Rapport de Projet : Système Domotique Intelligent avec détection d'incendie

Introduction

Ce projet vise à développer un système domotique intelligent basé sur l'ESP32, permettant le contrôle de dispositifs via MQTT et la détection d'incendie avec un capteur DHT22. Lors de la détection d'un incendie, une série d'actions automatiques se déclenche, notamment l'activation d'un buzzer, l'ouverture d'une porte (contrôlée par un servomoteur) et l'affichage d'alertes sur un écran OLED.

Matériels et Composants

1. **ESP32** : Module principal pour le contrôle et la connectivité.
2. **Capteur DHT22** : Capteur de température et d'humidité.
3. **Servomoteur** : Pour ouvrir et fermer la porte.
4. **OLED SSD1306** : Affichage des données et des alertes.
5. **LEDs (x2)** : au lieu des Lampes
6. **Boutons-poussoirs (x2)** : Contrôle manuel des LEDs.
7. **Buzzer** : Signal sonore pour alerte incendie.
8. **Résistances** : Pour les LEDs.
9. **Connexion Wi-Fi** : Communication avec le broker MQTT.

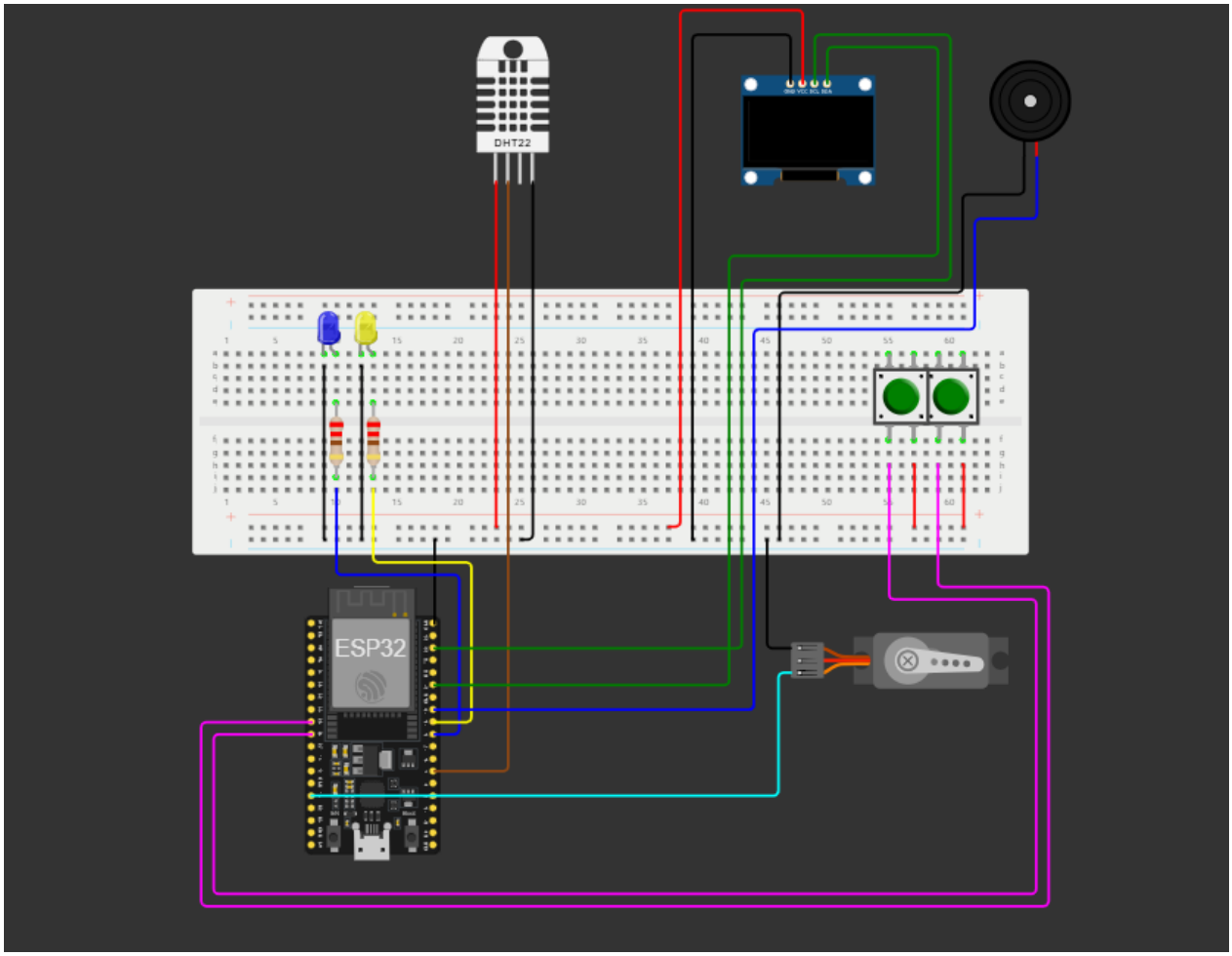
Fonctionnalités Principales

1. **Contrôle à distance via MQTT** :
    - Allumage/extinction des LEDs.
    - Ouverture/fermeture de la porte (servomoteur).
2. **Détection d'incendie** :
    - Analyse des données du capteur DHT22.
    - Activation du buzzer et du servomoteur si la température dépasse 60°C.
    - Affichage d'alertes sur l'écran OLED.
3. **Affichage des données** :
    - Température et humidité.
    - État des LEDs.
    - Messages d'alerte en cas de détection d'incendie.
