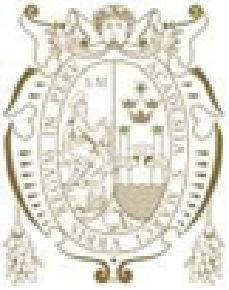


Universidad Nacional Mayor de San Marcos
Universidad del Perú. Decana de América

Facultad de Ingeniería de Sistemas e Informática

Proyecto de IoT: Sistema de Asistencia con ESP32 y Bluetooth

Curso Internet de las Cosas

AUTORES

Mitac Saavedra, Milena Diana	20200275
Rosas Sequeiros Fabricio	20200288
Salinas Mejías Ramsés Alfonzo	20200292

LIMA, PERÚ

2024

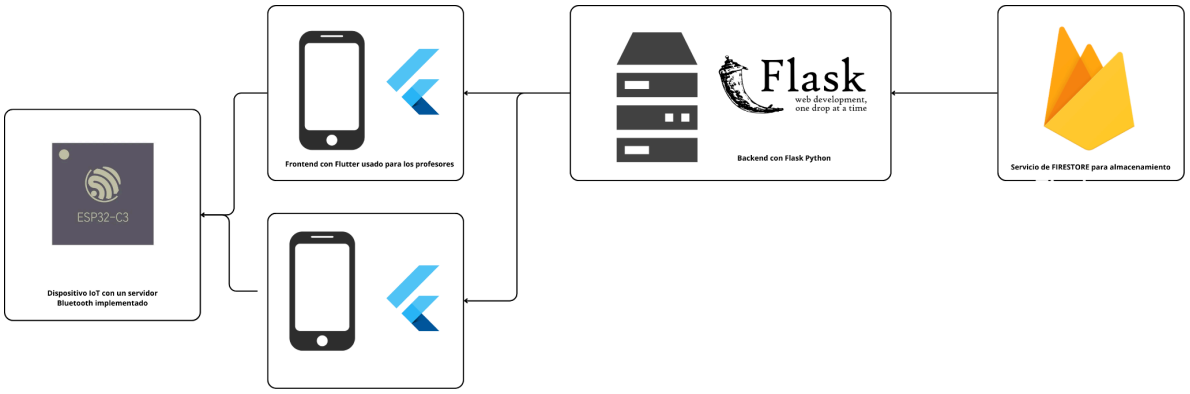
Índice

Objetivo

Desarrollar un sistema de asistencia sin dependencia de Internet, utilizando un ESP32 como servidor Bluetooth para registrar la asistencia en la facultad.

Descripción

El sistema consiste en un ESP32 que actúa como servidor Bluetooth. Los estudiantes se conectan al ESP32 con sus teléfonos móviles al inicio de la clase. El ESP32 recopila y almacena temporalmente los datos de los estudiantes conectados y, al finalizar un período determinado, transmite esta información a una aplicación móvil del profesor. La aplicación muestra una lista organizada de los estudiantes presentes, facilitando la gestión de asistencia en tiempo real.



Especificación del componente

ESP32

El ESP32 es un microcontrolador de bajo costo y bajo consumo con capacidades integradas de Wi-Fi y Bluetooth, diseñado y fabricado por Espressif Systems. Es una evolución del ESP8266, ofreciendo mayores prestaciones y funcionalidades adicionales. A continuación se proporciona una descripción detallada de sus características

Características Principales

Procesador

- CPU Dual-Core: Incluye dos núcleos Xtensa LX6 de 32 bits que pueden funcionar a una velocidad de reloj de hasta 240 MHz.
- Unidad de Punto Flotante (FPU): Soporta operaciones de punto flotante en hardware, mejorando el rendimiento en aplicaciones que requieren cálculos matemáticos intensivos.

Memoria:

- RAM: Dispone de 520 KB de SRAM interna.
- ROM: 448 KB de ROM para funciones internas.
- Flash Externa: Soporta hasta 16 MB de memoria Flash externa, lo que permite almacenamiento adicional para código y datos.

