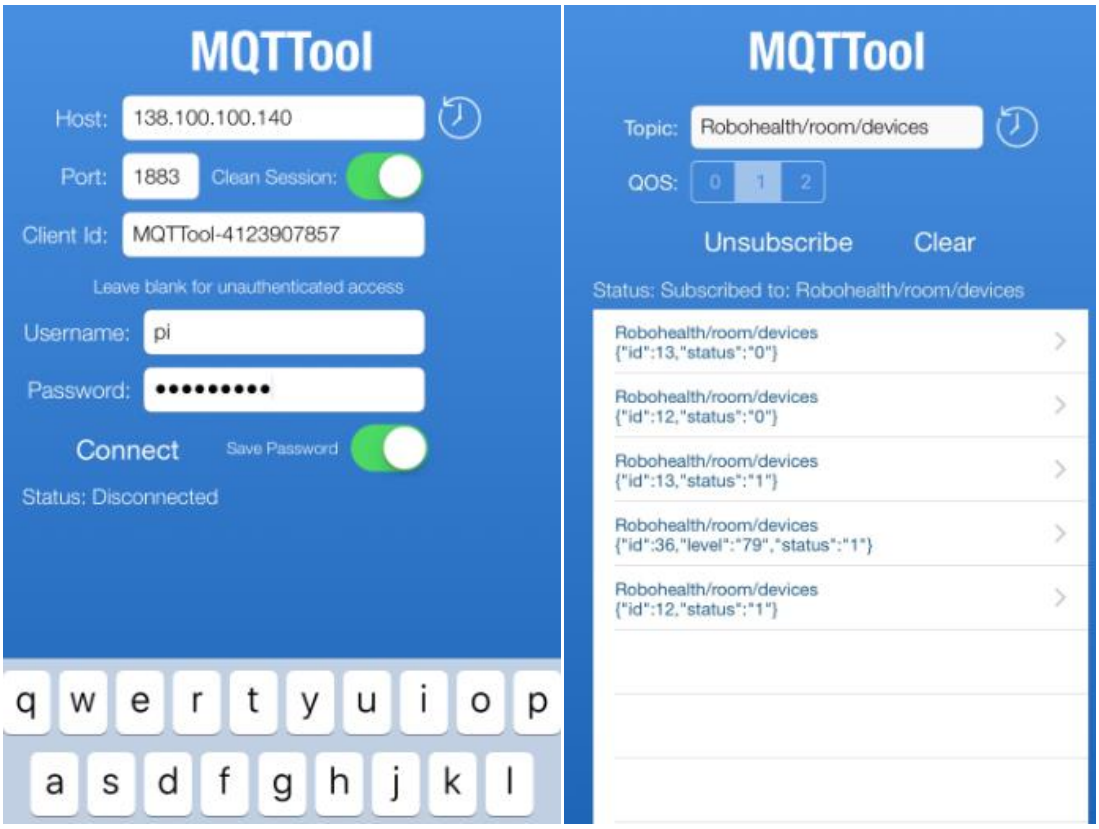

Ilustración 24: App demostrativa MQTTool (iOS)