4. **Contrôle manuel** :
    - Utilisation de deux boutons pour contrôler les LEDs.

Schéma de Connexion

Les connexions principales incluent :

- **DHT22** : GPIO4 (Données), VCC et GND.
- **OLED SSD1306** : GPIO22 (SCL) et GPIO21 (SDA).
- **Servomoteur** : GPIO13 (PWM), VCC et GND.
- **Boutons** : GPIO26 et GPIO25.
- **LEDs** : GPIO5 et GPIO18.
- **Buzzer** : GPIO19.



Implémentation Logicielle

Configuration du Wi-Fi

La fonction **connect_wifi** établit une connexion à un réseau Wi-Fi prédéfini, permettant à l'ESP32 de communiquer avec le **broker MQTT.**

Contrôle MQTT

Le client MQTT est configuré pour recevoir et traiter les messages sur trois sujets :

- **home/led1 :** Contrôle de la LED1.
- **home/led2 :** Contrôle de la LED2.
- **home/servo :** Contrôle du servomoteur.

Gestion des Boutons

Deux boutons permettent de contrôler manuellement les LEDs avec une gestion anti-rebond grâce à un délai de 200 ms.

Détection d'incendie

La fonction read_dht22 mesure la température et l'humidité. Si la température dépasse 60°C :

1. Le buzzer s'active.
2. Le servomoteur déverrouille la porte .
3. L'écran OLED affiche des alertes (« FIRE DETECTED! » et « Door Unlocked »).

Lorsque la température revient à la normale, le système :

1. Désactive le buzzer.
2. Referme la porte .

Mise à Jour OLED

La fonction **update_oled** met à jour l'affichage avec :

- Les états des LEDs.
- Les données de température et d'humidité.
- Les messages d'alerte en cas d'incendie.

