Módulo 01 – Sistema de Iluminación Programable con Anillo LED WS2812B
Implementación técnica y análisis energético en la Mesa STEAM

1. ¿Qué vamos a construir?

Un sistema de iluminación RGB direccionable controlado por un ESP32, capaz de encender los LEDs uno a uno generando un efecto secuencial.

Este módulo permite:

- Comprender cómo un microcontrolador controla dispositivos digitales.
- Entender cómo se representa el color en programación.
- Analizar consumo energético básico.
- Preparar la base para sistemas más complejos.

2. Materiales necesarios

Cantidad	Componente
1	ESP32 DevKit
1	Anillo LED WS2812B (16 LEDs)
1	Fuente DC primaria (7–12V, mínimo 1A) Ejemplo: Batería 9V o adaptador DC externo
1	Regulador de voltaje: Módulo LM2596
1	Protoboard
3	Jumpers M-M y M-H (según disposición de pines)
1	Cable USB

Opcional (recomendado para uso avanzado):

- Resistencia 330Ω (línea de datos)
- Capacitor 1000µF (entre 5V y GND)

3. Identificación de pines del anillo

El anillo LED tiene tres conexiones principales:

- 5V → Alimentación positiva
- GND → Tierra
- DIN → Entrada de datos

Verificar la flecha impresa en el anillo.
Debe indicar dirección hacia donde fluye la señal.

3.1 Arquitectura electrónica del LED WS2812B

Cada LED WS2812B integra:

- Un controlador digital interno.
- Un registro de desplazamiento de 24 bits.

- Tres diodos LED (Rojo, Verde y Azul).

El protocolo de comunicación opera a aproximadamente:

- 800 kHz
- Señal digital de un solo hilo
- Codificación por ancho de pulso (PWM temporizado)

Cada LED recibe 24 bits organizados en orden:

Green (8 bits)
Red (8 bits)
Blue (8 bits)

Para un anillo de 16 LEDs:

$$16 \times 24 = 384 \text{ bits por actualización}$$

Esto convierte al sistema en un ejemplo de:

- Comunicación digital serial.
- Sistemas direccionables en cascada.
- Arquitectura distribuida.

4. Conexión eléctrica paso a paso (con batería de 9V y regulador LM2596)
No energizar el circuito mientras se realizan conexiones. Ajustar el LM2596 a **5.0V** con multímetro antes de conectar el ESP32 y el anillo.

Paso 1 – Conectar la batería al LM2596

- Batería 9V **(+):** al pin **IN+** del LM2596
- Batería 9V **(–):** al pin **IN–** del LM2596

Paso 2 – Ajustar la salida del LM2596 a 5V

- Con multímetro, medir entre **OUT+** y **OUT–**
- Girar el tornillo del potenciómetro hasta obtener **5.0V** (aprox.)

Paso 3 – Llevar 5V y GND a la protoboard

- **OUT+ (LM2596)** → riel **+5V** de la protoboard
- **OUT– (LM2596)** → riel **GND** de la protoboard

Paso 4 – Alimentar el ESP32 desde el regulador

- Riel **+5V** → pin **VIN** del ESP32
- Riel **GND** → pin **GND** del ESP32

Paso 5 – Alimentar el anillo LED

- Riel **+5V** → pin **5V** del anillo WS2812B
- Riel **GND** → pin **GND** del anillo WS2812B

Paso 6 – Conectar la señal de datos (con resistencia en serie)

