Centro de Enseñanza Técnica Industrial

Desarrollo de Software

Proyecto - Lightsaber

Jesús Alberto Aréchiga Carrillo

22310439 7N

Profesor

José Francisco Pérez Reyes

Diciembre 2025

Guadalajara, Jalisco

```
// Configuración de Blynk
#define BLYNK_TEMPLATE_ID "TMPL2BaTNRQbp"
#define BLYNK_TEMPLATE_NAME "Sable de Luz"
#define BLYNK_AUTH_TOKEN "nnGIXXQV1ERfw6NbyUJV8ubZzkwvd6nu"

#define BLYNK_PRINT Serial

#include <WiFi.h>
#include <BlynkSimpleEsp32.h>
#include <FastLED.h>
#include <DFRobotDFPlayerMini.h>
#include <Wire.h>
#include <MPU9250_asukiaaa.h>

// ===== CONFIGURACIÓN WIFI =====
char ssid[] = "Wifi no disponible";
char pass[] = "Letmein!";

// ===== CONFIGURACIÓN LED STRIP =====
#define LED_PIN 12
#define NUM_LEDS 16
#define LED_TYPE WS2812B
#define COLOR_ORDER GRB
#define MAX_BRIGHTNESS 80
CRGB leds[NUM_LEDS];

// ===== CONFIGURACIÓN DFPLAYER =====
HardwareSerial dfPlayerSerial(1);
DFRobotDFPlayerMini dfPlayer;

// ===== CONFIGURACIÓN MPU9250 =====
MPU9250_asukiaaa mpu;

// ===== MODOS/PRESETS =====
enum SaberMode {
  MODE_JEDI = 0,
  MODE_SITH = 1,
  MODE_CUSTOM = 2
};

enum SoundTheme {
  SOUND_JEDI = 0,
  SOUND_SITH = 1,
  SOUND_CUSTOM = 2
};

// ===== VARIABLES DE CONTROL =====
bool saberOn = false;
SaberMode currentMode = MODE_JEDI;
```

```
SoundTheme currentSoundTheme = SOUND_JEDI;
CRGB currentColor = CRGB::Blue;
int brightness = 60;
bool customMode = false;

// Variables para detección de movimiento
float prevAccelMagnitude = 0;
unsigned long lastSwingTime = 0;
unsigned long lastClashTime = 0;

// ===== PINES =====
#define DFPLAYER_RX 16
#define DFPLAYER_TX 17

// ===== ARCHIVOS DE AUDIO (Carpeta 01) =====
// 001 - Encendido Jedi
// 002 - Apagado Jedi
// 003 - Encendido Sith
// 004 - Apagado Sith
// 005 - Zumbido en loop
// 006-009 - Swing
// 010-012 - Clash
// 013 - Canción Jedi
// 014 - Canción Sith

#define AUDIO_FOLDER 1

// ===== BLYNK VIRTUAL PINS =====
// V0 - Power (Switch ON/OFF)
// V1 - Red (0-255) - Solo activo en Custom
// V2 - Green (0-255) - Solo activo en Custom
// V3 - Blue (0-255) - Solo activo en Custom
// V4 - Preset Mode (0=Jedi, 1=Sith, 2=Custom)
// V5 - Brightness (0-80)
// V6 - Song (0=Stop, 1=Jedi, 2=Sith)
// V7 - Sound Theme (0=Jedi, 1=Sith, 2=Custom) - Solo activo en Custom

void setup() {
  Serial.begin(115200);

  // Inicializar WiFi y Blynk
  Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);
  Serial.println("Conectando a Blynk...");

  // Inicializar LED Strip
  FastLED.addLeds<LED_TYPE, LED_PIN, COLOR_ORDER>(leds, NUM_LEDS);
  FastLED.setMaxPowerInVoltsAndMilliamps(5, 350);
  FastLED.setBrightness(brightness);
  FastLED.clear();
```

```
  FastLED.show();
  Serial.println("LED Strip inicializado");

  // Inicializar DFPlayer
  dfPlayerSerial.begin(9600, SERIAL_8N1, DFPLAYER_RX, DFPLAYER_TX);
  delay(500);

  if (!dfPlayer.begin(dfPlayerSerial)) {
    Serial.println("Error al inicializar DFPlayer!");
    Serial.println("Verifica:");
    Serial.println("1. Conexiones RX/TX");
    Serial.println("2. MicroSD insertada y formateada FAT32");
    Serial.println("3. Archivos en carpeta /01/");
  } else {
    Serial.println("DFPlayer inicializado correctamente");
    dfPlayer.volume(25);  // Volumen 0-30
    delay(100);
  }

  // Inicializar MPU9250
  Wire.begin();
  mpu.setWire(&Wire);
  mpu.beginAccel();
  mpu.beginGyro();
  Serial.println("MPU9250 inicializado");

  // Aplicar preset inicial (Jedi)
  applyPreset(MODE_JEDI);

  delay(1000);
}

void loop() {
  Blynk.run();

  if (saberOn) {
    checkMotion();
    updateLEDs();
  }

  delay(10);
}

// ===== FUNCIONES DE PRESETS =====

void applyPreset(SaberMode mode) {
  currentMode = mode;
  bool wasOn = saberOn;
```

```
switch(mode) {
  case MODE_JEDI:
    currentColor = CRGB(0, 0, 255);  // Azul
