UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS
Universidad del Perú, DECANA DE AMÉRICA
FACULTAD DE INGENIERÍA DE SISTEMA E INFORMÁTICA
ESCUELA PROFESIONAL DE INGENIERÍA DE SOFTWARE

INTERNET OF THINGS

Sistema de Seguridad y Control de Acceso Inteligente Utilizando AWS IoT

DOCENTE

Yessica Rosas Cueva

INTEGRANTES:

Triveño Ruffner, Daniel Huber

Lima - Perú
2024

INDICE

1. **Introducción**
    - 1.1. **Revision del Estado del Arte**
    - 1.2. **Planteamiento del problema**
    - 1.3. **Objetivos**
2. **Marco Terorico**
3. **Componentes del Sistema**
    - 3.1. **ESP32**
    - 3.2. **MB102**
    - 3.3. **HCSR04**
    - 3.4. **SSD1306**
    - 3.5. **Servo Motor SG90**
    - 3.6. **Active buzzzer**
4. **Implementación**
5. **Resultados**
6. **Conclusiones**

1. Introducción

1.1. Revisión del Estado del Arte

La evolución de la tecnología ha permitido el desarrollo de dispositivos inteligentes que facilitan la vida cotidiana. Entre estos dispositivos, las cerraduras inteligentes se destacan por su capacidad de ofrecer mayor seguridad y conveniencia comparadas con las cerraduras tradicionales. Estos dispositivos utilizan diversas tecnologías como Bluetooth, Wi-Fi, y RFID para permitir el control remoto y la monitorización en tiempo real.

Investigaciones recientes han explorado el uso de sensores ultrasónicos, microcontroladores como el ESP32, y la integración con servicios en la nube para mejorar la funcionalidad y la seguridad de las cerraduras inteligentes. El uso del protocolo MQTT y plataformas como AWS y Node-RED son comunes en estos desarrollos para la transmisión eficiente de datos y la gestión

1.2. Planteamiento del Problema

Las cerraduras tradicionales presentan varias limitaciones, como la vulnerabilidad a robos mediante métodos convencionales y la falta de monitoreo remoto. Estas limitaciones pueden poner en riesgo la seguridad de hogares y oficinas. Además, en situaciones de emergencia, la falta de un mecanismo de control remoto puede resultar en tiempos de respuesta ineficaces.

El proyecto de cerradura inteligente busca abordar estos problemas implementando un sistema que combina varios componentes tecnológicos, permitiendo el control y monitoreo remoto, y proporcionando alertas de seguridad en tiempo real.

1.3. Objetivos

Objetivo General: Desarrollar una cerradura inteligente utilizando un ESP32, sensor ultrasónico HC-SR04, micro servo motor, buzzer, y display OLED, con la capacidad de control remoto mediante MQTT y monitorización en tiempo real.

Objetivos Específicos:

- Implementar un sistema de medición de distancia para detectar la presencia cerca de la puerta.
- Integrar un buzzer para alertas de seguridad.
- Controlar la cerradura mediante un servo motor.
- Utilizar un display OLED para mostrar información relevante en tiempo real.
- Conectar el sistema a una red Wi-Fi y configurar el protocolo MQTT para el control y la monitorización remotos.

2. Marco Teórico

El desarrollo de una cerradura inteligente implica la integración de múltiples tecnologías y conceptos teóricos. El uso de sensores ultrasónicos permite medir la distancia mediante la emisión y recepción de ondas de sonido, lo cual es crucial para detectar la presencia cerca de la puerta. Los microcontroladores como el ESP32 proporcionan la capacidad de procesamiento necesario para manejar múltiples tareas simultáneamente, como la conexión a Wi-Fi, la comunicación con el broker MQTT, y el control del servo motor y el display OLED.

El protocolo MQTT (Message Queuing Telemetry Transport) es un estándar de mensajería ligero y eficiente, ideal para aplicaciones de IoT (Internet de las Cosas). Permite la transmisión de mensajes entre dispositivos de manera eficiente, lo cual es esencial para el control y monitoreo remoto de la cerradura.

3. Componentes del Sistema

3.1. ESP32

El ESP32 es un microcontrolador de bajo costo y alto rendimiento fabricado por Espressif Systems. Es una evolución del popular ESP8266, y ha ganado reconocimiento en la comunidad de desarrollo debido a su versatilidad y capacidades de conectividad. A continuación, se detallan sus características principales:

1. Arquitectura y Procesador:
    - El ESP32 está basado en una arquitectura de doble núcleo Xtensa LX6, lo que le permite ejecutar múltiples tareas de manera eficiente. Cada núcleo funciona a una velocidad de reloj de hasta 240 MHz.
    - El procesador es un Tensilica Xtensa de 32 bits.
2. Conectividad Inalámbrica:
    - Wi-Fi: El ESP32 admite los estándares 802.11b/g/n/e/i para conectividad Wi-Fi. Puede operar en la banda de 2.4 GHz y alcanzar velocidades de hasta 150 Mbit/s.
    - Bluetooth: El ESP32 incluye Bluetooth v4.2 con perfiles BR/EDR (Bluetooth clásico) y Low Energy (BLE). Esto lo hace adecuado para aplicaciones de Internet de las cosas (IoT) y proyectos que requieren comunicación inalámbrica.
3. Memoria:
    - ROM: El ESP32 tiene una memoria de solo lectura (ROM) de 448 KiB.
    - SRAM: Dispone de 520 KiB de memoria RAM para almacenamiento temporal y ejecución de programas.

