¿Qué es BLE (Bluetooth Low Energy)?

BLE es una versión optimizada de Bluetooth tradicional que está diseñada para consumir mucha menos energía. Se usa principalmente en aplicaciones de IoT, como sensores y dispositivos que necesitan estar conectados todo el tiempo, pero sin usar demasiada batería.

Librerías que usamos:

BLEDevice.h: Esta es la librería principal para manejar el dispositivo BLE.

BLEServer.h: Se usa para crear y manejar el servidor BLE en el ESP32.

BLECharacteristic.h: Define características BLE, que son como variables que otros dispositivos pueden leer o escribir.

¿Cómo funciona?

Primero, configuramos el ESP32 como un **servidor BLE**, luego definimos un **UUID** (Identificador Único Universal) para nuestro servicio y una **característica** que otros dispositivos podrán leer y escribir.

```
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>

#define SERVICE_UUID "12345678-1234-1234-1234-123456789012"  // UUID único del servicio
#define CHARACTERISTIC_UUID "87654321-4321-4321-4321-210987654321"  // UUID de la caracter

bool ledState = false;
const int ledPin = 2;  // Definimos el pin donde está conectado el LED

BLEServer* pServer;
BLECharacteristic* pCharacteristic;

class MyCallbacks: public BLECharacteristicCallbacks {
  void onWrite(BLECharacteristic *pCharacteristic) {
    std::string rxValue = pCharacteristic->getValue();
    if (rxValue == "on") {
      digitalWrite(ledPin, HIGH);
    } else if (rxValue == "off") {
      digitalWrite(ledPin, LOW);
    }
  }
};

void setup() {
  Serial.begin(115200);
  pinMode(ledPin, OUTPUT);

  BLEDevice::init("ESP32_BLE");  // Configuramos el nombre del dispositivo
  pServer = BLEDevice::createServer();

  BLEService *pService = pServer->createService(SERVICE_UUID);
  pCharacteristic = pService->createCharacteristic(
    CHARACTERISTIC_UUID,
    BLECharacteristic::PROPERTY_READ | BLECharacteristic::PROPERTY_WRITE
  );

  pCharacteristic->setCallbacks(new MyCallbacks());
  pService->start();
  pServer->getAdvertising()->start();
}

void loop() {
```

BLEDevice::init(): Inicia el dispositivo BLE y establece el nombre que aparecerá cuando lo escanees con tu teléfono.

BLEServer:createService(): Crea un servicio BLE con un UUID específico.

BLECharacteristic:createCharacteristic(): Define una característica dentro de ese servicio, que puede leerse y escribirse.

Callbacks: La lógica que maneja la escritura en la característica BLE se ejecuta aquí. En este caso, al recibir "on" o "off", encendemos o apagamos el LED.

Comunicación Serial con Bluetooth Clásico

¿Qué es Bluetooth Clásico (SPP)?

El perfil de puerto serie (SPP) permite una comunicación directa y constante entre el ESP32 y otros dispositivos, como un teléfono o una PC, usando Bluetooth clásico. Se usa cuando necesitas transferir más datos o tener una conexión estable y continua.

Librerías que usamos:

BluetoothSerial.h: Proporciona las herramientas necesarias para que el ESP32 funcione como un dispositivo Bluetooth Serial.

```
#include "BluetoothSerial.h"

BluetoothSerial SerialBT;  // Instanciamos un objeto para manejar Bluetooth Serial

void setup() {
  Serial.begin(115200);
  SerialBT.begin("ESP32_BT");  // Nombre del dispositivo Bluetooth
  Serial.println("Bluetooth Iniciado. ¡Listo para emparejarse!");
}

void loop() {
  if (SerialBT.available()) {
    char incomingChar = SerialBT.read();  // Leemos los datos recibidos por Bluetooth
    Serial.print(incomingChar);  // Mostramos el dato recibido en el Monitor Serie
    if (incomingChar == '1') {
      Serial.println("Encender LED");
      // Aquí podrías encender un LED
    } else if (incomingChar == '0') {
      Serial.println("Apagar LED");
      // Aquí podrías apagar un LED
    }
  }
}
```

SerialBT.begin("ESP32_BT"): Configura el ESP32 como un dispositivo Bluetooth que otros dispositivos pueden encontrar y emparejar.

SerialBT.read(): Permite leer los datos recibidos por Bluetooth.

if (incomingChar == '1'): Con esto, puedes controlar dispositivos como un LED según el valor recibido.

Crear un Servidor Web en el ESP32

¿Qué es un servidor web en ESP32?

El ESP32 puede actuar como un pequeño servidor web. Esto te permite controlarlo desde cualquier navegador web, ideal para interfaces gráficas o control remoto.

Librerías que usamos:

WiFi.h: Maneja la conexión Wi-Fi del ESP32.

WebServer.h: Permite crear un servidor web que puede manejar múltiples rutas.

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

const char* ssid = "TuSSID";  // Nombre de tu red Wi-Fi
const char* password = "TuPassword";  // Contraseña de tu red

WebServer server(80);  // Creamos el servidor en el puerto 80

void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, password);  // Conectamos el ESP32 a la red Wi-Fi

  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Conectando a WiFi...");
  }

  Serial.println("Conectado a WiFi");

  server.on("/", [](){
    server.send(200, "text/html", "<h1>Servidor Web en ESP32</h1><button>Encender LED</bu
  });

  server.begin();  // Iniciamos el servidor
}

void loop() {
  server.handleClient();  // Manejamos las peticiones web
}
```

WiFi.begin(): Conecta el ESP32 a la red Wi-Fi usando el SSID y la contraseña.

server.on(): Define la página web que se muestra al acceder al servidor.

server.handleClient(): Maneja todas las peticiones que llegan al servidor.

ESP32 en Modo Punto de Acceso (AP Mode)

¿Qué es el modo AP en ESP32?

El modo **Punto de Acceso (AP)** permite que el ESP32 cree su propia red Wi-Fi. Esto es útil cuando no tienes una red Wi-Fi disponible o si quieres conectar directamente dispositivos a tu ESP32.

Librerías que usamos:

WiFi.h: La misma librería que usamos para conectarnos a una red, pero en este caso configuramos el ESP32 para crear una.

```
#include <WiFi.h>

const char* ssid = "ESP32_AP";  // Nombre del Punto de Acceso
const char* password = "12345678";  // Contraseña de la red

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

  // Configuramos el ESP32 en modo AP (Punto de Acceso)
  WiFi.softAP(ssid, password);

  Serial.println("Punto de acceso configurado.");
  Serial.print("Dirección IP: ");
  Serial.println(WiFi.softAPIP());  // Mostramos la IP del Punto de Acceso
}

void loop() {
```

WiFi.softAP(ssid, password): Configura el ESP32 como un Punto de Acceso (AP) con el nombre y contraseña definidos.

WiFi.softAPIP(): Muestra la dirección IP del ESP32 para que otros dispositivos puedan conectarse.