Conectividad:

- Wi-Fi: Compatible con los estándares 802.11 b/g/n, soportando tanto modo AP (Access Point) como modo estación.
- Bluetooth: Compatible con Bluetooth v4.2 y Bluetooth Low Energy (BLE), permitiendo conectividad con una amplia gama de dispositivos.

Interfaces de Periféricos:

- GPIO: Dispone de hasta 34 pines GPIO (General Purpose Input/Output), muchos de los cuales son multifuncionales.
- ADC: 18 canales de ADC (Analog to Digital Converter) de 12 bits.
- DAC: 2 canales de DAC (Digital to Analog Converter) de 8 bits.
- Interfaces Seriales: Incluye múltiples interfaces UART, SPI, I2C, I2S, y CAN.
- PWM: Soporte para múltiples canales PWM (Pulse Width Modulation), ideal para el control de motores y la generación de señales analógicas.

Seguridad:

- Criptografía: Soporte para criptografía avanzada incluyendo AES, SHA-2, RSA, y ECC.
- Secure Boot: Funcionalidad de arranque seguro para proteger contra modificaciones no autorizadas del firmware.
- Flash Encryption: Cifrado de la memoria Flash para proteger datos y código almacenado.

Sensores Integrados:

- Sensor de Temperatura: Integrado en el chip para monitorización de temperatura interna.

Modos de Bajo Consumo:

- Deep Sleep: Modo de sueño profundo con un consumo de energía extremadamente bajo (del orden de microamperios).
- Light Sleep y Modem Sleep: Modos de sueño con consumo reducido y tiempos de respuesta rápidos.

Desarrollo y Programación

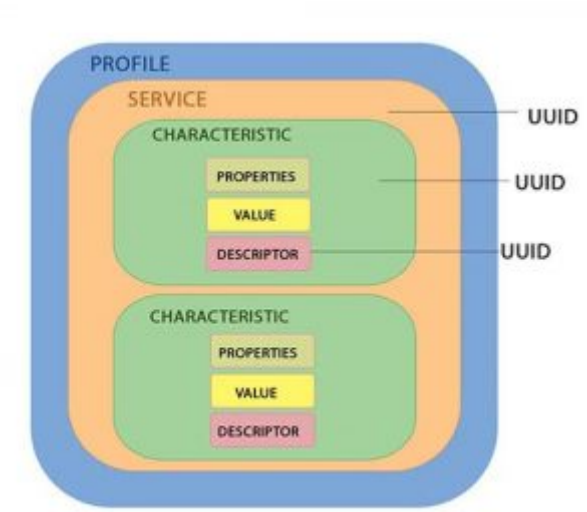
El ESP32 es compatible con múltiples entornos de desarrollo, entre los que destacan:

- Espressif IoT Development Framework (ESP-IDF): El entorno oficial de desarrollo proporcionado por Espressif, basado en C/C++.
- Arduino IDE: Amplia compatibilidad con el entorno de desarrollo Arduino, facilitando la transición para aquellos que ya están familiarizados con esta plataforma.
- MicroPython: Soporte para programación en Python, adecuado para desarrolladores que prefieren trabajar con lenguajes de alto nivel.
- PlatformIO: Un entorno de desarrollo basado en Visual Studio Code, que soporta múltiples plataformas de hardware y lenguajes de programación.

Bluetooth Low Energy (BLE)

Bluetooth Low Energy (BLE) es una tecnología de comunicación inalámbrica diseñada para aplicaciones de baja potencia y bajo consumo de energía. A diferencia del Bluetooth clásico, BLE está optimizado para transferencias de datos pequeñas y periódicas, lo cual lo hace ideal para dispositivos que necesitan operar con baterías durante largos períodos de tiempo. La arquitectura de BLE es:

- Central y Periférico: Los dispositivos BLE se dividen en centrales (central) y periféricos (peripheral). Los periféricos exponen servicios y características que las centrales pueden descubrir y utilizar.
- Servicios y Características: Los servicios agrupan características relacionadas y definen las operaciones disponibles. Cada característica puede ser de lectura, escritura o notificación.
- Conexión y Publicidad: Los dispositivos BLE pueden anunciarse (advertise) para que los dispositivos centrales los descubran. Una vez conectados, se establece una conexión BLE para la transferencia de datos.



