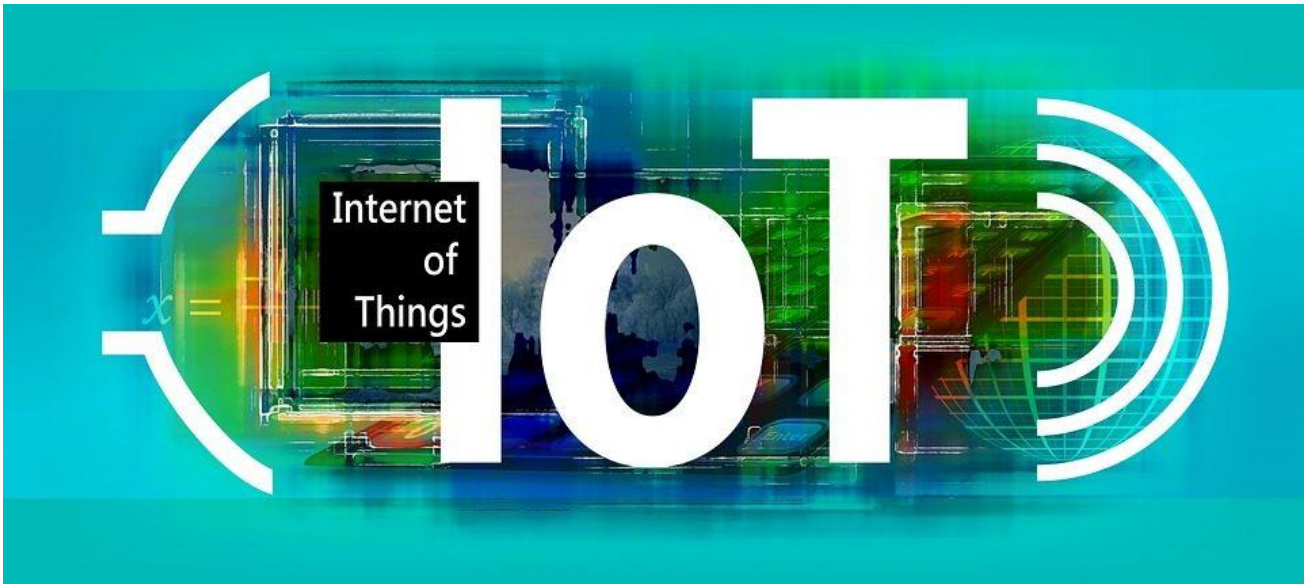


Control de Riego Automático con ESP32

Marini Alan

DNI: 35785659

email: aland.marini@gmail.com

Asignatura y profesor: AULA 2

Aproximación al Mundo del Trabajo - TSDS - 2024 - Mainero Alejandro Luis

Índice

1. Introducción

1. **Objetivos**: Desarrollar un sistema de riego automático que monitorice la humedad del suelo y active una bomba de riego cuando sea necesario, utilizando un ESP32 y un sensor de humedad.
2. **Justificación**: La gestión eficiente del agua es crucial en la agricultura, y este sistema busca optimizar el riego, minimizando el desperdicio de recursos.

2. Descripción del Proyecto

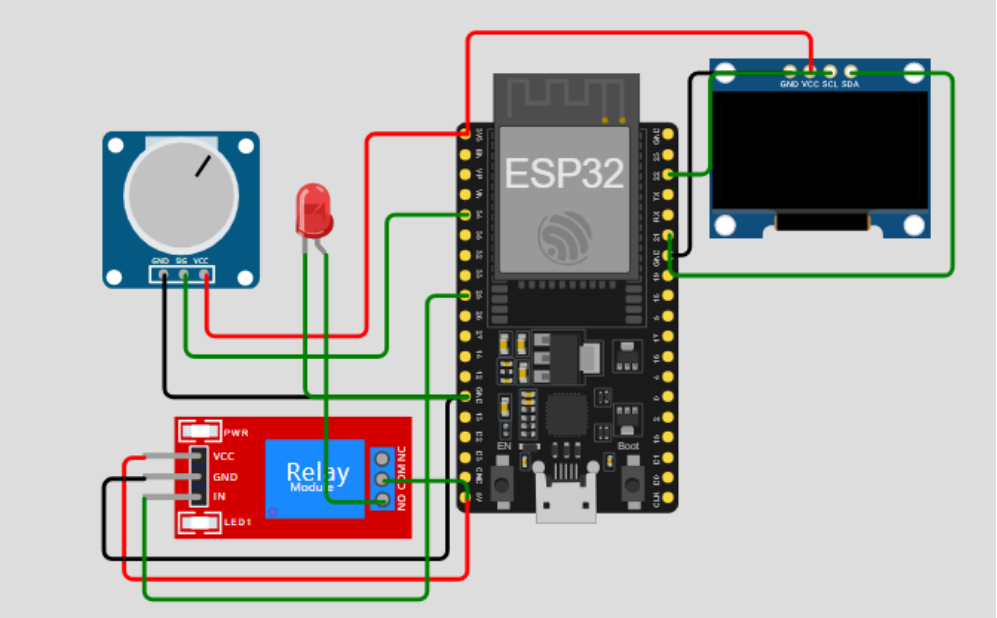
2.1. Hardware Utilizado

- **ESP32**: Microcontrolador que permite conectividad Wi-Fi y control de dispositivos.
- **Sensor de Humedad**: Dispositivo que mide el nivel de humedad en el suelo, utilizado para decidir cuándo activar la bomba.
- **Relé**: Componente que permite encender y apagar la bomba de riego.
- **Pantalla OLED**: Muestra información sobre el estado del sistema, como el nivel de humedad y el estado de la bomba.