Code

main.py ● diagram.json ● ssd1306.py lcd_api.py i2c_lcd.py

```python
import network
import time
from umqtt.simple import MQTTClient
from machine import Pin, PWM, SoftI2C
import utime
from dht import DHT22
import ssd1306

SSID = "Wokwi-GUEST"
PASSWORD = ""

MQTT_BROKER = "mqtt-dashboard.com"
CLIENT_ID = "ahmedchFise20242"
TOPIC_LED1 = "home/led1"
TOPIC_LED2 = "home/led2"
TOPIC_SERVO = "home/servo"

led1 = Pin(5, Pin.OUT)
led2 = Pin(18, Pin.OUT)

i2c = SoftI2C(scl=Pin(22), sda=Pin(21), freq=400000)
oled_width = 128
oled_height = 64
oled = ssd1306.SSD1306_I2C(oled_width, oled_height, i2c)

btn1 = Pin(26, Pin.IN)
btn2 = Pin(25, Pin.IN)

servo = PWM(Pin(13), freq=50)

dht_sensor = DHT22(Pin(4))

buzzer = Pin(19, Pin.OUT)

def set_servo_angle(angle):
    duty = int((angle / 180) * 102 + 26)
    servo.duty(duty)

def connect_wifi():
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    wlan.connect(SSID, PASSWORD)
```

```python
def connect_wifi():
    wlan.connect(SSID, PASSWORD)
    while not wlan.isconnected():
        print("Connecting to Wi-Fi...")
        time.sleep(1)
    print("Connected to Wi-Fi! IP Address:", wlan.ifconfig()[0])

def mqtt_callback(topic, msg):
    print(f"Received message on topic {topic}: {msg}")
    if topic == TOPIC_LED1.encode() and msg == b"ON":
        led1.value(1)
    elif topic == TOPIC_LED1.encode() and msg == b"OFF":
        led1.value(0)
    elif topic == TOPIC_LED2.encode() and msg == b"ON":
        led2.value(1)
    elif topic == TOPIC_LED2.encode() and msg == b"OFF":
        led2.value(0)
    elif topic == TOPIC_SERVO.encode() and msg == b"OPEN":
        set_servo_angle(40)
        oled.fill(0)
        oled.text("Door Unlocked", 0, 0)
        oled.show()
        time.sleep(4)
        set_servo_angle(90)
        oled.fill(0)
        oled.text("Door Locked", 0, 0)
        oled.show()

def connect_mqtt():
    client = MQTTClient(CLIENT_ID, MQTT_BROKER)
    client.set_callback(mqtt_callback)
    client.connect()
    print(f"Connected to MQTT Broker: {MQTT_BROKER}")
    client.subscribe(TOPIC_LED1)
    client.subscribe(TOPIC_LED2)
    client.subscribe(TOPIC_SERVO)
    return client

last_triggered = {btn1: 0, btn2: 0}
DEBOUNCE_TIME_MS = 200
```

```
def button_handler(pin):

    if utime.ticks_diff(current_time, last_triggered[pin]) > DEBOUNCE_TIME_MS:
        last_triggered[pin] = current_time

        if pin == btn1:
            led1.value(not led1.value())
            if led1.value() == 1:
                print("LED1 set manually to ON")
            else:
                print("LED1 set manually to OFF")
        elif pin == btn2:
            led2.value(not led2.value())
            if led2.value() == 1:
                print("LED2 set manually to ON")
            else:
                print("LED2 set manually to OFF")

btn1.irq(trigger=Pin.IRQ_FALLING, handler=button_handler)
btn2.irq(trigger=Pin.IRQ_FALLING, handler=button_handler)

def update_oled():
    global temp
    oled.fill(0)
    oled.text("LED1: ON" if led1.value() == 1 else "LED1: OFF", 0, 0)
    oled.text("LED2: ON" if led2.value() == 1 else "LED2: OFF", 0, 10)
    temp = dht_sensor.temperature()
    oled.text("Temp: {:.1f}C".format(temp), 0, 20)
    hum = dht_sensor.humidity()
    oled.text("Hum: {:.1f}%".format(hum), 0, 30)
    oled.show()

def read_dht22():
    try:
        dht_sensor.measure()
        update_oled()
        print("Temperature: {:.1f}C, Humidity: {:.1f}%".format(dht_sensor.temperature(), dht_sensor.humidity()))
        if temp >= 60:
            print("Fire detected! Activating buzzer...")
            oled.fill(0)
            oled.text("FIRE DETECTED!", 0, 0)
```

```
def read_dht22():
    try:
        dht_sensor.measure()
        update_oled()
        print("Temperature: {:.1f}C, Humidity: {:.1f}%".format(dht_sensor.temperature(), dht_sensor.humidity()))
        if temp >= 60:
            print("Fire detected! Activating buzzer...")
            oled.fill(0)
            oled.text("FIRE DETECTED!", 0, 0)

            buzzer.value(1)
            set_servo_angle(40)
            oled.text("Door Unlocked", 0, 10)
            oled.show()
        else:
            buzzer.value(0)
            set_servo_angle(90)
    except OSError as e:
        print("Failed to read sensor.")

connect_wifi()
mqtt_client = connect_mqtt()
try:
    while True:
        mqtt_client.check_msg()
        read_dht22()
        time.sleep(2)
except KeyboardInterrupt:
    print("Disconnecting...")
    mqtt_client.disconnect()
```

Tests et Validation

1. **Connexion Wi-Fi** : Vérifiée via l'adresse IP attribuée.
2. **Contrôle MQTT** : Testé avec des commandes ON/OFF pour les LEDs et OPEN pour la porte (servomoteur).
3. **Détection d'incendie** : Température simulée au-dessus de 60°C pour activer les actions prévues.
4. **Affichage OLED** : Confirmé que toutes les données et alertes s'affichent correctement.

Conclusion

Ce système domotique intègre des capteurs, un contrôle manuel et un contrôle à distance via MQTT, tout en assurant une sécurité supplémentaire grâce à la détection d'incendie. Les fonctionnalités ajoutées, notamment l'alerte incendie et la gestion du servomoteur, renforcent la polyvalence du projet et son utilité dans des applications réelles.