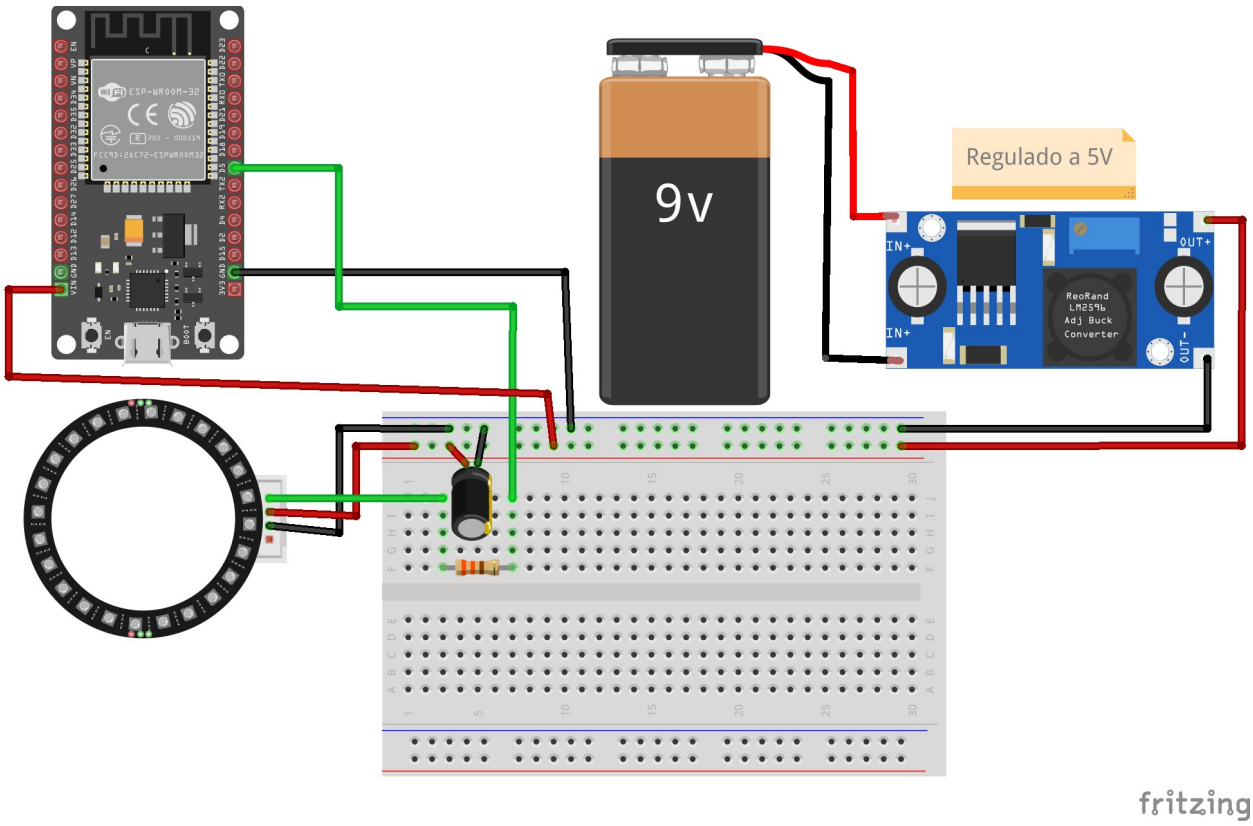
- Pin **GPIO 5** del ESP32 → **resistencia 330Ω** → pin **DIN** del anillo

Paso 7 – Estabilización de la alimentación (recomendado)

- Conectar un **capacitor de 1000µF** entre riel **+5V** y riel **GND** (cerca del anillo). Respetar polaridad:
    - Terminal **(+)** al **+5V**
    - Terminal **(–)** a **GND**

Nota: Es obligatorio que **ESP32, anillo y regulador compartan GND** (tierra común).

5. Plano del circuito



Nota técnica:

Es obligatorio compartir tierra (GND) entre el microcontrolador, el anillo LED y el módulo regulador.
La ausencia de referencia común puede provocar funcionamiento errático o ausencia total de señal.

El pin **VIN del ESP32** recibe **5V regulados** provenientes del módulo LM2596.
No debe conectarse directamente a 9V.

Antes de energizar el sistema, se debe verificar con multímetro que la salida del regulador esté ajustada a **5.0V**.

La batería de 9V puede sustituirse por una fuente de alimentación regulada externa, siempre que cumpla con las siguientes especificaciones:

- **Voltaje de salida estable: 5V DC**

- **Capacidad de corriente mínima recomendada: 1A**

Para aplicaciones con todos los LEDs encendidos en blanco a máxima intensidad, el sistema puede requerir hasta: ≈ 1A (solo anillo LED)

Por seguridad y estabilidad, se recomienda una fuente de **5V – 2A** cuando se prevea uso intensivo o expansión del sistema.

