

## Descripción del funcionamiento

Este proyecto está enfocado al monitoreo de una casa usando el chip ESP32, y de cómo a través de un dispositivo móvil se pueden controlar ciertas funciones comunes dentro de ella como lo serían el control de luces de la casa , la temperatura de sus ambientes y el uso de una alarma. También aplicaremos, los programas que hemos utilizado para la realización de este proyecto, como también los materiales a utilizar y el beneficio que nos puede dar.

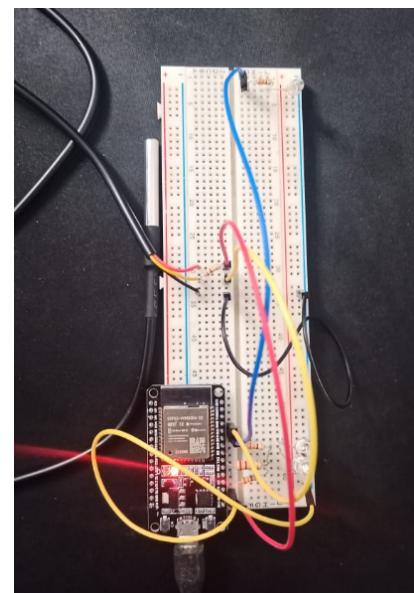
### ¿De qué trata?

Primero expliquemos un concepto básico, **¿qué es la domótica?** Se refiere a la integración de las distintas tecnologías en el hogar mediante el uso simultáneo de electricidad, electrónica, informática y telecomunicaciones.

A través del chip ESP32 controlaremos las luces LED simulando los focos de una casa mediante una interfaz de control web, así como también se medirá la temperatura de cada habitación de la casa haciendo uso del sensor de temperatura DS18B20 y se contará con una alarma simulada a través de un Buzzer activo de 12mm

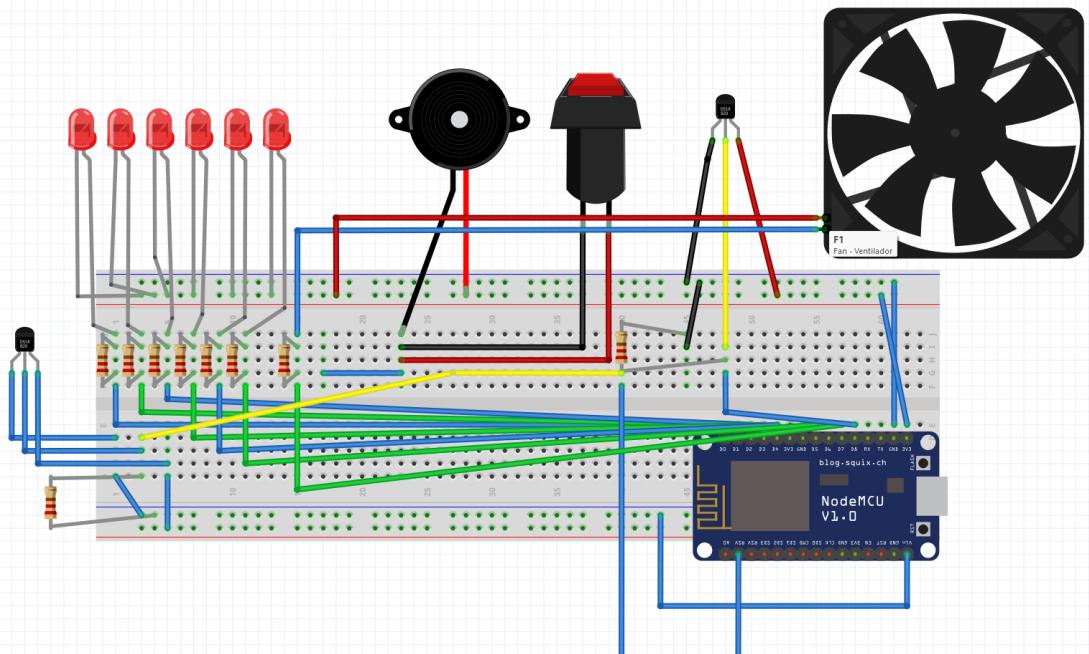
### Materiales y Características:

- Chip ESP32
  - 520 Kb de RAM.
  - Wifi integrado
  - Compatible con Arduino
  - Se puede programar con límite de entrada de 1V ,2V y 4V
- Cables Macho-Macho
  - 20 centímetros de longitud.
  - Excelente conductividad eléctrica.
  - Colores variados en el arnés.
  - 40 Piezas por arnés.
  - Conector Dupont Macho en ambos extremos.
- Protoboard
  - Puntos de contacto: 400
  - Permite detectar errores de diseño.
  - Permite la conexión de los componentes
  - Cable soportado: 29-20AWG ( $\varnothing$  0.3 a 0.8 mm aprox.)
  - Espaciamiento estándar entre pines de 0.1"(2.54mm)

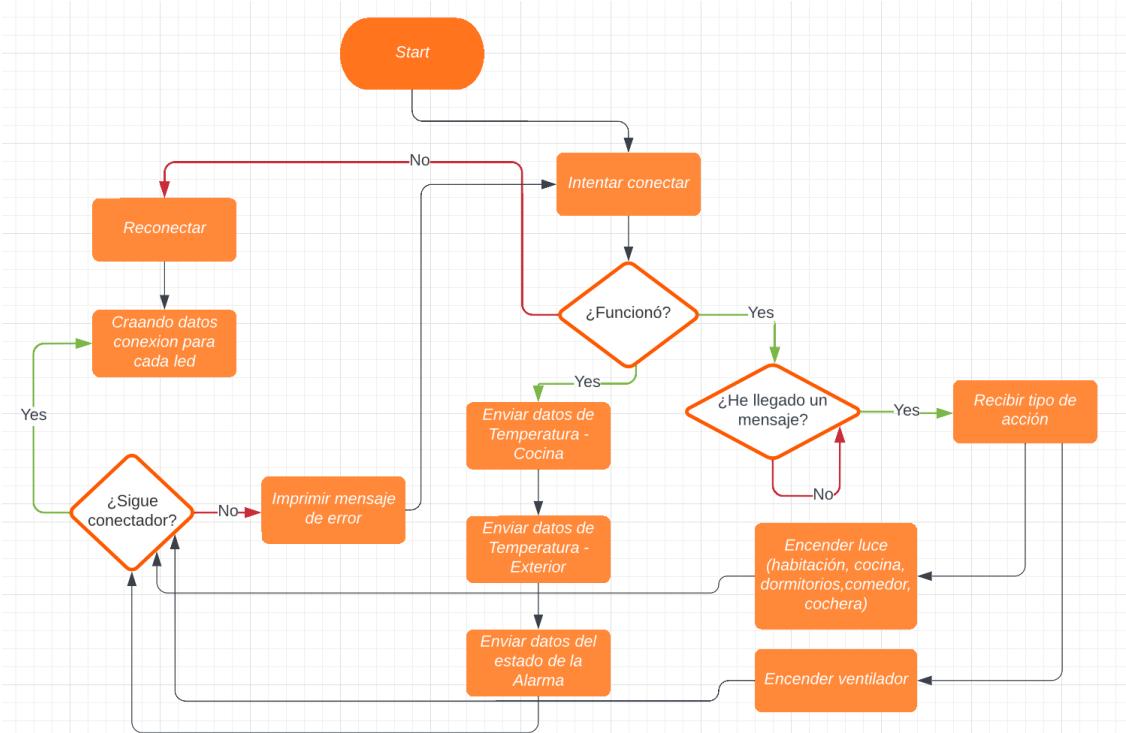


- Resistencia 330 ohm
  - Potencia máxima de ¼ W
  - Potencia de disipación: 0,25 vatios
  - Tolerancia: 5%
  - Temperatura de operación: -55 a 20°
- Resistencia 10k ohm
  - Potencia: 0.25W
  - Rango de resistencia: de 1E a 22M (serie E12)
  - Temperatura de funcionamiento: de -55°C a +155°C
  - Tolerancia: 5%
  - Tensión de funcionamiento: 250V máx
- Sensor de temperatura DS18B20
  - Mide temperatura desde los -55°C hasta los 125°C
  - Precisión +- 0.5 grados.
  - Protocolo OneWire
  - Precisión: ±0.5°C (de -10°C a +85°C)
  - Resolución: de 9 a 12 bits (configurable)
- Diodo Led x6
  - Trabajan a corriente de 2V
  - Permite el paso de corriente en un solo sentido
  - Temperatura 40°C a 65°C
- Ventilador(potenciado por motor de carrito)
  - Hecho a mano, 2.3 cm cada hélice
- Buzzer Activo 12mm
  - Voltaje de operación: 3.5V to 5.5V
  - Frecuencia: 2300Hz +/- 200Hz Min.
  - Temperatura de operación:-30'C a 85'C.
  - Salida de sonido @ 10cm: 90dB.