- RTC Slow SRAM: Adicionalmente, cuenta con 8 KiB de memoria RAM de acceso lento para el reloj en tiempo real (RTC).
- RTC Fast SRAM: También tiene 8 KiB de memoria RAM de acceso rápido para el RTC.

4. Seguridad:
    - El ESP32 implementa estándares de seguridad como IEEE 802.11, WFA, WPA/WPA2 y WAPI para proteger las comunicaciones inalámbricas.
5. Alimentación y Voltaje:
    - El voltaje de trabajo del ESP32 es de 3.3 VDC.
    - Puede recibir energía y datos a través del conector microUSB con una entrada de 5 VDC.

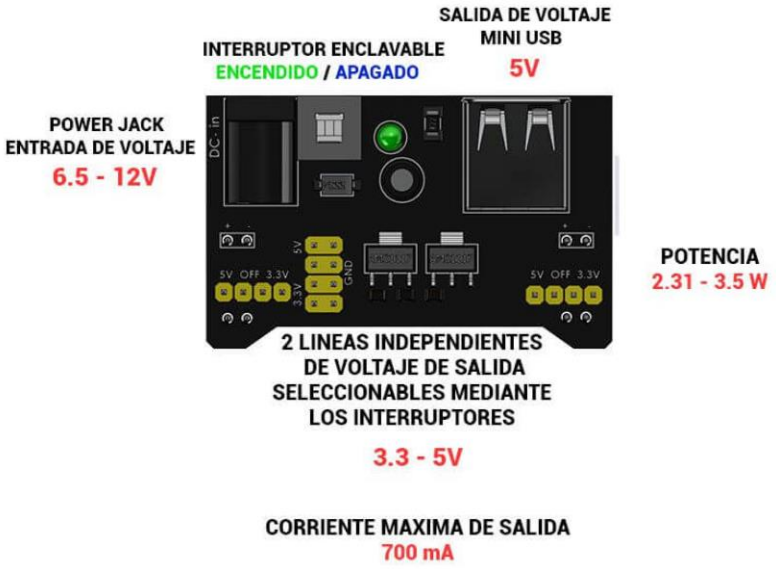
3.2. MB102

Esta Fuente de alimentación es muy útil para conectarlo a la protoboard de 400 y 830 y alimentar los circuitos electrónicos, especialmente los de carácter digital. Este modulo posee salidas de 3.3V y 5V, posee un regulador para cada voltaje, Estos voltajes se seleccionan con jumpers.

La entrada de voltaje al módulo puede ser por el conector Jack o a través del conector USB (no recomendable, la corriente se limita a la que puede entregar el puerto USB de la PC). Si se usa el conector Jack, el conector USB se puede utilizar como salida de 5V. Permite alimentar un protoboard mediante plug a un transformador de pared hasta 12v. Proporciona dos salidas independientes que son seleccionables mediante jumpers y permiten suministrar 5V o 3.3V. Cuenta además con salida de 5V por conector USB.

Características:

- Compatibilidad: Protoboard de 400 y 830
- Voltaje de entrada/alimentación: 6.5V/12V. (Jack) o fuente de alimentación USB
- Voltaje de salida (2 lí-neas independientes): 3.3V/5V (Seleccionables por Jumpers)
- Salida de tensión USB: 5V
- Máxima corriente: <700mA
- Botón: Encendido/apagado
- LED: Indicador de encendido



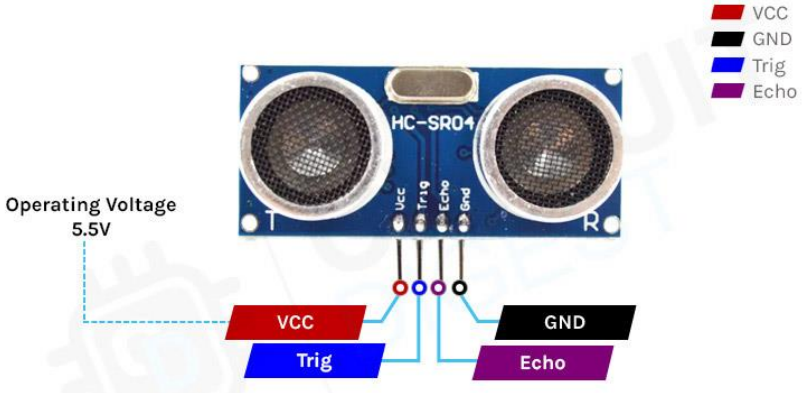
3.3. Sensor de proximidad ultrasónico HC-SR04

Éste es el sensor de alcance ultrasónico HC-SR04, proporciona de 2 a 400 cm de funcionalidad de medición sin contacto con una precisión de alcance que puede alcanzar hasta 3 mm. Cada módulo HC-SR04 incluye un transmisor ultrasónico, un receptor y un circuito de control.

En el HC-SR04 sólo hay que preocuparse por cuatro pines: VCC (Alimentación), Trig (Trigger), Echo (Receive) y GND (Ground). Usted encontrará este sensor muy fácil de configurar y utilizar para su próximo proyecto de alcance.

Características:

- Voltaje de funcionamiento: 5v DC
- Corriente de funcionamiento: 15mA
- Ángulo de medición: 15 °
- Rango de distancia: 2cm – 4m

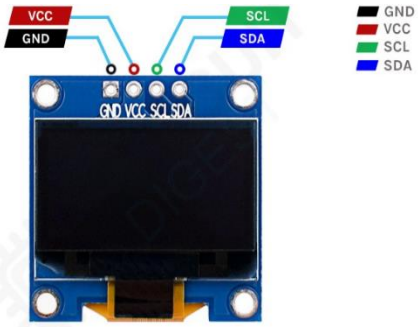


3.4. OLED SSD1306

Los Display Oled 128×64 0.96″ I2C SSD1306 son dispositivos electrónicos tipo led, permiten controlar cada píxel individualmente y mostrar tanto texto como gráficos por medio de comunicación I2C, trabajando con un voltaje de operación entre 3.3V – 5.5V DC.
Son ideales para mostrar texto, mapas de bits, píxeles, rectángulos, círculos y líneas; o en proyectos para monitoreo, equipos industriales, médicos portátiles o Smartwatch (Reloj Inteligente).