Tabla resumen de conexiones (con batería de 9V y LM2596)

Origen	Destino
Batería 9V (+)	LM2596 IN+
Batería 9V (–)	LM2596 IN–
LM2596 OUT+ (5V)	Riel +5V protoboard
LM2596 OUT– (GND)	Riel GND protoboard
Riel +5V regulado	ESP32 VIN
Riel GND	ESP32 GND
Riel +5V	Anillo 5V
Riel GND	Anillo GND
ESP32 GPIO 5	Resistencia 330Ω → DIN anillo
Capacitor 1000µF (+)	Riel +5V (cerca del anillo)
Capacitor 1000µF (–)	Riel GND

6. Configuración del entorno

1. Instalar Arduino IDE.
2. Instalar soporte ESP32.
3. Seleccionar placa correcta.
4. Instalar librería **Adafruit NeoPixel**.

7. Código completo listo para copiar y pegar

Copiar todo este código y pegarlo en Arduino IDE.

```
#include <Adafruit_NeoPixel.h>
// Pin donde está conectado DIN
#define PIN 5
// Número de LEDs del anillo
#define NUMPIXELS 16
// Crear objeto para controlar el anillo
Adafruit_NeoPixel pixels(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);
```

```
void setup() {
  // Inicializa el anillo
  pixels.begin();
  // Ajusta brillo (0-255)
  pixels.setBrightness(100);
}
void loop() {
  // Encender LEDs uno a uno en azul
  for(int i = 0; i < NUMPIXELS; i++) {
    pixels.clear(); // Apaga todos antes de encender uno
    // Color azul (R=0, G=0, B=255)
    pixels.setPixelColor(i, pixels.Color(0, 0, 255));
    pixels.show(); // Envía datos al anillo
    delay(100);
  }
  // Apagar todos al finalizar
  pixels.clear();
  pixels.show();
  delay(500);
}
```

8. ¿Qué hace exactamente este código?

1. Inicializa el anillo.
2. Define que tiene 16 LEDs.
3. Recorre cada LED.
4. Enciende uno en azul.
5. Espera 100 ms.
6. Continúa con el siguiente.
7. Apaga todos.
8. Repite.

9. Cálculo básico de consumo energético

Consumo máximo por LED:
≈ 60 mA (blanco total)

Si todos los LEDs estuvieran en blanco:

16 × 60mA = 960mA

Potencia aproximada:

P = V × I
P = 5 × 0.96 = 4.8W

En este programa solo hay un LED azul activo:

≈ 20mA
Consumo muy bajo.

9.1 Análisis de corriente en condiciones reales

En aplicaciones prácticas:

- No se utiliza brillo máximo continuo.
- Se recomienda limitar intensidad con setBrightness().