    currentSoundTheme = SOUND_JEDI;
    customMode = false;
    Serial.println("Preset aplicado: JEDI (Azul)");

    Blynk.virtualWrite(V1, 0);    // Red
    Blynk.virtualWrite(V2, 0);    // Green
    Blynk.virtualWrite(V3, 255);  // Blue
    Blynk.virtualWrite(V7, 0);    // Sound Theme Jedi
    break;

  case MODE_SITH:
    currentColor = CRGB(255, 0, 0);  // Rojo
    currentSoundTheme = SOUND_SITH;
    customMode = false;
    Serial.println("Preset aplicado: SITH (Rojo)");

    Blynk.virtualWrite(V1, 255);  // Red
    Blynk.virtualWrite(V2, 0);    // Green
    Blynk.virtualWrite(V3, 0);    // Blue
    Blynk.virtualWrite(V7, 1);    // Sound Theme Sith
    break;

  case MODE_CUSTOM:
    customMode = true;
    Serial.println("Modo CUSTOM activado");
    break;
}

if (wasOn) {
  dfPlayer.stop();
  delay(100);

  // Reproducir sonido de encendido del nuevo preset
  if (currentSoundTheme == SOUND_JEDI) {
    dfPlayer.playFolder(AUDIO_FOLDER, 1);  // 001 - Encendido Jedi
    Serial.println("Reproduciendo: 01/001 - Encendido Jedi");
  } else if (currentSoundTheme == SOUND_SITH) {
    dfPlayer.playFolder(AUDIO_FOLDER, 3);  // 003 - Encendido Sith
    Serial.println("Reproduciendo: 01/003 - Encendido Sith");
  }

  updateLEDs();

  // Esperar e iniciar zumbido
  delay(2000);
  dfPlayer.playFolder(AUDIO_FOLDER, 5);  // 005 - Zumbido
```

```
    dfPlayer.enableLoop();
    Serial.println("Zumbido iniciado: 01/005");
  }
}

// ===== FUNCIONES BLYNK =====

BLYNK_WRITE(V0) {
  int value = param.asInt();

  if (value == 1 && !saberOn) {
    saberOn = true;
    turnOnSaber();
  } else if (value == 0 && saberOn) {
    saberOn = false;
    turnOffSaber();
  }
}

BLYNK_WRITE(V1) {
  if (customMode) {
    int r = param.asInt();
    currentColor.r = r;
    Serial.printf("Custom - Rojo: %d\n", r);
  } else {
    Serial.println("Cambio de color bloqueado - Cambia a modo Custom");
  }
}

BLYNK_WRITE(V2) {
  if (customMode) {
    int g = param.asInt();
    currentColor.g = g;
    Serial.printf("Custom - Verde: %d\n", g);
  } else {
    Serial.println("Cambio de color bloqueado - Cambia a modo Custom");
  }
}

BLYNK_WRITE(V3) {
  if (customMode) {
    int b = param.asInt();
    currentColor.b = b;
    Serial.printf("Custom - Azul: %d\n", b);
  } else {
    Serial.println("Cambio de color bloqueado - Cambia a modo Custom");
  }
}
```

```
BLYNK_WRITE(V4) {
  int modeValue = param.asInt();

  if (modeValue >= 0 && modeValue <= 2) {
    applyPreset((SaberMode)(modeValue));
  }
}

BLYNK_WRITE(V5) {
  brightness = constrain(param.asInt(), 0, MAX_BRIGHTNESS);
  FastLED.setBrightness(brightness);
  Serial.printf("Brillo: %d\n", brightness);
}

BLYNK_WRITE(V6) {
  int songNumber = param.asInt();

  if (songNumber == 0) {
    dfPlayer.stop();
    Serial.println("Música detenida");
  } else if (songNumber == 1) {
    dfPlayer.playFolder(AUDIO_FOLDER, 13);  // 013 - Tema Jedi
    dfPlayer.enableLoop();
    Serial.println("Reproduciendo: 01/013 - Tema Jedi (loop)");
  } else if (songNumber == 2) {
    dfPlayer.playFolder(AUDIO_FOLDER, 14);  // 014 - Tema Sith
    dfPlayer.enableLoop();
    Serial.println("Reproduciendo: 01/014 - Tema Sith (loop)");
  }
}

BLYNK_WRITE(V7) {
  if (customMode) {
    int themeValue = param.asInt();

    if (themeValue >= 0 && themeValue <= 2) {
      currentSoundTheme = (SoundTheme)themeValue;

      if (currentSoundTheme == SOUND_JEDI) {
        Serial.println("Tema de sonido: JEDI");
      } else if (currentSoundTheme == SOUND_SITH) {
        Serial.println("Tema de sonido: SITH");
      } else {
        Serial.println("Tema de sonido: CUSTOM");
      }
    }
  } else {
    Serial.println("Cambio de sonido bloqueado - Cambia a modo Custom");
  }
```

```
}

// ===== FUNCIONES DEL SABLE =====

void turnOnSaber() {