Características:

- Tipo: Display Oled
- Voltaje de Operación: 3.3V – 5V DC
- Resolución: 128×64 píxeles – 0.96″
- Monocromo: Disponible en Píxeles Blancos y Azules
- Driver: SSD1306
- Interfaz: I2C
- Pines: 4 (VCC, GND, SCL «Reloj» y SDA «Datos»)
- Dimensiones: 27mm x 27mm x 3mm
- Ultra bajo consumo de energía: 0.08W Cuando están encendidos todos los píxeles
- Temperatura de trabajo: -30ºC ~ 70ºC



3.5. Micro Servo Motor SG90 9G

El SG90 es un servo miniatura de gran calidad y diminutas dimensiones, además es bastante económico. Funciona con la mayoría de tarjetas electrónicas de control con microcontroladores y además con la mayoría de los sistemas de radio control comerciales. Funciona especialmente bien en aeronaves dadas sus características de torque, tamaño y peso.

El servo SG90 tiene un conector universal tipo “S” que encaja perfectamente en la mayoría de los receptores de radio control incluyendo los Futaba, JR, GWS, Cirrus,

Hitec y otros. Los cables en el conector están distribuidos de la siguiente forma: Rojo = Alimentación (+), Cafe = Alimentación (-) o tierra, Naranjo = Señal PWM.

Características:

- Dimensiones: 22mm x 11,5mm x 27mm
- Peso: 9 gramos
- Peso con cable y conector: 10.6 gramos
- Torque a 4.8 volts: 1.2 kg/cm
- Voltaje de operación: 4.0 a 7.2 volts
- Velocidad de giro a 4.8 volts: 120ms / 60 º
- Conector universal para la mayoría de los receptores de radio control
- Compatible con tarjetas como Arduino y microcontroladores que funcionan a 5 volts.

3.6. Active Buzzer

El buzzer activo a diferencia del pasivo solo requiere alimentación a 5v para emitir un sonido, ya que cuenta con un oscilador integrado para generar la señal de audio.

Éste buzzer es un componente con polaridad, misma que viene indicada por el largo de sus terminales donde la corta corresponde a GND y la terminal larga a VCC.

Características:

- Peso: 2g/pcs
- Voltaje de operación: 3.5V to 5.5V (se recomienda 5V)
- Corriente máxima de operación: 35mA
- Frecuencia: 2300Hz +/- 200Hz Min.
- Salida de sonido @ 10cm: 90dB
- Temperatura de operación:-30'C a 85'C
- Diámetro: 12 mm
- Altura: 9.5 mm
- Separación aproximada de pines: 8 mm

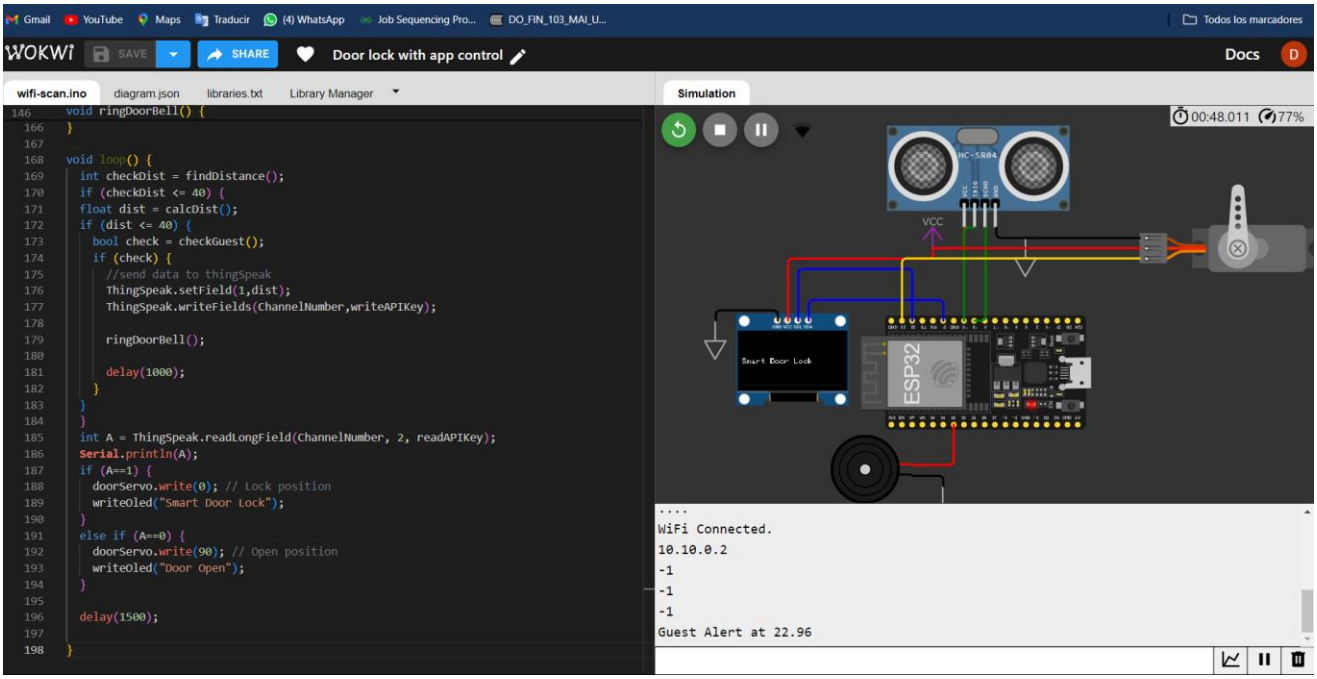
4. Implementación

Librerías Utilizadas:

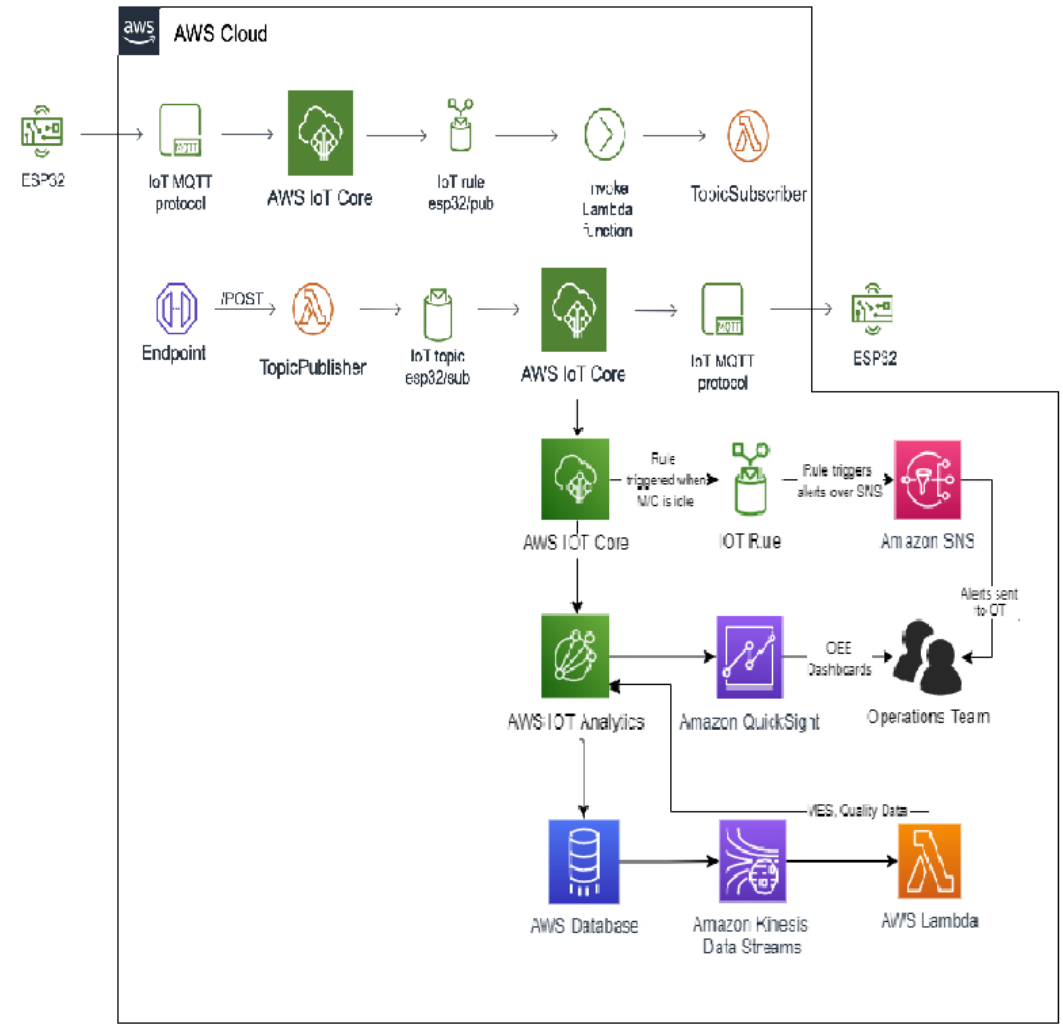
- WiFi.h: Para conectar el ESP32 a una red WiFi.
- ESP32Servo.h: Para controlar el servo motor.
- Adafruit_SSD1306.h y Adafruit_GFX.h: Para manejar la pantalla OLED.
- ThingSpeak.h y PubSubClient.h: Para enviar datos a ThingSpeak y manejar la comunicación MQTT.

