



IOT – Sistemas Embebidos

Laboratorio 2

Integrantes del grupo:

- Nicolás Raposo
- Martín Da Rosa
- Rafael Durán

Docentes: Nicolás Calarco y Pablo Alonso

29/5/2025

Contenido

Primera Parte.....	4
1. Rehacer la configuración del LED del laboratorio anterior.....	4
a-b-c.	4
2. Lectura de botones mediante polling	4
a. Lea las referencias brindadas por Espressif para botones touch y haga una breve descripción de su funcionamiento.	4
b. Revise el esquemático de la placa base y el esquemático del TouchPad ¿Hay que modificar algo en la placa base para poder utilizar la placa de expansión? ¿Qué problemas existen para esta placa de expansión?	4
c. Diseñe un algoritmo (en pseudocódigo) indicando cómo utilizar el TouchPad y cómo detectar el estado de sus botones (presionado/no presionado).....	5
d. Cree una librería nueva inicializando el TouchPad e implementando la detección de los botones y modifique el código anterior para que al presionar un botón cambie el estado del LED (color, brillo, on/off, etc).	6
Segunda parte	7
1a. Para poder llevar a cabo la implementación tanto del punto de acceso Wifi como la estación WiFi es necesario comprender que es el ESP-NETIF y que es el Event Loop.	7
1b. Comente que hace la función ‘esp_netif_init()’ y si cree que es necesario para la implementación de redes WiFi.....	7
1c. Comente que hace la función ‘esp_event_loop_create_default()’ y qué utilidad podría tener para la implementación de redes WiFi.	8
2a. Comente con qué opciones de configuración cuenta el API de WiFi del equipo basado en la guía de referencia.....	8
2b. Implemente en pseudocódigo, y utilizando las funciones que aparecen en la referencia	9
2c. ¿Qué datos del equipo que se conecta muestra el equipo? ¿Qué datos de red relevantes puede obtener en el monitor del idf?	9
2d. Pruebe distintas configuraciones (WiFi abierto/con contraseña, SSID oculto, etc.) y comente su resultado en la implementación.....	10
3a. Implemente, en pseudocódigo una STA con el ESP32-S2 y conectarlo a alguna red WiFi que tenga disponible.	11
3b. ¿Qué datos de red relevantes puede obtener en el monitor del idf? ¿Qué diferencias hay comparado con la implementación del AP?	11

3c. Pruebe distintas configuraciones y comente su efecto en la implementación.	12
Tercera parte	13
3.a Implemente en pseudocódigo un servidor web y que al acceder despliegue el mensaje de “Hola mundo” en el navegador.	13
3.b Usando las librerías ya implementadas genere una red WiFi (que el equipo funcione como AP) o conecte el mismo a una red ya presente (que el equipo funcione como STA).....	14
3.c Monte un servidor web basándose en los ejemplos brindados previamente. Se debe poder conectar/acceder a este desde una PC como de un dispositivo móvil (por ej. Celular).	15
3.d ¿Cómo haría para enviar información al equipo? Implemente dicha solución.	16
3.e ¿Qué datos de un sistema de IoT cree relevantes para persistir en el equipo?	17

Primera Parte

1. Rehacer la configuración del LED del laboratorio anterior

a-b-c.

En el tag “Laboratorio_2a” del repositorio se encuentra esta esta parte del laboratorio. Las partes del Laboratorio 1 efectivamente funcionan bien y además a las librerías creadas durante el mismo se añadieron nuevas funciones como por ejemplo para cambiar el brillo del led.

2. Lectura de botones mediante polling

a. Lea las referencias brindadas por Espressif para botones touch y haga una breve descripción de su funcionamiento.