  Serial.println("=== Encendiendo sable ===");

  // Limpiar estado del DFPlayer
  dfPlayer.stop();
  dfPlayer.disableLoop();
  delay(200);

  // Sonido de encendido según el tema
  if (currentSoundTheme == SOUND_JEDI) {
    dfPlayer.playFolder(AUDIO_FOLDER, 1);  // 001 - Encendido Jedi
    Serial.println("Reproduciendo: 01/001 - Encendido Jedi");
  } else if (currentSoundTheme == SOUND_SITH) {
    dfPlayer.playFolder(AUDIO_FOLDER, 3);  // 003 - Encendido Sith
    Serial.println("Reproduciendo: 01/003 - Encendido Sith");
  } else {
    dfPlayer.playFolder(AUDIO_FOLDER, 1);  // Default: Jedi
    Serial.println("Reproduciendo: 01/001 - Encendido Jedi (default)");
  }

  delay(500);

  // Efecto de LEDs primero
  for (int i = 0; i < NUM_LEDS; i++) {
    leds[i] = currentColor;
    FastLED.show();
    delay(10);
  }

  // Esperar a que termine el sonido de encendido
  delay(1500);

  // Iniciar zumbido en loop
  Serial.println("Iniciando zumbido...");
  dfPlayer.playFolder(AUDIO_FOLDER, 5);  // 005 - Zumbido
  delay(100);
  dfPlayer.enableLoop();
  Serial.println("Zumbido en loop: 01/005");

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

void turnOffSaber() {
  Serial.println("=== Apagando sable ===");
```

```
  // Detener zumbido
  dfPlayer.stop();
  dfPlayer.disableLoop();
  delay(200);

  // Sonido de apagado
  if (currentSoundTheme == SOUND_JEDI) {
    dfPlayer.playFolder(AUDIO_FOLDER, 2);  // 002 - Apagado Jedi
    Serial.println("Reproduciendo: 01/002 - Apagado Jedi");
  } else if (currentSoundTheme == SOUND_SITH) {
    dfPlayer.playFolder(AUDIO_FOLDER, 4);  // 004 - Apagado Sith
    Serial.println("Reproduciendo: 01/004 - Apagado Sith");
  } else {
    dfPlayer.playFolder(AUDIO_FOLDER, 2);  // Default: Jedi
    Serial.println("Reproduciendo: 01/002 - Apagado Jedi (default)");
  }

  delay(1000);

  // Efecto de apagado de LEDs
  for (int i = NUM_LEDS - 1; i >= 0; i--) {
    leds[i] = CRGB::Black;
    FastLED.show();
    delay(10);
  }

  delay(1000);
  dfPlayer.stop();

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

void updateLEDs() {
  fill_solid(leds, NUM_LEDS, currentColor);
  FastLED.show();
}

void checkMotion() {
  if (mpu.accelUpdate() == 0) {
    float aX = mpu.accelX();
    float aY = mpu.accelY();
    float aZ = mpu.accelZ();

    float accelMagnitude = sqrt(aX * aX + aY * aY + aZ * aZ);
    float accelChange = abs(accelMagnitude - prevAccelMagnitude);

    unsigned long currentTime = millis();

    // Detectar swing
```

```
    if (accelChange > 5.0 && (currentTime - lastSwingTime) > 300) {
      playSwingSound();
      lastSwingTime = currentTime;
    }

    // Detectar clash
    if (accelChange > 12.0 && (currentTime - lastClashTime) > 500) {
      playClashSound();
      flashEffect();
      lastClashTime = currentTime;
    }

    prevAccelMagnitude = accelMagnitude;
  }
}

void playSwingSound() {
  // Archivos 006-009 para swings
  int soundFile = 6 + random(0, 4);

  // Pausar zumbido
  dfPlayer.pause();
  delay(50);

  // Reproducir swing
  dfPlayer.playFolder(AUDIO_FOLDER, soundFile);
  Serial.printf("Swing! 01/%03d\n", soundFile);

  // Reanudar zumbido
  delay(500);
  dfPlayer.start();
}

void playClashSound() {
  // Archivos 010-012 para clash
  int soundFile = 10 + random(0, 3);

  // Pausar zumbido
  dfPlayer.pause();
  delay(50);

  // Reproducir clash
  dfPlayer.playFolder(AUDIO_FOLDER, soundFile);
  Serial.printf("Clash! 01/%03d\n", soundFile);

  // Reanudar zumbido
  delay(600);
  dfPlayer.start();
}
```

```
void flashEffect() {
  CRGB originalColor = currentColor;

  for (int i = 0; i < 2; i++) {
    fill_solid(leds, NUM_LEDS, CRGB::White);
    FastLED.show();
    delay(40);
    fill_solid(leds, NUM_LEDS, originalColor);
    FastLED.show();
    delay(40);
  }
}
```