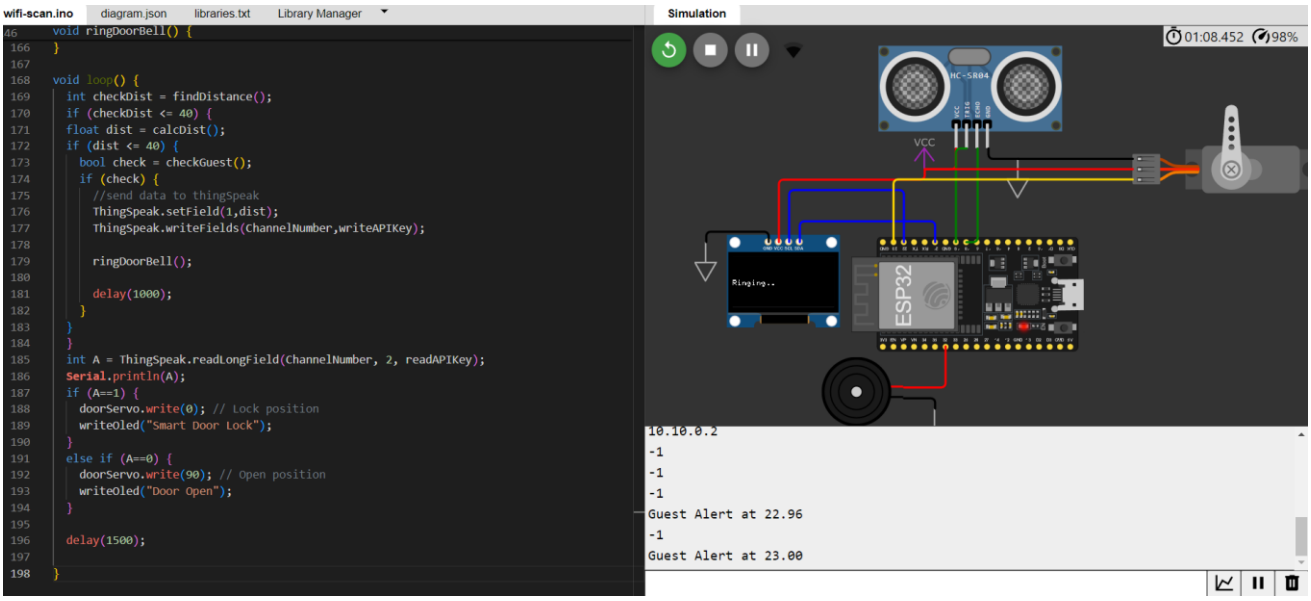
Variables Globales:

- SSID y PWD: Credenciales de la red WiFi.
- ChannelNumber, writeAPIKey, readAPIKey: Identificación y claves de acceso para el canal de ThingSpeak.
- espClient, client: Clientes para la conexión WiFi y MQTT.
- echoPin, trigPin, buzzerPin: Pines para el sensor ultrasónico y el buzzer.
- doorServo: Objeto para controlar el servo motor.
- display: Objeto para manejar la pantalla OLED.



Arquitectura del sistema






```
46  void ringDoorBell() {
166 }
167
168 void loop() {
169   int checkDist = findDistance();
170   if (checkDist <= 40) {
171   float dist = calcDist();
172   if (dist <= 40) {
173     bool check = checkGuest();
174     if (check) {
175       //send data to thingSpeak
176       ThingSpeak.setField(1,dist);
177       ThingSpeak.writeFields(ChannelNumber,writeAPIKey);
178
179       ringDoorBell();
180
181       delay(1000);
182     }
183   }
184   }
185   int A = ThingSpeak.readLongField(ChannelNumber, 2, readAPIKey);
186   Serial.println(A);
187   if (A==1) {
188     doorServo.write(0); // Lock position
189     writeOled("Smart Door Lock");
190   }
191   else if (A==0) {
192     doorServo.write(90); // Open position
193     writeOled("Door Open");
194   }
195
196   delay(1500);
197
198 }
```

Código Fuente

El código fuente se encarga de gestionar las lecturas del sensor ultrasónico, controlar el servo motor para abrir/cerrar la cerradura, activar el buzzer en modo seguro, y actualizar la pantalla OLED con información relevante. Además, el dispositivo se conecta a AWS IoT para la gestión remota de modos y la publicación de datos.