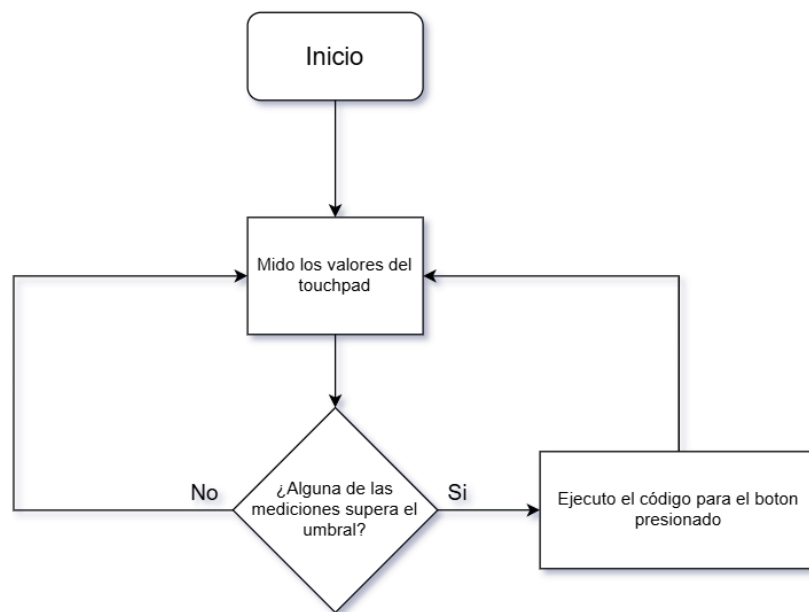
Según la referencia de Espressif, el touchpad funciona detectando un cambio en la capacitancia del sensor con el ambiente. Utiliza una FSM implementada con hardware que puede ser activada con la función: `touch_pad_set_fsm_mode(TOUCH_FSM_MODE_xx)`, cambiando “xx” por SW o TIMER se elige entre la activación de la FSM por software (cuando se le indique) o mediante un timer (de forma periódica y automática). Luego, utilizando la función: `touch_pad_read_raw_data(touch_pad_t touch_num, uint16_t * touch_value)` se devuelve el valor en crudo del sensor y en que botón sucedió.

b. Revise el esquemático de la placa base y el esquemático del TouchPad ¿Hay que modificar algo en la placa base para poder utilizar la placa de expansión? ¿Qué problemas existen para esta placa de expansión?

No fue necesario modificar nada en la placa para usar el touchpad ni tampoco se encontró algo en el alguno de los esquemáticos que indique lo contrario. Sin embargo, se encontró que si se usa el touchpad junto a la placa de audio y si se incluye la lectura del botón PHOTO del touchpad, se bloquea todo el funcionamiento de la placa siendo necesario sacar la placa de audio o no incluir el botón mencionado en las mediciones.

c. Diseñe un algoritmo (en pseudocódigo) indicando cómo utilizar el TouchPad y cómo detectar el estado de sus botones (presionado/no presionado).

Como se observa en la figura a continuación, el algoritmo para detectar los botones del touchpad es el siguiente:



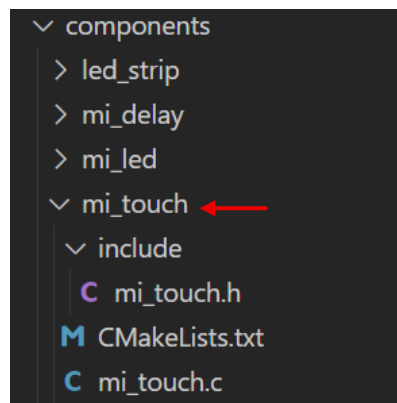
La idea es que el algoritmo esté todo el tiempo midiendo el touchpad, si alguno de los botones supera el umbral determinado entonces ejecuta lo indicado. Algunos problemas que tiene este algoritmo son, por ejemplo, si se presiona un botón mientras estoy ejecutando el código para el botón anterior, no se detectará el nuevo. Por otro lado, la medición no se puede hacer en todos los botones al mismo tiempo, sino que se hace en un orden indicado, en este caso se eligió el orden siguiente:

- 1) VOL_UP
- 2) PLAY/PAUSE
- 3) VOL_DOWN
- 4) RECORD
- 5) PHOTO
- 6) NETWORK

Entonces, si el sensor está comenzando la medición y se presionan VOL_UP y NETWORK exactamente al mismo tiempo, el que se detectará primero es VOL_UP.

d. Cree una librería nueva inicializando el TouchPad e implementando la detección de los botones y modifique el código anterior para que al presionar un botón cambie el estado del LED (color, brillo, on/off, etc).

Como se puede ver en la figura de abajo (y en el repositorio), se creó la librería `mi_touch` que tiene funciones para configurar el touchpad, para recorrer todos los botones y ver si alguno fue presionado y finalmente la función que implementa el algoritmo de la parte anterior.