Información de la aplicación

El proyecto combina sensores de movimiento, iluminación LED programable de tipo RGB y un sistema de reproducción de audio sincronizado, todo ello controlable de forma remota a través de una aplicación móvil o interfaz web mediante la plataforma Blynk IoT. La aplicación permite no solo controlar manualmente los aspectos visuales y sonoros del dispositivo, sino que también implementa detección automática de movimientos para generar respuestas en tiempo real, simulando de manera realista el comportamiento de un sable de luz.

El proyecto se diseñó siguiendo el modelo de arquitectura de tres capas característico de los sistemas de Internet de las Cosas, lo que garantiza una separación clara de responsabilidades entre el procesamiento local en el dispositivo, la gestión de datos en la nube y la interfaz de usuario. Esta arquitectura modular facilita el mantenimiento, escalabilidad y mejoras futuras del sistema.

Modelo de 3 capas

Capa 1: Sistema embebido

Esta capa utiliza un microcontrolador ESP32 como núcleo del sistema, seleccionado por su conectividad WiFi-integrada y capacidad de procesamiento. El ESP32 gestiona tres componentes principales: el sensor MPU9250 que detecta movimientos mediante comunicación I2C, una tira LED WS2812B de 16 LEDs RGB controlados individualmente, y un módulo DFPlayer Mini que reproduce archivos MP3 desde una microSD mediante UART.

El sistema procesa localmente los datos del acelerómetro para detectar dos tipos de movimientos: swings (movimientos rápidos con umbral >5.0 G) y clashes

(impactos fuertes con umbral >12.0 G). Cada detección genera automáticamente efectos visuales y sonoros apropiados sin necesidad de comunicación con la nube, reduciendo significativamente la latencia de respuesta. El ESP32 también maneja la sincronización entre LEDs y audio, gestiona estados del dispositivo y mantiene comunicación bidireccional con el servidor Blynk.

Capa 2: Servidor en la nube

La infraestructura en la nube se implementa mediante Blynk IoT, una plataforma especializada en aplicaciones IoT que proporciona backend completo sin necesidad de servidores propios. La comunicación entre el ESP32 y Blynk utiliza el protocolo MQTT sobre conexiones TCP/IP seguras, autenticadas mediante un Auth Token único generado durante la configuración inicial.