Si se configura:

```
pixels.setBrightness(80);
```

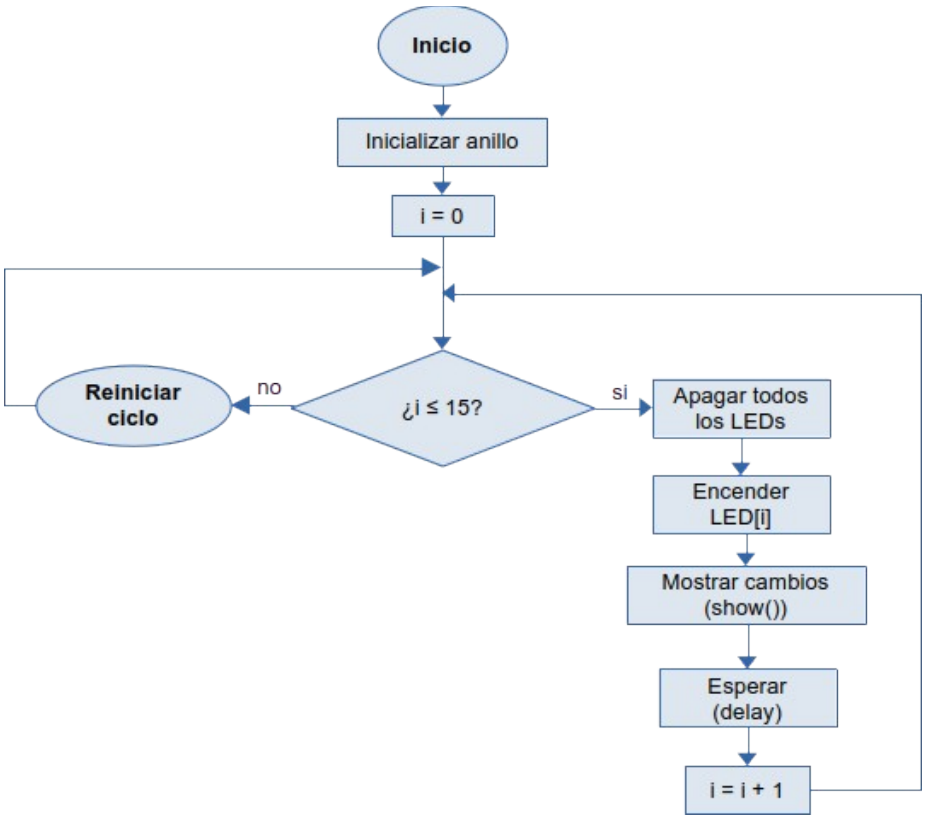
Esto representa aproximadamente: 80 / 255 ≈ 0.31

La corriente máxima teórica (960 mA) se reduce a:
0.31 × 960 mA ≈ 297 mA

Esto reduce:

- Riesgo de caída de tensión.
- Calentamiento.
- Sobrecarga del regulador del ESP32.

10. Diagrama de flujo del programa



10.1 – Modelo algorítmico en pseudocódigo formal

```
INICIO
 Inicializar anillo
 Definir brillo
 MIENTRAS verdadero HACER
    PARA i desde 0 hasta 15 HACER
        Apagar todos los LEDs
        Encender LED[i] en azul
        Mostrar cambios
        Esperar 100 ms
    FIN PARA
    Apagar todos
    Esperar 500 ms
 FIN MIENTRAS
FIN
```

11. Problemas comunes

No enciende:

- Revisar GND común.
- Verificar pin DIN correcto.

Colores incorrectos:

- Revisar orden GRB.

Parpadea:

- Fuente inestable.
- Cables largos.

12. Pruebas y modificaciones

Cambiar azul por rojo:

```
pixels.setPixelColor(i, pixels.Color(255, 0, 0));
```

Encender todos al tiempo:

```
for(int i=0; i<NUMPIXELS; i++){
  pixels.setPixelColor(i, pixels.Color(0,255,0));
}
pixels.show();
```

13. Retos avanzados

- Eliminar delay() usando millis().
- Crear efecto arcoíris.
- Control Web desde el celular (ESP32)
- Integrarlo con sensor LDR.

14. Integración con Inteligencia Artificial para Optimización y Variación del Módulo

El uso de herramientas de inteligencia artificial puede facilitar:

- Generación de nuevos efectos de iluminación.
- Optimización del consumo energético.
- Conversión del código a otros lenguajes.
- Eliminación de errores.
- Mejora de estructura del programa.

Esta sección no reemplaza la comprensión técnica, sino que la complementa.

14.1 ¿Cuándo usar IA en este módulo?

Se recomienda utilizar IA cuando:

- Se desea modificar el comportamiento del anillo.
- Se requiere optimizar el código.
- Se quiere migrar a otro microcontrolador.
- Se busca eliminar bloqueos por uso de delay().
- Se quiere implementar efectos complejos.

14.2 Prompts sugeridos para generar variaciones

A continuación, ejemplos de instrucciones (prompts) que pueden utilizarse:

Generar efecto arcoíris

> “Genera un efecto arcoíris dinámico para un anillo WS2812B de 16 LEDs usando ESP32 y la librería Adafruit NeoPixel.”