Segunda parte

1a. Para poder llevar a cabo la implementación tanto del punto de acceso Wifi como la estación WiFi es necesario comprender que es el ESP-NETIF y que es el Event Loop.

ESP-NETIF es la capa de abstracción de red del ESP-IDF que proporciona una interfaz unificada para manejar diferentes tipos de conexiones de red (WiFi, Ethernet, etc.). Esta capa actúa como intermediario entre las aplicaciones de usuario y los drivers de red de bajo nivel, simplificando la configuración y gestión de interfaces de red. ESP-NETIF maneja automáticamente la asignación de direcciones IP, configuración de DNS, y la gestión del stack TCP/IP.

Event Loop es un sistema de manejo de eventos basado en el patrón observer que permite a diferentes componentes del sistema comunicarse de manera asíncrona. En el contexto de redes WiFi, el Event Loop es fundamental ya que las operaciones de red (conexión, desconexión, asignación de IP, etc.) son eventos asíncronos que no pueden predecirse en tiempo de ejecución. Este sistema permite que la aplicación reaccione a eventos como:

- Conexión exitosa a una red WiFi.
- Pérdida de conexión.
- Asignación de dirección IP.
- Conexión de nuevos dispositivos del AP.

1b. Comente que hace la función 'esp_netif_init()' y si cree que es necesario para la implementación de redes WiFi.

Si, es completamente necesario, dado que inicializa el stack TCP-IP. Debe llamarse una única vez, sin ello no se pueden crear las interfaces para contactarse a una red ni para levantar un AP.

1c. Comente que hace la función 'esp_event_loop_create_default()' y qué utilidad podría tener para la implementación de redes WiFi.

Es la función que va a crear el sistema de eventos y suscripciones que suceden en la red, por ejemplo cuando un nuevo dispositivo se conecta, se genera un evento relacionado. Igual cuando se desconecta. Todo eso ocurre de manera asincrónica y es necesario un event loop para crear un mecanismo donde poder ir procesando esos eventos a medida que suceden.

2a. Comente con qué opciones de configuración cuenta el API de WiFi del equipo basado en la guía de referencia.

Primero debemos inicializar la configuración mediante WIFI_INIT_CONFIG_DEFAULT, para asegurarnos que todos los parámetros se inicializarán correctamente, a continuación podremos sobrescribir los [parametros](#) que necesitemos modificar. Los que utilizamos para el laboratorio son:

- ssid_len: Largo del SSID
- max_connection: Máximo numero de estaciones que se permiten simultáneamente
- authmode: Método de autenticación elegido.

2b. Implemente en pseudocódigo, y utilizando las funciones que aparecen en la referencia

```
void init_wifi_ap(char ssid, char password)
{
    // 1. Inicializacion de componentes y event loop
    esp_netif_init();
    esp_event_loop_create_default();

    // 2. Configuración del AP
    esp_wifi_init(WIFI_INIT_CONFIG_DEFAULT());
    wifi_config_t = {
        .ap = {
            .ssid_len = strlen(ssid),
            .max_connection = 4,
            .authmode = WIFI_AUTH_WPA2_PSK,
            .ssid = ssid
            .password = password
        },
    };
    esp_wifi_set_mode(WIFI_MODE_AP);
    esp_wifi_set_config(WIFI_IF_AP, wifi_config);

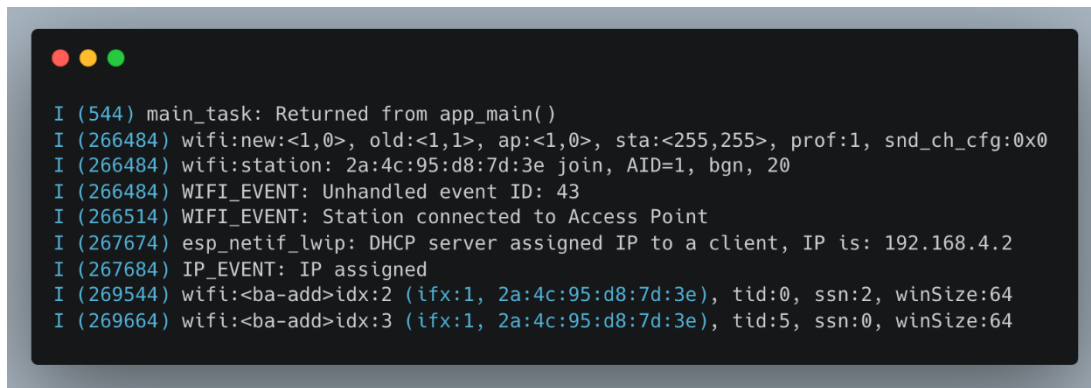
    // 3. Suscripciones a event loop
    esp_event_handler_register(WIFI_EVENT, ESP_EVENT_ANY_ID, wifi_event_cb);
    esp_event_handler_register(IP_EVENT, IP_EVENT_AP_STAIPASSIGNED,
    ip_event_cb);
    // 4. Lanzamiento del AP
    esp_wifi_start();
}
```