El servidor gestiona ocho datastreams virtuales (V0-V7) que actúan como canales bidireccionales entre el dispositivo y los clientes. V0 controla el encendido/apagado, V1-V3 transmiten valores RGB (0-255), V4 selecciona entre tres presets (Jedi/Sith/Custom), V5 controla el brillo (0-80), V6 gestiona la reproducción musical, y V7 permite seleccionar el tema de sonido en modo Custom. Blynk almacena el último estado conocido de cada datastream, implementa reconexión automática y sincroniza el estado entre múltiples clientes conectados simultáneamente.

Capa 3: Cliente

Blynk proporciona dos tipos de interfaces: aplicaciones móviles nativas para iOS/Android y un dashboard web accesible desde cualquier navegador. Ambas interfaces utilizan widgets configurables vinculados a los datastreams del servidor, incluyendo un switch de encendido, selector de presets, tres sliders RGB sincronizados, control de brillo, selector de tema de sonido (activo solo en modo Custom) y controles de reproducción musical.

Las interfaces actualizan automáticamente sus elementos visuales cuando cambia el estado del dispositivo, ya sea por comandos de otros usuarios o cambios del propio hardware. También implementan validaciones que desactivan controles no aplicables según el contexto actual, proporcionan retroalimentación visual inmediata y manejan apropiadamente situaciones de desconexión temporal.

Características Principales del Sistema

Control Remoto

Los usuarios pueden encender/apagar el sable remotamente, cambiar el color en tiempo real mediante sliders RGB (más de 16 millones de combinaciones), ajustar el brillo de 0 a 80 para mantener consumo seguro, y reproducir música temática en loop continuo. Los cambios se aplican instantáneamente sin necesidad de reiniciar el dispositivo.

Sistema de Presets

El Modo Jedi configura color azul y sonidos del lado luminoso. El Modo Sith establece color rojo y sonidos del lado oscuro. El Modo Custom permite personalización completa de color y tema de sonido, habilitando combinaciones creativas como un sable verde con sonidos Sith. Los presets se aplican de forma coordinada incluso si el sable ya está encendido.

Detección Automática de Movimiento

El sistema analiza continuamente datos del acelerómetro calculando la magnitud vectorial de aceleración. Los swings (>5.0 G, cooldown 300ms) pausan el zumbido continuo, reproducen uno de cuatro sonidos aleatorios y reanudan el zumbido automáticamente. Los clashes (>12.0 G, cooldown 500ms) activan un flash blanco de LEDs y reproducen sonidos de impacto más dramáticos.

Efectos Visuales

Encendido progresivo LED por LED (10ms entre cada uno) sincronizado con el audio. Apagado progresivo en orden reverso. Flash de impacto durante clashes con parpadeos blancos de 40ms. Actualización continua de color sólido que refleja cambios remotos instantáneamente.

Sistema de Audio

Zumbido continuo en loop (archivo 005) mientras el sable está encendido. Sonidos de encendido/apagado específicos por preset (Jedi: 001/002, Sith: 003/004). Efectos de swing (006-009) y clash (010-012) que pausan y reanudan el zumbido automáticamente. Música temática opcional (013 Jedi, 014 Sith) que coexiste con el zumbido y efectos mediante mezcla interna del DFPlayer.

Optimización Energética

El diseño completo está optimizado para operar con alimentación USB estándar (límite 500mA). Se implementaron tres estrategias principales: reducción del número de LEDs a 16 unidades, limitación del brillo máximo al 31% (80/255) y protección por hardware mediante FastLED.setMaxPowerInVoltsAndMilliamps(5, 350) que ajusta dinámicamente el brillo según el presupuesto de corriente. El consumo típico es de aproximadamente 120mA para el ESP32 con WiFi y 200-250mA para los LEDs, totalizando 320-370mA y dejando un margen de seguridad que absorbe variaciones y el consumo del DFPlayer (30-40mA durante reproducción). Esta optimización asegura operación estable sin riesgo de sobrecorriente que podría causar reinicios inesperados.