Ventajas:

- **Sin Dependencia de Internet:** Ideal para áreas con poca cobertura.
- **Eficiencia:** Automatiza la toma de asistencia, ahorrando tiempo y reduciendo errores.
- **Accesibilidad:** Fácil de usar con mínima configuración.
- **Escalabilidad:** Implementable en múltiples aulas.

Consideraciones:

- **Rango de Bluetooth:** Adecuado para cubrir todo el aula.
- **Seguridad:** Autenticación para evitar conexiones no autorizadas.
- **Interfaz de Usuario:** Diseño intuitivo para la aplicación del profesor.

Este sistema optimiza el proceso de registro de asistencia, haciendo el entorno académico más eficiente y conectado.

Descripción de los servicios

Servicio de Cambio de Estado

Este servicio permite iniciar y detener un período de registro de asistencia. Utiliza una característica BLE con capacidad de escritura (PROPERTY_WRITE) para recibir comandos "start" y "stop" desde un cliente BLE. Cuando se recibe "start", se activa un periodo de asistencia, y cuando se recibe "stop", se detiene, reiniciando el contador de asistentes registrados.

Servicio de Registrar Asistencia

Este servicio permite a los clientes BLE escribir nombres de asistentes. Durante un período activo de asistencia (definido por el servicio anterior), los nombres enviados por los clientes se agregan a un arreglo local en el ESP32. Cada nombre se almacena como una cadena de texto en un arreglo de tamaño limitado (maxAsistentes).

Servicio de Leer Asistentes

Este servicio permite a los clientes BLE leer la lista actualizada de asistentes registrados. Utiliza una característica BLE con capacidad de lectura (PROPERTY_READ). Durante un período activo de asistencia, esta lista se actualiza dinámicamente con los nombres almacenados en el arreglo local.