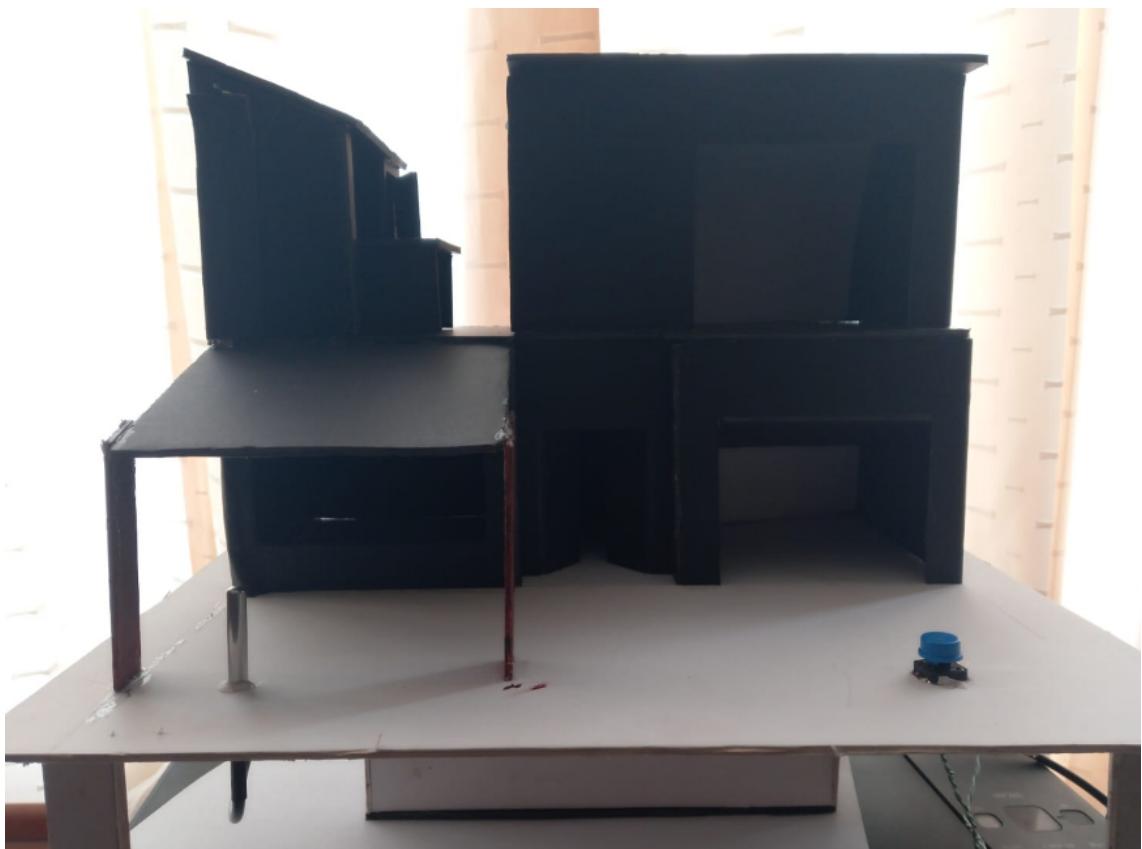
## Diagrama de circuito:



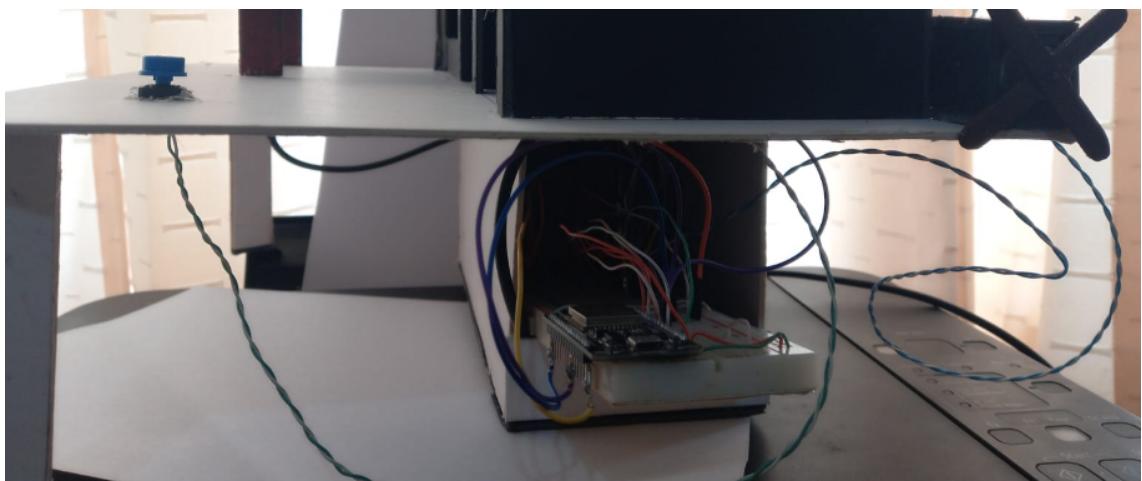
## Diagrama de flujo:

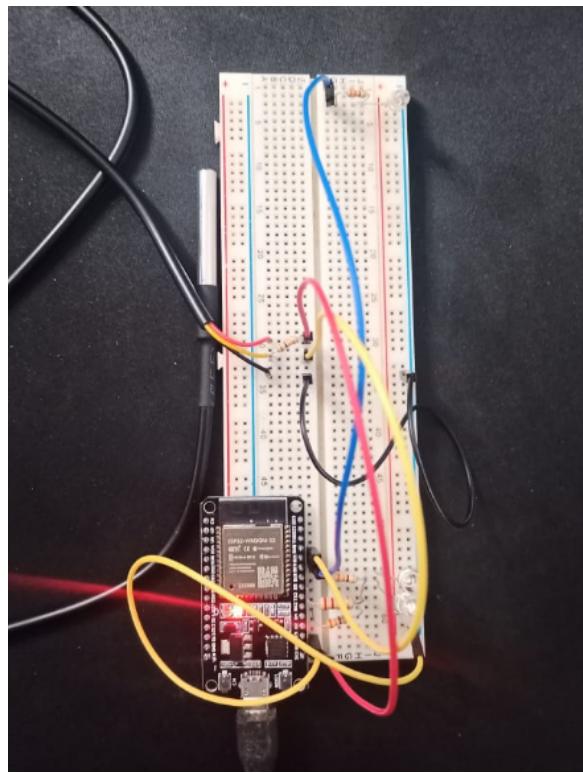


## implementación de la Maqueta:



Implantación de la arquitectura electrónica:





### Despliegue en Cloud (Plataforma Ubidots):

NAME	LAST ACTIVITY	CREATED AT	ACTIONS
esp32_led	20 hours ago	2022-12-21 16:34:05 -05:00	

0.00  
Alarma  
Last activity:  
20 hours ago

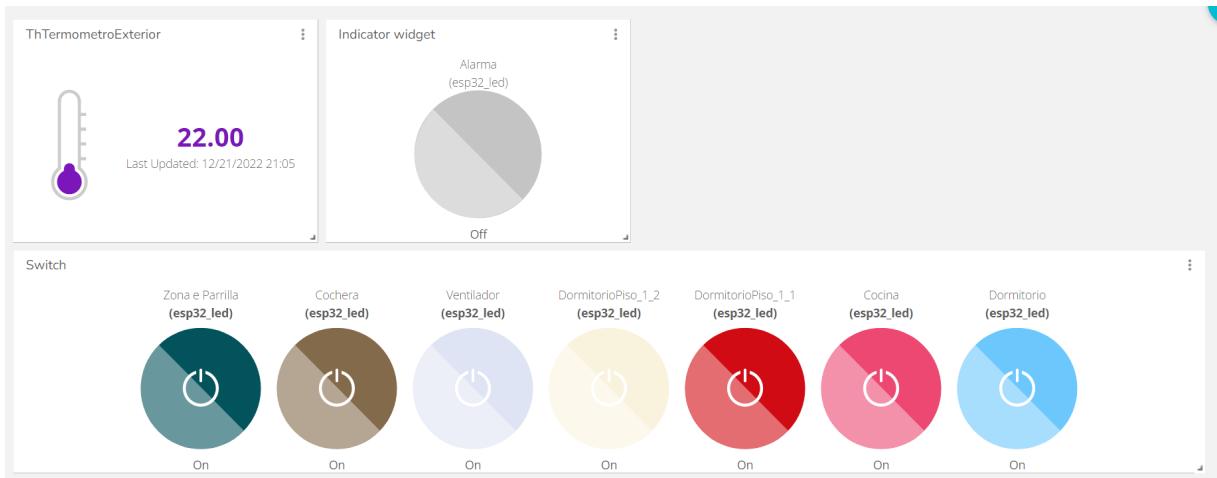
1.00  
Cochera  
Last activity:  
20 hours ago