Eliminar uso de delay()

> “Reescribe este código eliminando delay() y usando millis() para que el programa no sea bloqueante.”

Reducir consumo energético

> “Optimiza este código para reducir el consumo energético del anillo LED manteniendo efecto visual.”

Migrar a MicroPython

> “Convierte este código de Arduino C++ para ESP32 a MicroPython usando la librería neopixel.”

Adaptar a Arduino UNO

> “Modifica este código para que funcione en un Arduino UNO manteniendo el mismo efecto.”

14.3 Buenas prácticas al usar IA

1. Siempre comprender el código generado.
2. Verificar compatibilidad con la placa seleccionada.
3. Probar el código en simulación o con bajo brillo primero.
4. Validar que no se exceda consumo de corriente.
5. Revisar que el orden GRB sea correcto.

14.4 Ejemplo práctico de mejora con IA

Código original (bloqueante):

```
delay(100);
```

Código sugerido por IA (no bloqueante):

```
unsigned long currentMillis = millis();
if (currentMillis - previousMillis >= interval) {
  previousMillis = currentMillis;
  // acción
}
```

Resultado:

- Sistema más eficiente.
- Permite integrar sensores simultáneamente.
- Preparado para IoT.

14.5 IA como herramienta pedagógica

El uso de IA permite:

- Comparar soluciones.
- Analizar eficiencia.
- Detectar errores de lógica.
- Explorar alternativas de diseño.

Pero el criterio técnico debe ser del estudiante.

14.6 Validación técnica del código generado por IA

Todo código generado mediante IA debe:

1. Verificarse contra hoja de datos del componente.
2. Comprobar consumo máximo estimado.
3. Validar compatibilidad de librería.
4. Probar con bajo brillo inicialmente.
5. Analizar si introduce bloqueos innecesarios.

La IA puede proponer soluciones funcionales, pero la validación técnica es responsabilidad del ingeniero o estudiante.

15. Extensión IoT del Módulo: Control Web desde Dispositivo Móvil

¿Qué agrega esta extensión?

Permite controlar el anillo LED desde el navegador del celular, conectándose a una página web servida directamente por el ESP32.

Ventajas:

- No requiere app.
- No requiere computador después de cargar el programa.
- Permite visualizar diferencia entre control local y control remoto.

15.1 Objetivo de la extensión

Implementar una interfaz Web alojada directamente en el ESP32 que permita controlar el anillo LED desde el navegador de un celular, sin necesidad de aplicaciones externas.

Esta extensión introduce conceptos de:

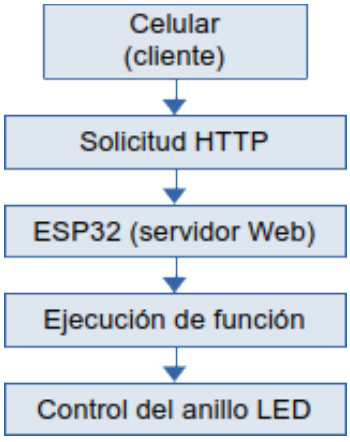
- Servidor Web embebido.
- Comunicación cliente–servidor.
- Arquitectura IoT básica.
- Control remoto de dispositivos físicos.

El circuito eléctrico no cambia.
Únicamente se reemplaza el programa cargado en el ESP32.

15.2 Arquitectura del sistema

El ESP32 actúa como un **servidor Web embebido**, mientras el celular funciona como **cliente**.

Flujo de funcionamiento:



Este modelo representa una arquitectura IoT simplificada.

15.3 Modos de funcionamiento de red

El código incluido permite trabajar en tres modos, seleccionando únicamente una línea:

```
WifiMode WIFI_MODE = MODE_AP;
```

MODE_AP (Access Point)

- El ESP32 crea su propia red WiFi.
- No requiere router ni internet.
- Dirección IP fija: 192.168.4.1
- Recomendado para laboratorio o aula.

MODE_STA (Station)

- El ESP32 se conecta a un router existente.
- Recibe una IP asignada por el router.
- Requiere conocer SSID y contraseña.
- Simula una integración IoT real.