Main.ino

```
#include <Wire.h>

#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <ESP32Servo.h>
#include <WiFiClientSecure.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>
#include "WiFi.h"
#include "secrets.h"
#include <NTPClient.h>
#include <WiFiUdp.h>

// Pines del sensor ultrasónico HC-SR04
#define TRIG_PIN 19
#define ECHO_PIN 18

// Pines del servo motor
Servo servo;
#define SERVO_PIN 23

// Buzzer
#define BUZZER_PIN 32

// Display OLED
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

// Variables
long duration;
int distance;
int thresholdDistance = 5; // Umbral de distancia en cm
bool doorLocked = true;
int closedAngle = 0;   // Ajusta este valor según la calibración
int openedAngle = 90;  // Ajusta este valor según la calibración

// Modos
```

```
enum Mode { AUTOMATIC, SECURE };
Mode currentMode = AUTOMATIC;

// AWS IoT
WiFiClientSecure net;
PubSubClient client(net);

#define AWS_IOT_PUBLISH_TOPIC   "hcsr04data"
#define AWS_IOT_SUBSCRIBE_TOPIC "hcsr04/mode"

// NTP Client to get time
WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP, "pool.ntp.org", 0, 60000); // Actualiza cada
minuto

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

  // Inicializar pines del sensor ultrasónico
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);

  // Inicializar el buzzer
  pinMode(BUZZER_PIN, OUTPUT);

  // Inicializar el servo
  servo.attach(SERVO_PIN);
  lockDoor(); // Cerradura cerrada

  // Inicializar la pantalla OLED
  Wire.begin();
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println(F("SSD1306 allocation failed"));
    for (;;);
  }
  display.display();
  delay(2000);
  display.clearDisplay();

  // Conectar a AWS IoT
  connectAWS();

  // Inicializar NTP
  timeClient.begin();
}
```

```
void loop() {
  // Actualizar la hora
  timeClient.update();

  // Mantener la conexión con AWS IoT
  if (!client.connected()) {
    reconnectAWS();
  }
  client.loop();

  // Medir distancia
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);

  duration = pulseIn(ECHO_PIN, HIGH);
  distance = duration * 0.034 / 2;

  // Depuración: Imprimir distancia en el serial monitor
  Serial.print("Distance: ");
  Serial.println(distance);

  // Actualizar display
  updateDisplay();

  // Verificar si la distancia es menor al umbral
  if (distance <= thresholdDistance) {
    if (currentMode == AUTOMATIC) {
      servo.attach(SERVO_PIN);
      unlockDoor();
      delay(3000); // Esperar 3 segundos antes de cerrar la puerta
      lockDoor();
    } else if (currentMode == SECURE) {
      servo.detach();
      tone(BUZZER_PIN, 2500, 500);  // Emitir tono de 2500Hz durante 0.5
segundos
    }
  } else {
    noTone(BUZZER_PIN);
  }

  // Publicar datos en AWS IoT
```

```
  publishMessage();

  delay(1000); // Retraso de 1 segundo entre lecturas
}

void lockDoor() {
  if (currentMode == AUTOMATIC) {
    servo.write(closedAngle); // Posición de cerradura cerrada
  }
  doorLocked = true;
  Serial.println("Door locked");
}

void unlockDoor() {
  if (currentMode == AUTOMATIC) {
    servo.write(openedAngle); // Posición de cerradura abierta
  }
  doorLocked = false;
  Serial.println("Door unlocked");
}

void updateDisplay() {
  display.clearDisplay();

  // Header
  display.setTextSize(2);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(0, 0);
  display.print("Door Lock");

  // Separator line
  display.drawLine(0, 20, SCREEN_WIDTH, 20, SSD1306_WHITE);

  // Distance
  display.setTextSize(1);
  display.setCursor(0, 25);
  display.print("Distance: ");
  display.setTextSize(2);
  display.setCursor(70, 25);
  display.print(distance);
  display.print(" cm");

  // Mode
  display.setTextSize(1);
  display.setCursor(0, 50);
```

```
  display.print("Mode: ");
  display.setTextSize(2);
  display.setCursor(70, 50);
  display.print(currentMode == AUTOMATIC ? "Auto" : "Secure");

  display.display();
}


void connectAWS() {
  WiFi.mode(WIFI_STA);
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);

  Serial.println("Connecting to Wi-Fi");

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  // Configure WiFiClientSecure to use the AWS IoT device credentials
  net.setCACert(AWS_CERT_CA);
  net.setCertificate(AWS_CERT_CRT);
  net.setPrivateKey(AWS_CERT_PRIVATE);

  // Connect to the MQTT broker on the AWS endpoint we defined earlier
  client.setServer(AWS_IOT_ENDPOINT, 8883);
  client.setCallback(messageReceived);

  Serial.println("Connecting to AWS IoT");

  while (!client.connect(THINGNAME)) {
    Serial.print(".");
    delay(100);
  }

  if (!client.connected()) {
    Serial.println("AWS IoT Timeout!");
    return;
  }

  Serial.println("AWS IoT Connected!");
  client.subscribe(AWS_IOT_SUBSCRIBE_TOPIC);
}
```

```
void reconnectAWS() {
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    if (client.connect(THINGNAME)) {
      Serial.println("connected");
      client.subscribe(AWS_IOT_SUBSCRIBE_TOPIC);
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      delay(5000);
    }
  }
}

void publishMessage() {
  StaticJsonDocument<200> doc;
  doc["distance"] = distance;
  doc["doorLocked"] = doorLocked;
  doc["timestamp"] = timeClient.getEpochTime(); // Añadir timestamp
  doc["mode"] = currentMode == AUTOMATIC ? "Automatic" : "Secure";
  char jsonBuffer[512];
  serializeJson(doc, jsonBuffer);

  client.publish(AWS_IOT_PUBLISH_TOPIC, jsonBuffer);
}

void messageReceived(char* topic, byte* payload, unsigned int length) {
  char message[length + 1];
  memcpy(message, payload, length);
  message[length] = '\0';

  StaticJsonDocument<200> doc;
  deserializeJson(doc, message);
  const char* mode = doc["mode"];

  if (strcmp(mode, "Automatic") == 0) {
    currentMode = AUTOMATIC;
  } else if (strcmp(mode, "Secure") == 0) {
    currentMode = SECURE;
  }

  Serial.print("Mode changed to: ");
  Serial.println(mode);
}
```