1.00  
Cocina  
Last activity:  
20 hours ago

0.00  
CocinaT°  
Last activity:  
20 hours ago

1.00  
Dormitorio  
Last activity:  
20 hours ago

1.00  
DormitorioPiso\_1\_1  
Last activity:  
20 hours ago



## Código implementado:

### Código

```
#define BS18B20 27
OneWire oneWire(BS18B20);
DallasTemperature Sensor(&oneWire);
float valorLectura;

-----
#define VARIABLE_LABEL "pot"           // Nombre de la
#define DEVICE_LABEL "esp32_led"       // Nombre del dispositivo
#define SENSOR 36                      // Set the GPIO3
#define LED_1 23
#define LED_2 22
#define LED_3 21
#define LED_4 19
#define LED_5 18
#define LED_6 5
#define LED_7 4

#define BTN 15
int valorBoton = 0;

char mqttBroker[] = "industrial.api.ubidots.com";
char payload[100];
char topic[150];
char topicSubscribe[150];
char str_sensor[10];
char valorAlarma[10];
char valorTemp2[10];
```

Definir variables para trabajar con el sensor de temperatura BS18B20.

BS18B20 (27) es el número de pin asignado al sensor en el ESP32. OneWire es un objeto de la clase OneWire que se utiliza para comunicarse con el sensor a través del protocolo OneWire.

Definen constantes para los pines de los leds y botón conectados al ESP32.

Definen variables de tipo carácter para almacenar el nombre del servidor MQTT de Ubidots, el payload (datos) de un mensaje MQTT, el tópico de un mensaje MQTT y diversas otras cadenas de caracteres necesarias para el proyecto.

```
60
61 // Variable de timer usando millis()
62 unsigned long lastTime = 0;
63 unsigned long timerDelay = 2000;
64
65 ****
66 * Funciones Auxiliares
67 ****
68 WiFiClient ubidots;
69 PubSubClient client(ubidots);
70
```

Se definen dos variables de tipo entero sin signo largo para utilizar como un timer con la función millis() de Arduino. La variable lastTime almacena el tiempo en milisegundos cuando se ejecutó por última vez la función loop(). La variable timerDelay establece el intervalo de tiempo en milisegundos entre ejecuciones de la función loop().

Estas líneas crean dos objetos, ubidots y client, para conectarse a Ubidots a través de MQTT. ubidots es un objeto de la clase WiFiClient que se utiliza para establecer la conexión con Ubidots. client es un objeto de la clase PubSubClient que se utiliza para enviar y recibir mensajes MQTT a través de la conexión WiFiClient.

## Código

La función callback que se ejecuta cada vez que se recibe un mensaje MQTT en el tópico especificado. La función imprime el tópico y el payload del mensaje recibido en el puerto serie. Luego, extrae el valor de la habitación del tópico y el estado del led desde el payload del mensaje. Finalmente, utiliza una estructura switch-case para encender o apagar el led correspondiente según el valor de la habitación y el estado del led recibido.

## Código

La función **setup()** se encarga de inicializar el **ESP32** y conectarse a la red WiFi y al servidor MQTT de **Ubidots**.

Primero, utiliza las constantes **WIFISSID** y **PASSWORD** para conectarse a la red WiFi.

Se inicializa el sensor de temperatura llamando a la función **begin()** de la clase **DallasTemperature**.

Luego, utiliza la función **setServer()** para establecer la conexión con el servidor **MQTT** de Ubidots y la función **setCallback()** para establecer la función callback que se ejecutará cuando se reciba un mensaje MQTT.