MODE_AP_STA

- Activa ambos modos simultáneamente.
- Permite acceso local y por router.
- Nivel avanzado.

```
#include <WiFi.h>
#include <WebServer.h>
#include <Adafruit_NeoPixel.h>

// ======================
// CONFIGURACIÓN PRINCIPAL
// ======================

// 1) Elige el modo de red (cambia SOLO esta línea)
enum WifiMode { MODE_AP, MODE_STA, MODE_AP_STA };
WifiMode WIFI_MODE = MODE_AP;   // <-- Cambia a MODE_STA o MODE_AP_STA

// 2) Datos para MODO AP (ESP32 crea su red)
const char* ap_ssid = "MesaSTEAM_LED";
const char* ap_password = "";   // vacío = sin contraseña (opcional)

// 3) Datos para MODO ROUTER / STA (ESP32 se conecta al router)
```

```
const char* sta_ssid = "TU_RED";
const char* sta_password = "TU_PASSWORD";

// ======================
// CONFIGURACIÓN ANILLO LED
// ======================
#define PIN 5
#define NUMPIXELS 16
Adafruit_NeoPixel pixels(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);

// ======================
// SERVIDOR WEB
// ======================
WebServer server(80);

// Página HTML simple (botones)
String page() {
  return R"rawliteral(
  <!DOCTYPE html>
  <html>
  <head>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Mesa STEAM - LED</title>
    <style>
      body{font-family:Arial;text-align:center;margin-top:40px;}
      button{padding:14px 18px;margin:8px;font-size:16px;}
      .box{max-width:420px;margin:auto;padding:10px;}
      code{background:#f2f2f2;padding:2px 6px;border-radius:6px;}
    </style>
  </head>
  <body>
    <div class="box">
      <h2>Control Anillo LED - Mesa STEAM</h2>
      <p>Seleccione un color:</p>
```

```
    <button onclick="fetch('/red')">Rojo</button>
    <button onclick="fetch('/green')">Verde</button>
    <button onclick="fetch('/blue')">Azul</button>
    <button onclick="fetch('/off')">Apagar</button>
    <p style="margin-top:25px;font-size:14px;">
      Si los botones no responden, refresque la pagina.
    </p>
   </div>
  </body>
  </html>
  )rawliteral";
}

void setAll(uint8_t r, uint8_t g, uint8_t b) {
  for (int i = 0; i < NUMPIXELS; i++) {
    pixels.setPixelColor(i, pixels.Color(r, g, b));
  }
  pixels.show();
}

void handleRoot() { server.send(200, "text/html", page()); }

// =======================
// WIFI: UTILIDADES
// =======================
void printNetworkInfo() {
  Serial.println("===== RED ACTIVA =====");

  // Si AP está activo, el ESP32 tiene IP fija 192.168.4.1
  if (WiFi.getMode() == WIFI_AP || WiFi.getMode() == WIFI_AP_STA) {
    Serial.print("AP SSID: ");
    Serial.println(ap_ssid);
    Serial.print("AP IP:   ");
    Serial.println(WiFi.softAPIP()); // normalmente 192.168.4.1
```

```
  }

  // Si STA está activo, toma IP del router
  if (WiFi.getMode() == WIFI_STA || WiFi.getMode() == WIFI_AP_STA) {
    Serial.print("STA SSID: ");
    Serial.println(sta_ssid);
    Serial.print("STA IP:   ");
    Serial.println(WiFi.localIP());
  }

  Serial.println("======================");
}

void startWiFi() {
  if (WIFI_MODE == MODE_AP) {
    WiFi.mode(WIFI_AP);
    WiFi.softAP(ap_ssid, ap_password);

  } else if (WIFI_MODE == MODE_STA) {
    WiFi.mode(WIFI_STA);
    WiFi.begin(sta_ssid, sta_password);

    // Esperar conexión al router (máximo 15 segundos)
    unsigned long startAttempt = millis();
    while (WiFi.status() != WL_CONNECTED && millis() - startAttempt < 15000) {
      delay(300);
      Serial.print(".");
    }
    Serial.println();

    // Si no conecta, cae automáticamente a AP para que el estudiante no quede bloqueado
    if (WiFi.status() != WL_CONNECTED) {
      Serial.println("No se pudo conectar al router. Cambiando a MODO AP...");
      WiFi.mode(WIFI_AP);
```

```
    WiFi.softAP(ap_ssid, ap_password);
  }

 } else { // MODE_AP_STA
  WiFi.mode(WIFI_AP_STA);
  WiFi.softAP(ap_ssid, ap_password);

  WiFi.begin(sta_ssid, sta_password);

  // Intentar conectar al router (máximo 15 segundos), pero mantener AP activo sí o sí
  unsigned long startAttempt = millis();
  while (WiFi.status() != WL_CONNECTED && millis() - startAttempt < 15000) {
    delay(300);
    Serial.print(".");
  }
  Serial.println();
 }

 printNetworkInfo();
}

// ======================
// SETUP Y LOOP
// ======================
void setup() {
 Serial.begin(115200);

 // LED ring init
 pixels.begin();
 pixels.setBrightness(100);
 pixels.clear();
 pixels.show();

 // WiFi init
```

```
  startWiFi();

  // Rutas del servidor
  server.on("/", handleRoot);
  server.on("/red", []{ setAll(255, 0, 0); server.send(200, "text/plain", "OK"); });
  server.on("/green", []{ setAll(0, 255, 0); server.send(200, "text/plain", "OK"); });
  server.on("/blue", []{ setAll(0, 0, 255); server.send(200, "text/plain", "OK"); });
  server.on("/off", []{ setAll(0, 0, 0); server.send(200, "text/plain", "OK"); });

  server.begin();
  Serial.println("Servidor web iniciado.");
}

void loop() {
  server.handleClient();
}
```