secrets.h

```
#include <pgmspace.h>

#define SECRET
#define THINGNAME "ESP32_HCSR04"

const char WIFI_SSID[] = "iPhone de Daniel";
const char WIFI_PASSWORD[] = "12345678";
const char AWS_IOT_ENDPOINT[] = "a3uo8wcrn4x1hp-ats.iot.us-east-
1.amazonaws.com";

// Amazon Root CA 1
static const char AWS_CERT_CA[] PROGMEM = R"EOF(
-----BEGIN CERTIFICATE-----
MIIDQTCCAimgAwIBAgITBmyfz5m/jAo54vB4ikPmljZbyjANBgkqhkiG9w0BAQsF
ADA5MQswCQYDVQQGEwJVUzEPMA0GA1UEChMGQW1hem9uMRkwFwYDVQQDExBBbWF6
b24gUm9vdCBDQSAxMB4XDTE1MDUyNjAwMDAwMFoXDTM4MDExNzAwMDAwMFowOTEL
MAkGA1UEBhMCVVMxDzANBgNVBAoTBkFtYXpvbjEZMBcGA1UEAxMQQW1hem9uIFJv
b3QgQ0EgMTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBALJ4gHHKeNXj
ca9HgFB0fW7Y14h29Jlo91ghYPl0hAEvrAIthtOgQ3pOsqTQNroBvo3bSMgHFzZM
9O6II8c+6zf1tRn4SWiw3te5djgdYZ6k/oI2peVKVuRF4fn9tBb6dNqcmzU5L/qw
IFAGbHrQgLKm+a/sRxmPUDgH3KKHOVj4utWp+UhnMJbulHheb4mjUcAwhmahRWa6
VOujw5H5SNz/0egwLX0tdHA114gk957EWW67c4cX8jJGKLhD+rcdqsq08p8kDi1L
93FcXmn/6pUCyziKrlA4b9v7LWIbxcceVOF34GfID5yHI9Y/QCB/IIDEgEw+OyQm
jgSubJrIqg0CAwEAAaNCMEAwDwYDVR0TAQH/BAUwAwEB/zAOBgNVHQ8BAf8EBAMC
AYYwHQYDVR0OBBYEFIQYzIU07LwMlJQuCFmcx7IQTgoIMA0GCSqGSIb3DQEBCwUA
A4IBAQCY8jdaQZChGsV2USggNiMOruYou6r4lK5IpDB/G/wkjUu0yKGX9rbxenDI
U5PMCCjjmCXPI6T53iHTfIUJrU6adTrCC2qJeHZERxhlbI1Bjjt/msv0tadQ1wUs
N+gDS63pYaACbvXy8MWy7Vu33PqUXHeeE6V/Uq2V8viTO96LXFvKWlJbYK8U90vv
o/ufQJVtMVT8QtPHRh8jrdkPSHCa2XV4cdFyQzR1bldZwgJcJmApzyMZFo6IQ6XU
5MsI+yMRQ+hDKXJioaldXgjUkK642M4UwtBV8ob2xJNDd2ZhwLnoQdeXeGADbkpy
rqXRfboQnoZsG4q5WTP468SQvvG5
-----END CERTIFICATE-----
)EOF";

// Device Certificate
static const char AWS_CERT_CRT[] PROGMEM = R"KEY(
-----BEGIN CERTIFICATE-----
MIIDWjCCAkKgAwIBAgIVAIOkFOAHbi90jlopqBr3YH3ZpVTXMA0GCSqGSIb3DQEB
CwUAME0xSzBJBgNVBAsMQkFtYXpvbiBXZWIgU2VydmljZXMgTz1BbWF6b24uY29t
IEluYy4gTD1TZWF0dGxlIFNUPVdhc2hpbmd0b24gQz1VUzAeFw0yNDA3MTQwMzEy
MjdaFw00OTEyMzEyMzU5NTlaMB4xHDAaBgNVBAMME0FXUyBJb1QgQ2VydGlmaWNh
dGUwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCYcavIN/qvi+4U52Tg
```

```
Z3mvg5NhmS/5qHrvx12G/rASlICv3541S0uDIpK7fQa85pDneB1urRGE90OyotWc
ZWa7Q2eBM4Dz7VrKulQNasD1EDXgVtpfDkINGUHjOq4EKE8y5ToVnKTg9LDK79Sr
SlNvkNKRoc7WTZ3SEtsV4ENugK4tG5D9TugeFIAzM3YBU91UOUdQfbslOGfBtepK
qK1jWFWQpSEI4rwbfe2+xafvAizl6ob9EXfctatI6Q8xJ/ZutrxZN2xrb1Cwk4ew
promj8RT0YrzWNsZwEaH0Rqrt0z/sGWa4xN59ZJYbV4VS5JF+Ivc2QJ/VNvSddmE
dogvAgMBAAGjYDBeMB8GA1UdIwQYMBaAFKJaLKmgwU0mpSSJEn8l2iaUm6HnMB0G
A1UdDgQWBBSoHOJX7bdXr2qkYjVvI+SR3ef2ozAMBgNVHRMBAf8EAjAAMA4GA1Ud
DwEB/wQEAwIHgDANBgkqhkiG9w0BAQsFAAOCAQEASr4GCHx+whYHgKtA33ieqisP
87tdfAF3+aaX5dg/WAbabmG9ndbjKyea2wWeHNccWBzZoaCT/m6XHYXNJlYzzBhK
OB/hzhZW6I0ZhXc+gFqyo5xRn1QbrMYZas8g4ZsUTdUNBWEgT86PxqFKVHDD7Q9c
VU+I32gdu5KIY1raej0a/j/yRWnFzhedzz5ih9LLfWAWsZbC4hHz8qcYMRnsmMRk
UbkdrLbS4RjeYO0dLqVOmBk4ujsVGWdyeHySk5a5t7cHedEiphAW++W28IQvQVD8
es9QaT5bDQM8wHAokj6Vgt9tKddQlo1gNyNsMKaHI42CBcKVRxzxWSKhk11mvw==
-----END CERTIFICATE-----
)KEY";

// Device Private Key
static const char AWS_CERT_PRIVATE[] PROGMEM = R"KEY(
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAmHGryDf6r4vuFOdk4Gd5r4OTYZkv+ah678ddhv6wEpSAr9+e
NUtLgyKSu30GvOaQ53gdbq0RhPdDsqLVnGVmu0NngTOA8+1ayrpUDWrA9RA14Fba
Xw5CDRlB4zquBChPMuU6FZyk4PSwyu/Uq0pTb5DSkaHO1k2d0hLbFeBDboCuLRuQ
/U7oHhSAMzN2AVPdVDlHUH27JThnwbXqSqitY1hVkKUhCOK8G33tvsWn7wIs5eqG
/RF33LWrSOkPMSf2bra8WTdsa29QsJOHsKa6Jo/EU9GK81jbGcBGh9Eaq7dM/7Bl
muMTefWSWG1eFUuSRfiL3NkCf1Tb0nXZhHaILwIDAQABAoIBAF09CShItNcx6sh2
TsjCATdMbpQ/MYytc6O8eIPcuqxij+MCDq3iA6mNN9ncuoZHL1GjsbzDtfdxlMJp
veUOUCsxKtlnOvP2tJVSKZ6bMLGy3ID+HkoNWnDQhfAJL+3zCZ3DfidBy8abYpZ8
4cJyj12pYmEYan3As0qNpdpXdGbc8/1XLxhEaWge7vbt0LwqGmtQfWTvQLWL/HwU
PquBVQ7M9ebgS/gJxK9Vy3oWru+6E1o0v+OGpP43nkYJfbyNd9nNhLMVi2T+lUlK
zpfgA89N/7w2W3cNWm/UIZ1+x5dsghvBa3MKdjcjC3eVZxpSl08nw82+hgIWlxqJ
/U9V8YECgYEAyFnYyFQ284JkDtv6uutywmGzdyNDi7XDfXWW5aUlNFI0aoW+91lD
8MMxu9LYeFz0vHYk1L52qydaGpnjtLfiS1PP542i759DtLLuBy0eebmub/paXDbv
FjsaIhC14mjDT6LBFvqzMXNZNzgJbGpKRkDdoVSxzs+3T68fQNqoR5ECgYEAwsla
qrlEf2Xpd9FNegGaWdVbEdJa0rJvTr7Evs3tiavWDGGxCKcGjxl9Jj9l79evdcYk
0UgXfPJ8dzqbkUHxnkkUq8OR1yOGt/cA+/ZSJYD5PzvrrQ2/XYb1H18DevLw9ydQ
5jcAIPY6PchTmk0chNlzzG25LOMCx63rlWkTc78CgYBCXOBdmueIyUvzUBJBD/Q3
Rr13XQrmoBJc4DJXZthn+EK8dgjHTZ4fhchXeoORetjmBAw3G8+a6bIC1a6DEdW3
Nc/L1zJ9RLfchtgpMFzD3SlWlgqQOPIA3xrM5dQZZrIrErH6tpfcbWTD0Z/tdVec
5aeD5NXwtwT1NACaLtNGwQKBgQC/aD3AJe45glu5ceNAUPLr88xZ5ut4v/7A0pVj
KdiRwFnGMPlWF0tKszFXtFTnKnswsFTdkooCLgZA40abpsFXWVkGsFdPoYlwnThK
X/K5B9PlX2xn0WqK7WNSQHubBb/kHq5QixuFSHBS5dfBeKTdpxEzgV7FzQ2eSjmF
yItk4QKBgGUr2lgIaORxR9dbVXhC9+Xyc3+k3WebZUb70Esf68foVYvbk/F8MuUK
2qDCVunaqOfTJaQqfMOHNBHC+sDcJ5SxC+u7noWCR/mGmiQ04Rv4XZrZeqC8+zuV
Indut1SC2hAXErDXv6ecNXE+NdC7AwK9tW0eVMKigWbrIfbFV5qY
-----END RSA PRIVATE KEY-----
```

```
)KEY";
```