La secuencia consta de inicializar el stack TCP-IP y el event loop. Luego configurar la interfaz wifi para levantar el AP. Registrar los callbacks para escuchar los eventos y finalmente darle inicio al AP.

2c. ¿Qué datos del equipo que se conecta muestra el equipo? ¿Qué datos de red relevantes puede obtener en el monitor del idf?

```
I (454) wifi_init: tcp rx win: 5760
I (454) wifi_init: tcp mss: 1440
I (454) wifi_init: WiFi IRAM OP enabled
I (464) wifi_init: WiFi RX IRAM OP enabled
I (474) phy_init: phy_version 2620,f8f9319, Feb 6 2025, 14:35:09
I (524) wifi: mode : softAP (7c:df:a1:00:90:7d)
I (524) wifi: Total power save buffer number: 16
I (524) wifi: Init max length of beacon: 752/752
I (524) wifi: Init max length of beacon: 752/752
I (534) esp_netif_lwip: DHCP server started on interface WIFI_AP_DEF with IP: 192.168.4.1
I (544) WIFI_EVENT: Access Point started
I (544) main_task: Returned from app_main()
```

Al momento de inicializar el AP, se pueden ver parámetros de inicialización del AP, así también como la inicialización del DHCP server en la IP asignada. La línea WIFI_EVENT es un log que pusimos en la función de suscripción al event loop. Aquí se ve como al momento de levantar el AP un evento específico es emitido. En este caso WIFI_EVENT_AP_START que es mapeado a “Access Point started”.

A screenshot of a terminal window with a dark background and light-colored text. The window has three colored window control buttons (red, yellow, green) in the top-left corner. The log messages show the sequence of events for a WiFi connection, including station join, event handling, connection to an access point, and DHCP IP assignment.

```
I (544) main_task: Returned from app_main()
I (266484) wifi:new:<1,0>, old:<1,1>, ap:<1,0>, sta:<255,255>, prof:1, snd_ch_cfg:0x0
I (266484) wifi:station: 2a:4c:95:d8:7d:3e join, AID=1, bgn, 20
I (266484) WIFI_EVENT: Unhandled event ID: 43
I (266514) WIFI_EVENT: Station connected to Access Point
I (267674) esp_netif_lwip: DHCP server assigned IP to a client, IP is: 192.168.4.2
I (267684) IP_EVENT: IP assigned
I (269544) wifi:<ba-add>idx:2 (ifx:1, 2a:4c:95:d8:7d:3e), tid:0, ssn:2, winSize:64
I (269664) wifi:<ba-add>idx:3 (ifx:1, 2a:4c:95:d8:7d:3e), tid:5, ssn:0, winSize:64
```

Al conectarnos al AP vemos como dos nuevos eventos son emitidos, uno el cual es ignorado y el otro es el que indica que una nueva estación se ha conectado. También se pueden ver la MAC address de la estación y la IP que le asigno el servidor DHCP.

(El evento ignorado es el 43, que significa WIFI_EVENT_HOME_CHANNEL_CHANGE, el cual indica que el AP ha cambiado el canal por el cual se disponibiliza).

2d. Pruebe distintas configuraciones (WiFi abierto/con contraseña, SSID oculto, etc.) y comente su resultado en la implementación.

Para tener el wifi abierto se probó con:

```
wifi_config.ap.authmode = WIFI_AUTH_OPEN
```

Viendo efectivamente como ya no es necesario ingresar una contraseña para conectarse.