Si la conexión falla, se muestra un mensaje de error en el puerto serie ("."). Una vez conectado, se suscribe al tópico específico para recibir mensajes MQTT.

```
72 void callback(char* topic, byte* payload, unsigned int length) {
73     Serial.print("Mensaje llegado ");
74     Serial.print(topic);
75     Serial.print(":");
76     for (int i = 0; i < length; i++){
77         Serial.print((char)payload[i]);
78     }
79     char valorHabitacion = topic[2];
80     Serial.print("valor habitacion: ");
81     Serial.println(valorHabitacion);
82     Serial.print("Estado: ");
83     Serial.println((char)payload[10]);
84     Serial.println((char)payload[10]);
85     switch(valorHabitacion) {
86     case '1':
87         if ((char)payload[10]== '1'){
88             digitalWrite(LED_1, HIGH); // Encendido de led
89         }
90         else if ((char)payload[10]== '0'){
91             digitalWrite(LED_1, LOW); // Apagado de led
92         }
93         break;
94     case '2':
95         if ((char)payload[10]== '1'){
96             digitalWrite(LED_2, HIGH); // Encendido de led
97         }
98         else if ((char)payload[10]== '0'){
99             digitalWrite(LED_2, LOW); // Apagado de led
100        }
101        break;
102    case '3':
103        if ((char)payload[10]== '1'){
104            digitalWrite(LED_3, HIGH); // Encendido de led
105        }
106        else if ((char)payload[10]== '0'){
107            digitalWrite(LED_3, LOW); // Apagado de led
108        }
109        break;
110    case '4':
111        if ((char)payload[10]== '1'){
112            digitalWrite(LED_4, HIGH); // Encendido de led
113        }
114        else if ((char)payload[10]== '0'){
115            digitalWrite(LED_4, LOW); // Apagado de led
116        }
117        break;
118    case '5':
119        if ((char)payload[10]== '1'){
120            digitalWrite(LED_5, HIGH); // Encendido de led
121        }
122        else if ((char)payload[10]== '0'){
123            digitalWrite(LED_5, LOW); // Apagado de led
124        }
125        break;
126    }
```

```
197 void setup() {
198     Sensor.begin();
199
200
201
202     Serial.begin(115200);
203     WiFi.begin(WIFISSID, PASSWORD);
204
205     pinMode(LED_1, OUTPUT);
206     pinMode(LED_2, OUTPUT);
207     pinMode(LED_3, OUTPUT);
208     pinMode(LED_4, OUTPUT);
209     pinMode(LED_5, OUTPUT);
210     pinMode(LED_6, OUTPUT);
211     pinMode(LED_7, OUTPUT);
212     pinMode(BTN, INPUT);
213
214     Serial.println();
215     Serial.print("Conectandose a la red WiFi...");
216
217     while (WiFi.status() != WL_CONNECTED) {
218         Serial.print(".");
219         delay(500);
220     }
221
222     Serial.println("");
223     Serial.println("WiFi Conectado");
224     Serial.println("IP address: ");
225     Serial.println(WiFi.localIP()); // Mostrando la IP asignada
226
227     client.setServer(mqttBroker, 1883); // Conectando a servidor Ubidot
228     client.setCallback(callback); // Llamada a función callback
229
230 }
231 }
```

```

149
150 void reconnect() {
151
152     // Loop para reconexión MQTT
153     while (!client.connected()) {
154         Serial.println("Intentando conectarse...");
155
156         // Conectandose
157         if (client.connect(MQTT_CLIENT_NAME, TOKEN)) {
158             Serial.println("Conectado");
159             sprintf(topicSubscribe, "/v1.0/devices/%s/%s", DEVICE_LABEL, "led_1");
160             client.subscribe(topicSubscribe);
161
162             sprintf(topicSubscribe, "%s", ""); // Cleans the topicSubscribe
163             sprintf(topicSubscribe, "/v1.0/devices/%s/%s", DEVICE_LABEL, "led_2");
164             client.subscribe(topicSubscribe);
165
166             sprintf(topicSubscribe, "%s", ""); // Cleans the topicSubscribe
167             sprintf(topicSubscribe, "/v1.0/devices/%s/%s", DEVICE_LABEL, "led_3");
168             client.subscribe(topicSubscribe);
169
170             sprintf(topicSubscribe, "%s", ""); // Cleans the topicSubscribe
171             sprintf(topicSubscribe, "/v1.0/devices/%s/%s", DEVICE_LABEL, "led_4");
172             client.subscribe(topicSubscribe);
173
174             sprintf(topicSubscribe, "%s", ""); // Cleans the topicSubscribe
175             sprintf(topicSubscribe, "/v1.0/devices/%s/%s", DEVICE_LABEL, "led_5");
176             client.subscribe(topicSubscribe);
177
178             sprintf(topicSubscribe, "%s", ""); // Cleans the topicSubscribe
179             sprintf(topicSubscribe, "/v1.0/devices/%s/%s", DEVICE_LABEL, "led_6");
180             client.subscribe(topicSubscribe);
181
182             sprintf(topicSubscribe, "%s", ""); // Cleans the topicSubscribe
183             sprintf(topicSubscribe, "/v1.0/devices/%s/%s", DEVICE_LABEL, "led_7");
184             client.subscribe(topicSubscribe);
185
186         } else {
187             Serial.print("Falla, reca");
188             Serial.print(client.state());
189             Serial.println(" Espera 2 segundos");
190             delay(2000);
191         }
192     }
193 }
```