5. Resultados

Durante la fase de pruebas, se realizaron varias mediciones y observaciones sobre el funcionamiento del sistema:

1. **Detección de Proximidad:** El sensor ultrasónico HC-SR04 se calibró para detectar distancias con precisión. Se estableció un umbral de 5 cm para activar las acciones del sistema.
2. **Control del Servo Motor:** El servo motor SG90 respondió adecuadamente a los comandos de apertura y cierre, con los ángulos configurados en 0° para cerrado y 90° para abierto. La cerradura se mantuvo en estado bloqueado a menos que se detectara proximidad en el modo automático.
3. **Alarma de Seguridad:** En el modo seguro, el buzzer se activó correctamente cuando la distancia medida fue menor al umbral, emitiendo un tono de 2500Hz por 0.5 segundos.
4. **Visualización de Datos:** La pantalla OLED mostró de manera clara y legible la distancia medida, el modo actual del sistema (Automático o Seguro), y el estado de la cerradura.
5. **Conectividad AWS IoT:** El ESP32 se conectó exitosamente a AWS IoT, permitiendo el cambio de modos y la publicación de datos de distancia y estado de la cerradura en tiempo real. Los mensajes de control recibidos desde AWS IoT fueron procesados adecuadamente para cambiar entre los modos Automático y Seguro.

6. Conclusiones

El proyecto de la cerradura inteligente demostró ser efectivo en términos de funcionalidad y conectividad. Los objetivos planteados inicialmente se lograron, y se pueden destacar las siguientes conclusiones:

1. **Precisión y Fiabilidad:** El uso del sensor ultrasónico HC-SR04 proporcionó mediciones precisas, permitiendo una detección confiable de proximidad. La calibración adecuada del servo motor SG90 aseguró el correcto funcionamiento de la cerradura.
2. **Seguridad Mejorada:** La implementación del modo seguro, que activa una alarma sonora mediante el buzzer, añadió una capa de seguridad adicional, disuadiendo intentos de intrusión.
3. **Integración con AWS IoT:** La conectividad con AWS IoT facilitó el control remoto del sistema y la supervisión en tiempo real, demostrando la viabilidad de integrar soluciones IoT para mejorar la funcionalidad y seguridad de sistemas físicos.
4. **Interfaz de Usuario:** La pantalla OLED resultó ser una herramienta valiosa para proporcionar retroalimentación visual instantánea sobre el estado del sistema, mejorando la experiencia del usuario.
5. **Escalabilidad:** El diseño modular del sistema permite futuras expansiones, como la integración de otros sensores o la implementación de nuevas funcionalidades de seguridad.

El proyecto no solo alcanzó sus objetivos técnicos, sino que también proporcionó una plataforma robusta y escalable para futuras mejoras y aplicaciones en el ámbito de las cerraduras inteligentes y la seguridad del hogar.