Ilustración 25: UI Final Node-RED

Anexo

Código cama articulada (ESP32)

```
/*
  WiFi Web Server

  A simple web server that lets you blink an LED via the web.
  This sketch will print the IP address of your WiFi Shield (once connected)
  to the Serial monitor. From there, you can open that address in a web browser
  to turn on the selected pin.

  If the IP address of your shield is yourAddress:
  http://yourAddress/HU turns the pin on for 1300 ms

  This example is written for a network using WPA encryption. For
  WEP or WPA, change the Wifi.begin() call accordingly.

*/

#include <WiFi.h>

#define HU 32
#define HD 27
#define FU 13
#define FD 14
#define LED_WIFI 26

const char* ssid     = "siemens";
const char* password =  "proy**********";

// NETWORK: Static IP details...
IPAddress ip(192, 168, 1, 83);
IPAddress gateway(192, 168, 1, 1);
IPAddress subnet(255, 255, 255, 0);

int connected = 0;


WiFiServer server(80);
```

```
void setup()
{
  Serial.begin(115200);
  pinMode(LED_WIFI, OUTPUT);      // set the LED pin mode
  pinMode(HU, OUTPUT);      // set the LED pin mode
  pinMode(HD, OUTPUT);      // set the LED pin mode
  pinMode(FU, OUTPUT);      // set the LED pin mode
  pinMode(FD, OUTPUT);      // set the LED pin mode

  digitalWrite(HU, HIGH);
  digitalWrite(HD, HIGH);
  digitalWrite(FU, HIGH);
  digitalWrite(FD, HIGH);

  // We start by connecting to a WiFi network

  Serial.println();
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  // Static IP Setup Info Here...
  WiFi.config(ip, gateway, subnet);

  // Start Server
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
    digitalWrite(LED_WIFI, LOW);
  }



  connected = 0;
  Serial.println("");
  Serial.println("WiFi connected.");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
  digitalWrite(LED_WIFI, HIGH);
```

```
  server.begin();
  delay(500);

}

void reconnectWifi() {
  WiFi.begin(ssid, password);
}

/*void onDisconnected(const WiFiEventStationModeDisconnected& event)
    {
        connected = 0;
        reconnectWifi();
    }
*/

int value = 0;
//int reset_loops = 0;

void loop() {
  /*delay(100);
  digitalWrite(LED_WIFI, LOW);
  delay(100);
  digitalWrite(LED_WIFI, HIGH);*/
  delay(50);
  WiFiClient client = server.available();   // listen for incoming clients
  Serial.println(client);
  delay(50);
  if (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
    Serial.println(WiFi.status());
    digitalWrite(LED_WIFI, LOW);
    //esp_restart();
  }

  if (client) {                             // if you get a client,
    Serial.println("New Client.");           // print a message out the serial port
    String currentLine = "";                // make a String to hold incoming data from
the client
    while (client.connected()) {            // loop while the client's connected
```

```
      if (client.available()) {             // if there's bytes to read from the
client,
        char c = client.read();             // read a byte, then
        Serial.write(c);                    // print it out the serial monitor
        if (c == '\n') {                    // if the byte is a newline character

          // if the current line is blank, you got two newline characters in a row.
          // that's the end of the client HTTP request, so send a response:
          if (currentLine.length() == 0) {
            // HTTP headers always start with a response code (e.g. HTTP/1.1 200 OK)
            // and a content-type so the client knows what's coming, then a blank line:
            client.println("HTTP/1.1 200 OK");
            client.println("Content-type:text/html");
            client.println();

            // the content of the HTTP response follows the header:
            client.print("<a href=\"/HU\"><button style=\"width:300px;height:200px;
background: -webkit-gradient(linear, 0% 0%, 0% 50%, from(white), to(blue));
color:white; font-size:40px\">Cabeza Arriba<br>&#8593&#8593</button></a>");
            client.print("<a href=\"/HD\"><button style=\"width:300px;height:200px;
background: -webkit-gradient(linear, 0% 0%, 0% 50%, from(white), to(blue));
color:black; font-size:40px\">Cabeza Abajo<br>&#8595&#8595</button></a><br>");
            client.print("<a href=\"/FU\"><button style=\"width:300px;height:200px;
background: -webkit-gradient(linear, 0% 0%, 0% 50%, from(white), to(red)); color:white;
font-size:40px\">&#8593&#8593<br>Pies Arriba</button></a>");
            client.print("<a href=\"/FD\"><button style=\"width:300px;height:200px;
background: -webkit-gradient(linear, 0% 0%, 0% 50%, from(white), to(red)); color:black;
font-size:40px\">&#8595&#8595<br>Pies Abajo</button></a>");

            // The HTTP response ends with another blank line:
            client.println();
            // break out of the while loop:
            break;
          } else {    // if you got a newline, then clear currentLine:
            currentLine = "";
          }
        } else if (c != '\r') {  // if you got anything else but a carriage return
character,
          currentLine += c;      // add it to the end of the currentLine
        }
```

```
        // Check to see if the client request was "GET /H" or "GET /L":
        if (currentLine.endsWith("GET /HU")) {
          digitalWrite(HU, LOW);               // GET /H turns the LED on
          delay(1300);
          digitalWrite(HU, HIGH);               // GET /H turns the LED on
        } else if (currentLine.endsWith("GET /HD")) {
          digitalWrite(HD, LOW);               // GET /H turns the LED on
          delay(1300);
          digitalWrite(HD, HIGH);               // GET /H turns the LED on
        } else if (currentLine.endsWith("GET /FU")) {
          digitalWrite(FU, LOW);               // GET /H turns the LED on
          delay(1300);
          digitalWrite(FU, HIGH);               // GET /H turns the LED on
        } else if (currentLine.endsWith("GET /FD")) {
          digitalWrite(FD, LOW);               // GET /H turns the LED on
          delay(1300);
          digitalWrite(FD, HIGH);               // GET /H turns the LED on
        }
      }
    }
    // close the connection:
    client.stop();
    Serial.println("Client Disconnected.\n");
  } else {
    //Serial.println(WiFi.status());

  }
}
```