Configuración de servicios

Configuración de Blynk IoT

Para configurar la plataforma Blynk IoT, primero es necesario registrarse en https://blynk.cloud y crear un nuevo Template con el nombre "Sable de Luz", seleccionando ESP32 como hardware y WiFi como tipo de conexión. Una vez creado el template, se deben configurar ocho datastreams que actúan como canales de comunicación entre el dispositivo y las aplicaciones cliente.

El datastream V0 (Power) es de tipo Integer con rango 0-1 y utiliza un widget Switch para controlar el encendido y apagado. Los datastreams V1 (Red), V2 (Green) y V3 (Blue) son de tipo Integer con rango 0-255, utilizando widgets Slider para control RGB individual. El datastream V4 (Preset Mode) es de tipo Enumerable con valores 0-2 correspondientes a los modos Jedi, Sith y Custom, visualizado mediante un widget Menu. V5 (Brightness) es Integer con rango 0-80 usando Slider. V6 (Song) es Integer con rango 0-2 para controlar la reproducción musical mediante Menu. Finalmente, V7 (Custom Theme) es Enumerable con valores 0-2 para seleccionar Jedi Sounds, Sith Sounds o Custom Sounds.

Después de configurar los datastreams, se debe crear un nuevo dispositivo seleccionando "From Template" y eligiendo el template "Sable de Luz". El sistema generará un Template ID y un Auth Token único que se utilizarán en el código del ESP32 para autenticación. Estos valores deben copiarse y guardarse de forma segura. El dashboard se configura arrastrando widgets al canvas y asignando cada uno a su datastream correspondiente, creando una interfaz intuitiva con switch, sliders y menús organizados visualmente.