Para ocultar el SSID se probó con:

```
wifi_config.ap.ssid_hidden = 1
```

Viendo como ya no se hacia el broadcast del SSID.

También se probó con:

```
wifi_config.ap.channel = 1 - 11
```

Igualmente un Iphone es capaz de conectarse en cualquiera de esos canales.

3a. Implemente, en pseudocódigo una STA con el ESP32-S2 y conectarlo a alguna red WiFi que tenga disponible.

```
void connect_wifi_ap(char ssid, char password)
{
    // 1. Inicializacion de componentes y event loop
    esp_netif_init();
    esp_event_loop_create_default();

    // 2. Configuración de la estación
    esp_wifi_init(WIFI_INIT_CONFIG_DEFAULT());
    wifi_config_t wifi_config = {
        .sta = {
            .authmode = WIFI_AUTH_WPA2_PSK,
            .ssid = ssid
            .password = password
        },
    };
    esp_wifi_set_mode(WIFI_MODE_STA);
    esp_wifi_set_config(WIFI_IF_STA, wifi_config);

    // 3. Suscripciones a event loop
    esp_event_handler_register(WIFI_EVENT, ESP_EVENT_ANY_ID,
    wifi_event_cb);
    // 4. Lanzamiento de la estación y la conexión
    esp_wifi_start();
    esp_wifi_connect();
}
```

La secuencia es muy parecida a la creación del AP, solo que esta vez toca configurar la interfaz wifi para conectarse a una red. Luego registramos los callbacks para escuchar los eventos y finalmente realizar la conexión.

3b. ¿Qué datos de red relevantes puede obtener en el monitor del idf? ¿Qué diferencias hay comparado con la implementación del AP?

```
I (525) WIFI_EVENT: Station started
I (525) WIFI: Connecting to AP: Antelckcfe-2.4GHz
I (525) main_task: Returned from app_main()
I (535) wifi:new:<1,0>, old:<1,0>, ap:<255,255>, sta:<1,0>, prof:1, snd_ch_cfg:0x0
I (535) wifi:state: init -> auth (0xb0)
I (545) wifi:state: auth -> assoc (0x0)
I (565) wifi:state: assoc -> run (0x10)
I (585) wifi:connected with Antelckcfe-2.4GHz, aid = 10, channel 1, BW20, bssid = 20:e8:82:d5:5c:22
I (585) wifi:security: WPA2-PSK, phy: bgn, rssi: -50
I (585) wifi:pm start, type: 1

I (585) wifi:dp: 1, bi: 102400, li: 3, scale listen interval from 307200 us to 307200 us
I (595) WIFI_EVENT: Station connected to AP
I (625) wifi:AP's beacon interval = 102400 us, DTIM period = 1
I (3205) wifi:<ba-add>idx:0 (ifx:0, 20:e8:82:d5:5c:22), tid:0, ssn:1, winSize:64
I (4205) esp_netif_handlers: sta ip: 192.168.1.18, mask: 255.255.255.0, gw: 192.168.1.1
```

En este caso los eventos que nos llegan al event loop son:

Station started - WIFI_EVENT_STA_START

Station connected to AP - WIFI_EVENT_STA_CONNECTED

Luego el monitor del IDF nos informa acerca del handshake cuando la conexión con el AP se está ejecutando. También se observan parámetros del AP y la IP asignada a la placa.

3c. Pruebe distintas configuraciones y comente su efecto en la implementación.

Al ingresar una contraseña incorrecta, al event loop llega un evento de: WIFI_EVENT_STA_DISCONNECTED. Lo cual es curioso dado que no puede existir una desconexión si no hubo una conexión previamente. El mismo evento es emitido si el SSID es incorrecto.

Si se conecta correctamente, y luego se apaga el router, el mismo evento WIFI_EVENT_STA_DISCONNECTED es generado. Pero también se genera un evento WIFI_EVENT_STA_BEACON_TIMEOUT, lo cual puede servir de información de que el AP ya no está al alcance de la estación. Al prender el router, la estación no se reconecta (parece que hay que implementar este mecanismo si se quiere tener reconexión).

Tercera parte

3.a Implemente en pseudocódigo un servidor web y que al acceder despliegue el mensaje de “Hola mundo” en el navegador.