## Código

La función `reconnect()` es una función de la librería PubSubClient que se utiliza para reconnectar el cliente MQTT al servidor en caso de que la conexión se haya perdido.

En cada iteración del bucle, se intenta conectarse al servidor MQTT utilizando la función `connect()` de PubSubClient. Si la conexión es exitosa, se muestra un mensaje en el puerto serie y se suscribe al tópico de cada habitación para recibir mensajes MQTT de encendido o apagado de los LED.

Si la conexión falla, se muestra un mensaje en el puerto serie y se espera 2 segundos antes de volver a intentar conectarse al servidor.

```

244 void loop() {
245
246     // Si pierde conexión , vuelve a conectarse
247     if (!client.connected()) {
248         reconnect();
249     }
250     if((millis() - lastTime)> timerDelay){
251         lastTime = millis();
252         sprintf(topic, "%s%s", "/v1.0/devices/", DEVICE_LABEL);
253
254         sprintf(payload, "%s", ""); // Cleans the payload
255         sprintf(payload, "{\"%s\":%s", VARIABLE_LABEL); // Agregar label
256         // Lectura de potenciómetro
257         float sensor = analogRead(SENSOR);
258         Serial.print("sensor:");
259         Serial.println(sensor*1024);
260         dtosrtf(sensor*500/4020, 5, 2, str_sensor);
261
262         sprintf(payload, "%s {\\"value\\": %s}", payload, str_sensor); // Agregar valor
263         client.publish(topic, payload);
264
265         sprintf(payload, "%s", ""); // Cleans the payload
266         sprintf(payload, "{\"%s\":%s", "alarma"); // Agregar label de
267         valorBoton = digitalRead(BUTTON);
268         dtosrtf(valorBoton, 1, 0, valorAlarma);
269         Serial.print("valor del boton:");
270         Serial.println(valorBoton);
271
272         sprintf(payload, "%s {\\"value\\": %s}", payload, valorAlarma); // Agregar valor
273         client.publish(topic, payload);
274
275         sprintf(payload, "%s", ""); // Cleans the payload
276         sprintf(payload, "{\"%s\":%s", "temp2"); // Agregar label de
277         Sensor.requestTemperatures();
278         valorLectura = Sensor.getTempByIndex(0);
279         Serial.println(valorLectura);
280         dtosrtf(valorLectura, 3, 2, valorTemp2);
281
282         sprintf(payload, "%s {\\"value\\": %s}", payload, valorTemp2); // Agregar valor
283         client.publish(topic, payload);
284
285     }
286     client.loop();
287 }
```

## Código

La función `loop()` se ejecuta repetidamente y se encarga de enviar los valores del botón y del sensor de temperatura a Ubidots.

Luego, se utiliza el timer creado con las variables `lastTime` y `timerDelay` para enviar los valores del botón y del sensor de temperatura cada 2 segundos.

Se lee el valor del botón y se almacena en la variable `valorBoton`.

Después, se construye el topic y el payload del mensaje MQTT y se envía el mensaje a Ubidots con la función `publish()` de PubSubClient.

Luego, se lee el valor del sensor de temperatura utilizando la clase `DallasTemperature` y se convierte a una cadena de caracteres con la función `dtostrf()`. Se construye nuevamente el topic y el payload del mensaje MQTT y se envía el mensaje a Ubidots. Finalmente, se actualiza la variable `lastTime` con el tiempo actual para poder utilizar el timer en la próxima ejecución de la función `loop()`.