15.4 Procedimiento de prueba (Modo AP)

1. Cargar el código en el ESP32.
2. Encender el módulo.
3. Desde el celular, buscar la red WiFi:
   MesaSTEAM_LED
4. Conectarse a dicha red.
5. Abrir el navegador.
6. Escribir en la barra de direcciones:

```
192.168.4.1
```

7. Presionar los botones de la interfaz para cambiar el color del anillo.

15.5 Procedimiento de prueba (Modo Router)

1. Cambiar el modo a:

```
WifiMode WIFI_MODE = MODE_STA;
```

Configurar SSID y contraseña del router.

2. Cargar el programa.

3. Abrir el Monitor Serial.

4. Identificar la dirección IP asignada.

5. Desde el celular (conectado al mismo router), ingresar esa IP en el navegador.

15.6 Consumo energético con WiFi activo

Al habilitar conectividad inalámbrica, el consumo del ESP32 aumenta.

Consumo aproximado:

- 80 mA en reposo con WiFi activo.
- Hasta 240 mA en transmisión.

Si el anillo LED y el WiFi operan simultáneamente, el consumo total del sistema puede superar 1A en condiciones extremas.

Por esta razón, se recomienda utilizar:

Fuente regulada de 5V – 2A para funcionamiento estable.

15.7 Consideraciones de seguridad

Para entornos educativos se recomienda:

- Asignar contraseña al modo AP.
- No exponer el módulo a redes públicas sin configuración adicional.
- Evitar compartir credenciales reales en el código fuente.

15.8 Relación con la arquitectura Mesa STEAM

Esta extensión convierte el módulo en un sistema:

- Programable.
- Conectado.
- Escalable.
- Integrable con otros sensores.

Permite evolucionar el proyecto hacia:

- Automatización.
- Control remoto.
- Integración con sensores.
- Desarrollo de aplicaciones Web más complejas.

15.9 Uso estratégico de IA para modificación del código

El código presentado puede ser modificado utilizando herramientas de inteligencia artificial generativa. Sin embargo, es responsabilidad del estudiante validar técnicamente cada modificación antes de implementarla en el sistema físico.