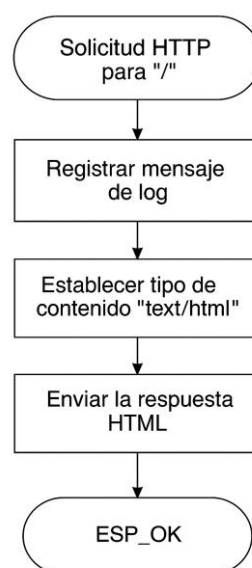
```
// Manejador para la ruta "/"
static esp_err_t hello_handler(httpd_req_t *req)
{
    ESP_LOGI(TAG, "Solicitud recibida para URI: %s", req->uri);
    // Establecer el tipo de contenido
    httpd_resp_set_type(req, "text/html");

    // Enviar la respuesta HTML
    extern const char resp[] asm("_binary_index_html_start");
    httpd_resp_send(req, resp, strlen(resp));

    return ESP_OK;
}
```


La función “*hello_handler*” actúa como manejador para las solicitudes HTTP dirigidas a la ruta raíz (“/”) en un servidor web embebido ejecutado sobre un microcontrolador ESP32. Su principal objetivo es procesar las solicitudes entrantes y devolver una respuesta HTML predefinida.

A continuación se muestra un diagrama de flujo mostrando el comportamiento de esta función:



3.b Usando las librerías ya implementadas genere una red WiFi (que el equipo funcione como AP) o conecte el mismo a una red ya presente (que el equipo funcione como STA).

Para la implementación de esta parte se generó una nueva función llamada *init_web_services()* la cual reutiliza los componentes creados previamente en el ejercicio anterior:



```
// Opción 1: Crear Access Point
ESP_LOGI(TAG, "Configurando WiFi en modo Access Point");
init_wifi_ap("ESP32-WebServer", "12345678");

// Opción 2: Conectar a red existente
ESP_LOGI(TAG, "Conectando a red WiFi existente");
connect_wifi_ap("NombreDeRed", "ContraseñaDeRed");
```

Modo Access Point (AP):


El ESP32 se configura para crear su propia red WiFi, a la cual otros dispositivos pueden conectarse directamente. En el código, esto se logra con la función *init_wifi_ap()*, que genera una red con el nombre ESP32-WebServer y la contraseña 12345678. Esta modalidad permite acceder al servidor web del ESP32 sin necesidad de una red externa.

Modo Station (STA):

Alternativamente, el ESP32 puede conectarse a una red WiFi ya existente. Esto se hace mediante la función *connect_wifi_ap()*, la cual debe ser adaptada con los datos reales de la red a la que uno se desee conectar. En este modo, el ESP32 actúa como un cliente dentro de esa red.

3.c Monte un servidor web basándose en los ejemplos brindados previamente. Se debe poder conectar/acceder a este desde una PC como de un dispositivo móvil (por ej. Celular).

La función que se encarga de montar el servidor web se llama *httpd_handle_t init_web_server()*. A continuación se muestran las partes más relevantes de dicha función:

A screenshot of a code editor with a dark background and light-colored text. The code is in C and defines the `init_web_server` function. It includes comments in Spanish. The function sets up an HTTP server on port 80 and registers handlers for `/` and `/led`.

```
httpd_handle_t init_web_server(led_strip_t *led_strip)
{
    g_led_strip = led_strip;
    httpd_handle_t server = NULL;
    httpd_config_t config = HTTPD_DEFAULT_CONFIG();

    config.server_port = 80;
    ...

    if (httpd_start(&server, &config) == ESP_OK) {
        httpd_register_uri_handler(server, &hello_uri);
        httpd_register_uri_handler(server, &led_uri);
        ...
        return server;
    }
    ...
    return NULL;
}
```

En esta sección se configura e inicia el servidor HTTP en el puerto **80**, el estándar para servidores web. También se registran los manejadores de las rutas `/` y `/led`.

Las rutas que maneja el servidor son las siguientes:

```
static const httpd_uri_t hello_uri = {
    .uri      = "/",
    .method   = HTTP_GET,
    .handler  = hello_handler,
    .user_ctx = NULL
};

static const httpd_uri_t led_uri = {
    .uri      = "/led",
    .method   = HTTP_GET,
    .handler  = led_handler,
    .user_ctx = NULL
};
```

Estas estructuras definen las rutas que atiende el servidor:

- /: carga la página principal (HTML embebido en el firmware).
- /led: permite controlar el color del LED mediante parámetros en la URL.