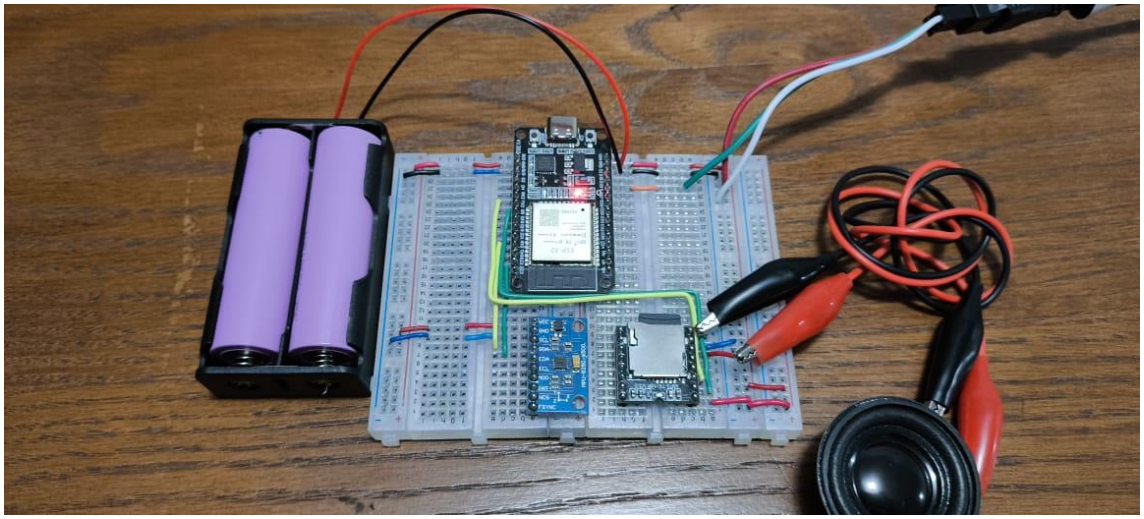
Configuración del código

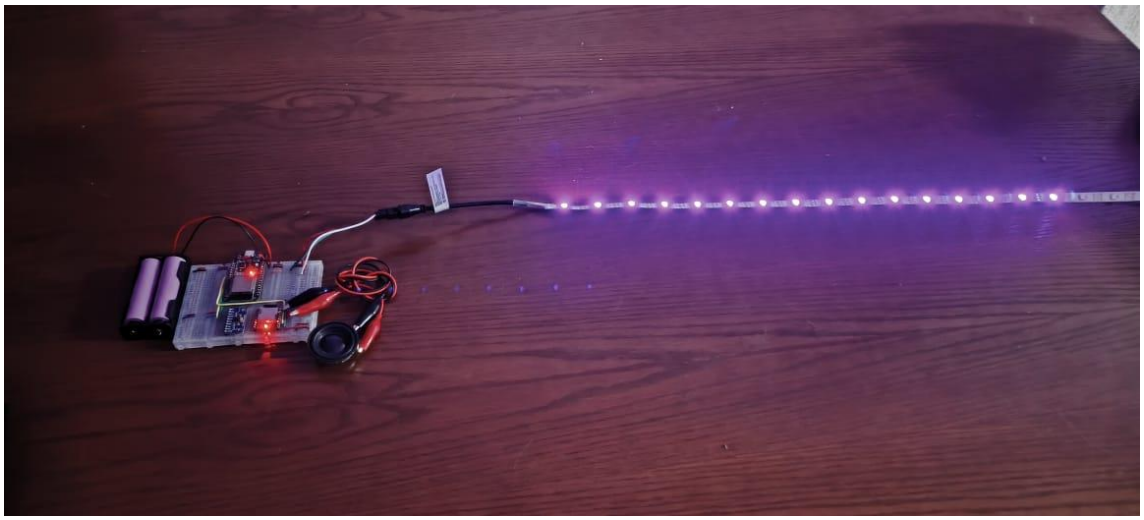
En el archivo lightsaber.ino se deben modificar tres secciones críticas para adaptar el código a cada instalación específica. En las líneas 8-10 se encuentran las credenciales de Blynk donde debe reemplazarse "TU_TEMPLATE_ID" con el Template ID obtenido de Blynk y "TU_AUTH_TOKEN" con el Auth Token único del dispositivo. En las líneas 22-23 se configuran las credenciales de la red WiFi, reemplazando "NOMBRE_DE_TU_WIFI" con el SSID de la red y "CONTRASEÑA_WIFI" con la contraseña correspondiente. Es importante asegurar que la red WiFi sea de 2.4GHz ya que el ESP32 no soporta bandas de 5GHz.

Preparación de microSD

La tarjeta microSD debe formatearse en sistema de archivos FAT32 con tamaño de unidad de asignación de 4096 bytes. El tamaño máximo soportado por el DFPlayer Mini es de 32GB. Una vez formateada, se debe crear una carpeta llamada "01" en la raíz de la tarjeta. Dentro de esta carpeta se copian los archivos de audio previamente renombrados con nomenclatura de 3 dígitos: 001.mp3 para encendido Jedi, 002.mp3 para apagado Jedi, 003.mp3 para encendido Sith, 004.mp3 para apagado Sith, 005.mp3 para el zumbido continuo, archivos 006.mp3 a 009.mp3 para efectos de swing, 010.mp3 a 012.mp3 para efectos de clash, 013.mp3 para la canción temática Jedi y 014.mp3 para la canción temática Sith. Es fundamental no incluir ningún otro archivo en la tarjeta, especialmente archivos .txt o documentos, ya que podrían interferir con el funcionamiento del DFPlayer. El formato de audio debe ser MP3 con bitrate recomendado de 128kbps o menor para garantizar reproducción fluida.

Evidencias





```
[1668] Connected to WiFi
[1668] IP: 192.168.1.92
[1668]
    ___  __          __
   / _ )/ /_ _____  / /__
  / _  / / // / _ \/  '_/
 /____/_/\_, /_//_/_/\_\
        /___/ v1.3.2 on ESP32

 #StandWithUkraine    https://bit.ly/swua


[1679] Connecting to blynk.cloud:80
[2192] Ready (ping: 159ms).
Conectando a Blynk...
LED Strip inicializado
DFPlayer inicializado correctamente
MPU9250 inicializado
Preset aplicado: JEDI (Azul)
Preset aplicado: JEDI (Azul)
Preset aplicado: SITH (Rojo)
=== Encendiendo sable ===
Reproduciendo: 01/003 - Encendido Sith
Iniciando zumbido...
Zumbido en loop: 01/005
=== Sable encendido ===
=== Apagando sable ===
Reproduciendo: 01/004 - Apagado Sith
=== Sable apagado ===
Reproduciendo: 01/013 - Tema Jedi (loop)
Reproduciendo: 01/014 - Tema Sith (loop)
Modo CUSTOM activado
Custom - Verde: 255
Custom - Rojo: 0
Custom - Azul: 255
Custom - Rojo: 255
Custom - Verde: 0
Custom - Rojo: 0
```