Código para la implementación del servidor Bluetooth

```
#include <BLEDevice.h>

#include <BLEServer.h>

#include <BLEUtils.h>

#include <BLE2902.h>



// Variables globales

BLEServer* pServer = NULL;

BLECharacteristic* pCharacteristicEstado = NULL;

BLECharacteristic* pCharacteristicRegistrar = NULL;

BLECharacteristic* pCharacteristicLeer = NULL;



bool periodoActivo = false;



const int maxAsistentes = 100;

String asistentes[maxAsistentes];

int numAsistentes = 0;



// UUIDs para los servicios y características

#define SERVICE_UUID_CAMBIO_ESTADO
"11111111-1111-1111-1111-111111111111"

#define CHARACTERISTIC_UUID_CAMBIO_ESTADO
"11111111-1111-1111-1111-111111111112"



#define SERVICE_UUID_REGISTRAR "22222222-2222-2222-2222-222222222222"
```

```
#define CHARACTERISTIC_UUID_REGISTRAR
"22222222-2222-2222-2222-222222222223"



#define SERVICE_UUID_LEER "33333333-3333-3333-3333-333333333333"

#define CHARACTERISTIC_UUID_LEER "33333333-3333-3333-3333-333333333334"



class MyServerCallbacks: public BLEServerCallbacks {

    void onConnect(BLEServer* pServer) {

      BLEDevice::startAdvertising();

    };



    void onDisconnect(BLEServer* pServer) {

    }

};



class EstadoCallbacks: public BLECharacteristicCallbacks {

    void onWrite(BLECharacteristic *pCharacteristic) {

      std::string value = pCharacteristic->getValue();

      if (value == "start") {

        periodoActivo = true;

        Serial.println("Periodo de asistencia iniciado.");

      } else if (value == "stop") {

        periodoActivo = false;

        Serial.println("Periodo de asistencia pausado.");

        // Vaciar el arreglo de asistencia
```

```
        numAsistentes = 0;
      }
    }
};

class RegistrarCallbacks: public BLECharacteristicCallbacks {
    void onWrite(BLECharacteristic *pCharacteristic) {
      if (periodoActivo && numAsistentes < maxAsistentes) {
        std::string value = pCharacteristic->getValue();
        asistentes[numAsistentes] = String(value.c_str()); //
Conversion std::string a String
        numAsistentes++;
        Serial.println("Asistencia registrada: " +
String(value.c_str()));
      }
    }
};

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

  // Crear el dispositivo BLE
  BLEDevice::init("ESP32");

  // Crear el servidor BLE
  pServer = BLEDevice::createServer();
```

```
  pServer->setCallbacks(new MyServerCallbacks());


  // Crear el servicio BLE para cambio de estado
  BLEService *pServiceEstado =
pServer->createService(SERVICE_UUID_CAMBIO_ESTADO);


  // Crear una característica BLE para cambio de estado
  pCharacteristicEstado = pServiceEstado->createCharacteristic(
                      CHARACTERISTIC_UUID_CAMBIO_ESTADO,
                      BLECharacteristic::PROPERTY_WRITE
                    );
  pCharacteristicEstado->addDescriptor(new BLE2902());
  pCharacteristicEstado->setCallbacks(new EstadoCallbacks());


  // Iniciar el servicio de cambio de estado
  pServiceEstado->start();


  // Crear el servicio BLE para registrar asistencia
  BLEService *pServiceRegistrar =
pServer->createService(SERVICE_UUID_REGISTRAR);


  // Crear una característica BLE para registrar asistencia
  pCharacteristicRegistrar = pServiceRegistrar->createCharacteristic(
                      CHARACTERISTIC_UUID_REGISTRAR,
                      BLECharacteristic::PROPERTY_WRITE
                    );
```

```
pCharacteristicRegistrar->addDescriptor(new BLE2902());
pCharacteristicRegistrar->setCallbacks(new RegistrarCallbacks());

// Iniciar el servicio registrar asistencia
pServiceRegistrar->start();

// Crear el servicio BLE para leer asistentes
BLEService *pServiceLeer = pServer->createService(SERVICE_UUID_LEER);

// Crear una característica BLE para leer asistentes
pCharacteristicLeer = pServiceLeer->createCharacteristic(
                      CHARACTERISTIC_UUID_LEER,
                      BLECharacteristic::PROPERTY_READ
                    );
pCharacteristicLeer->addDescriptor(new BLE2902());

// Iniciar el servicio leer asistentes
pServiceLeer->start();

// Iniciar la publicidad
BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
pAdvertising->addServiceUUID(SERVICE_UUID_CAMBIO_ESTADO);
pAdvertising->addServiceUUID(SERVICE_UUID_REGISTRAR);
pAdvertising->addServiceUUID(SERVICE_UUID_LEER);
pAdvertising->setScanResponse(false);
```

```
  pAdvertising->setMinPreferred(0x0);

  BLEDevice::startAdvertising();

  Serial.println("Esperando conexión de un cliente para cambiar estado,
registrar o leer asistencia...");

}



void loop() {

    // Imprimir el arreglo de asistentes en el monitor serial

    Serial.println("Lista de Asistentes:");

    for (int i = 0; i < numAsistentes; i++) {

      Serial.println(asistentes[i]);

    }



    // Si hay un dispositivo conectado, enviar la lista de asistentes

    if (periodoActivo) {

        String asistentesStr = "";

        for (int i = 0; i < numAsistentes; i++) {

            asistentesStr += asistentes[i] + "\n";

        }

        pCharacteristicLeer->setValue(asistentesStr.c_str());

    }



    delay(10000); // Esperar 10 segundos antes de imprimir de nuevo

}
```