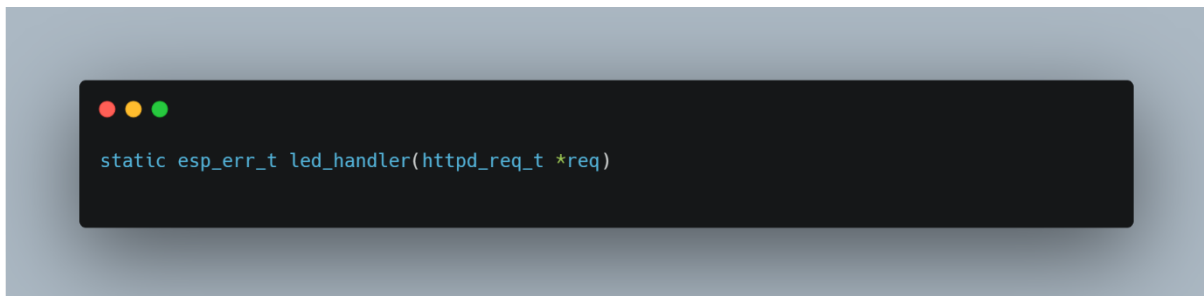
3.d ¿Cómo haría para enviar información al equipo? Implemente dicha solución.

Para el envío de información a la computadora se implementó un script (programado en Javascript), el cuál se encarga de enviar información al ESP32 mediante una solicitud HTTP GET a **/led?color=valor**, donde valor puede ser "red", "green", etc. Dicha solución se encuentra implementada dentro del archivo index.html.

A continuación se muestra dicho fragmento del código:

```
<script>
    function controlledLed(color) {
        fetch('/led?color=' + color)
        .then(response => response.text())
        .then(data => {
            console.log('LED cambiado a: ' + color);
            document.getElementById('status').innerHTML = 'LED: ' + color.toUpperCase();
        })
        .catch(error => {
            console.error('Error:', error);
            document.getElementById('status').innerHTML = 'Error al cambiar LED';
        });
    }
</script>
```


Por otro lado, en el archivo *mi_web_server.c* se encuentra el manejador del endpoint **/led**, que se encarga de *leer el parámetro color de la query string, interpretar su valor, cambiar el color del LED en el dispositivo y devolver una respuesta al cliente*.



La información que se envía es el valor del color: "red", "green", "blue", "yellow", o "off". Este dato lo ingresa el usuario **mediante un botón en la página web**. El valor viaja por HTTP al servidor, que **actúa en consecuencia** (cambia el color del LED).

3.e ¿Qué datos de un sistema de IoT cree relevantes para persistir en el equipo?

En un sistema de Internet de las Cosas (IoT), la persistencia de ciertos datos en el dispositivo es fundamental para garantizar su operatividad autónoma, continuidad frente a reinicios inesperados o fallos de energía, y para permitir una experiencia de usuario coherente. Algunos de los datos más relevantes que deberían guardarse de forma local en el equipo son los siguientes:

1. Configuración de red

Guardar información como el SSID y la contraseña de la red Wi-Fi permite que el dispositivo se reconecte automáticamente tras un reinicio o pérdida de conexión. También puede incluirse información adicional como el modo de operación de la red (Access Point, Station o ambos) y parámetros de red estática en caso de ser necesario.

2. Estado actual del sistema

Es recomendable almacenar el estado de los actuadores y sensores más relevantes, tales como el estado de un LED (encendido/apagado, intensidad, color), o el valor más reciente de un sensor crítico. Esto permite restaurar el último estado funcional del dispositivo ante una interrupción.

3. Credenciales de autenticación

Los dispositivos IoT que se comunican con servicios en la nube requieren almacenar tokens de autenticación, claves API o credenciales cifradas. Guardar esta información de manera segura garantiza una conexión confiable y continua con los servidores remotos.

4. Parámetros de configuración del usuario

Los valores configurables por el usuario, como umbrales de temperatura o humedad, intervalos de muestreo, horarios programados o modos operativos, deben persistirse para asegurar que las preferencias del usuario se mantengan entre sesiones.