15.10.1 Ejemplos de prompts técnicos

A continuación se presentan ejemplos de instrucciones que pueden utilizarse para extender el proyecto:

1. Agregar control por intensidad (slider de brillo)

> Modifica el código para que la página web incluya un control deslizante (slider) que permita variar el brillo del anillo LED entre 0 y 255. Asegúrate de que el cambio se refleje en tiempo real.

2. Agregar selección de color RGB personalizada

> Reemplaza los botones fijos por un selector de color HTML tipo input color y adapta el código del ESP32 para interpretar los valores RGB enviados desde la página web.

3. Agregar animaciones programadas

> Añade una función que genere un efecto tipo "rainbow cycle" en el anillo LED y crea un botón en la interfaz web que active dicha animación.

4. Implementar protección con contraseña en modo AP

> Modifica el código para que el modo Access Point tenga una contraseña de al menos 8 caracteres y explica los cambios realizados.

5. Optimizar consumo energético

> Ajusta el código para que el ESP32 entre en modo de bajo consumo cuando el anillo esté apagado durante más de 2 minutos.

15.10.2 Validación técnica obligatoria

Toda modificación generada mediante IA debe ser evaluada bajo los siguientes criterios:

- ¿Compila sin errores?
- ¿Respeta la estructura del servidor Web?
- ¿No modifica la configuración eléctrica del circuito?
- ¿No supera los límites de corriente calculados?
- ¿Mantiene la referencia común de tierra?

16. Matriz de Evaluación Mesa STEAM – Nivel Modular

Proyecto 1 – Sistema de Iluminación Inteligente IoT

Criterio	Nivel 1 Inicial	Nivel 2 En desarrollo	Nivel 3 Competente	Nivel 4 Avanzado
Montaje eléctrico	Conexiones incorrectas o sin referencia común	Montaje funcional con ajustes	Montaje correcto y estable	Montaje optimizado y justificado técnicamente
Programación base	Código no compila o	Código funcional con	Código funcional y	Código adaptado o

	presenta fallos	modificaciones mínimas	comprendido	mejorado
Control Web	No logra conexión	Conexión parcial	Interfaz funcional	Interfaz modificada o ampliada
Análisis energético	No realiza cálculo	Cálculo incompleto	Cálculo correcto	Análisis comparativo y optimización
Uso crítico de IA	Copia sin validar	Ajustes básicos	Valida funcionamiento	Integra mejoras justificadas
Documentación	No documenta	Documenta parcialmente	Evidencia clara del proceso	Documenta, reflexiona técnicamente y propone mejoras justificadas

17. Bitácora Técnica Mesa STEAM – Nivel Modular

Proyecto: _____
Módulo: _____
Fecha: _____
Integrantes: _____

1. Configuración eléctrica implementada

Describa detalladamente cómo se realizó el montaje eléctrico.
Incluya referencia a alimentación, tierra común, distribución de energía y posibles ajustes realizados.

2. Decisiones de diseño adoptadas

Explique por qué seleccionó esta configuración eléctrica y esta estructura de programación.
Indique si consideró alternativas y por qué fueron descartadas.

3. Resultados obtenidos

Describa el comportamiento del sistema una vez energizado.
Indique si el funcionamiento fue el esperado y qué observaciones técnicas realizó.

4. Análisis técnico y energético

Registre el consumo estimado del sistema.
Justifique la fuente de alimentación utilizada (voltaje y corriente).
Indique si el sistema opera dentro de los márgenes seguros calculados.

5. Dificultades técnicas y resolución de problemas

Describa errores presentados (eléctricos, lógicos o de conectividad).
Explique cómo fueron diagnosticados y solucionados.

6. Modificaciones o extensiones realizadas

Indique si realizó cambios al código base o a la arquitectura.
En caso de utilizar herramientas de inteligencia artificial, describa el prompt utilizado y explique cómo validó técnicamente la modificación.

7. Propuesta de mejora para siguiente iteración

Proponga una mejora técnica o funcional para optimizar el sistema en una futura versión.

Observación metodológica

Esta bitácora debe completarse una vez finalizada la implementación del módulo y constituye evidencia del proceso de aprendizaje, validación técnica e iteración del sistema desarrollado.