2.2. Funcionalidad

El sistema está diseñado para leer los niveles de humedad del suelo en intervalos regulares. Cuando la humedad es inferior a un umbral predefinido, activa un relé que enciende la bomba de riego. También envía la información de humedad a una API para su gestión y análisis.

3. Esquema de Conexión



4. Código Fuente

4.1. Estructura del Código

```
# Importar bibliotecas

from machine import Pin, ADC, I2C

from ssd1306 import SSD1306_I2C

import urequests

import network

import time

import ujson


# ==================  SECCION SIMULACION WIFI
==================================


# Parámetros de WiFi

SSID = "Wokwi-GUEST"

PASSWORD = ""


# Conexión a Wifi

def connect_wifi():

    wlan = network.WLAN(network.STA_IF)

    wlan.active(True)

    wlan.connect(SSID, PASSWORD)


    print('Conectando a la red...')

    while not wlan.isconnected():

        time.sleep(1)
```

```
    print('Conectado a la red:', wlan.ifconfig())


# Iniciamos ESP32 con conexión a Wi-Fi

connect_wifi()


# ================================================================
==============


# ============= Función para enviar un POST a la API
============================


def send_post_request(humedad):

    url = "https://control-de-riego-automatico.onrender.com/api/riego/data"

    data = {'humedad': humedad}

    json_data = ujson.dumps(data)


    # Enviar la solicitud POST

    try:

        print("Enviando solicitud POST a la API...")

        response = urequests.post(url, data=json_data, headers={"Content-Type":
"application/json"})

        print("Código de respuesta:", response.status_code)

        print("Respuesta del servidor:", response.text)

        response.close()

    except Exception as e:

        print("Error al enviar la solicitud:", e)
```

```
#
================================================================
==============


# Configuración del display OLED

i2c = I2C(0, scl=Pin(22), sda=Pin(21))  # Pines I2C del ESP32

oled = SSD1306_I2C(128, 64, i2c)        # Configuración de la pantalla OLED
(128x64)


# Configuración de pines

sensorHumedadPin = ADC(Pin(34))  # Pin analógico para el sensor de humedad
(simulado con potenciómetro)

sensorHumedadPin.atten(ADC.ATTN_11DB)  # Ajuste del rango de lectura (0-3.3V)

relePin = Pin(25, Pin.OUT)  # Pin digital para controlar el relé (y el LED)


# Activación humedad al 40% (aprox.)

UMBRAL_HUMEDAD = 1638  # Potencia máxima del ADC es 4095, esto es aprox.
40% del rango.


#
================================================================
==============


def leer_humedad():

    # Leer valor analógico del sensor de humedad

    return sensorHumedadPin.read()


def mostrar_texto(texto1, texto2=""):
```

```
    # Limpiar la pantalla y mostrar dos líneas de texto
    oled.fill(0)  # Limpia la pantalla
    oled.text(texto1, 0, 0)  # Primera línea
    oled.text(texto2, 0, 10)  # Segunda línea (si es necesario)
    oled.show()

def main():
    while True:
        # Leer el valor del sensor de humedad
        humedad = leer_humedad()
        print("Humedad del suelo (valor ADC):", humedad)

        # Controlar la bomba según el valor de humedad
        if humedad < UMBRAL_HUMEDAD:
            relePin.value(1)  # Encender bomba (activar relé)
            print("Bomba activada")
            mostrar_texto("NECESITO AGUA", "Bomba activada")
        else:
            relePin.value(0)  # Apagar bomba (desactivar relé)
            print("Bomba desactivada")
            mostrar_texto("TODO BIEN POR AQUI", "Bomba desactivada")

        # Enviar la lectura de humedad a la API
        send_post_request(humedad)
```

```
    time.sleep(10)  # Esperar 10 segundos antes de la siguiente lectura


# Ejecutar el bucle principal

main()
```

4.2. Explicación del Código

- Conexión Wi-Fi: La función connect_wifi() establece la conexión del ESP32 a la red Wi-Fi utilizando las credenciales proporcionadas. Se activa la interfaz de red y se intenta conectar. Se imprime un mensaje en la consola una vez que la conexión es exitosa.

- Lectura de Humedad: La función leer_humedad() lee el valor del sensor de humedad, que se conecta al pin analógico definido. Este valor se utiliza para determinar si la bomba debe ser activada o desactivada.

- Control del Relé: Dentro del bucle principal (main()), se comprueba si el valor de humedad es menor que un umbral predefinido (UMBRAL_HUMEDAD). Si es así, se activa el relé para encender la bomba, y se actualiza la pantalla OLED para indicar que la bomba está activa. De lo contrario, se desactiva el relé y se actualiza la pantalla para reflejar que no se requiere riego.

- Envío de Datos a la API: La función send_post_request(humedad) se encarga de enviar el valor de humedad a una API a través de una solicitud POST. Se utiliza urequests para realizar la solicitud y se imprimen los códigos de respuesta y la respuesta del servidor en la consola.

5. Resultados y Pruebas

5.1. Lecturas de Humedad

Durante el desarrollo del proyecto, se realizaron múltiples lecturas de humedad simulada del suelo en diferentes condiciones para evaluar el rendimiento del sistema.

- **Condiciones de prueba**: Se utilizó un potenciómetro para simular diversas condiciones de humedad, replicando escenarios de sequedad y humedad.
- **Resultados obtenidos**:
    - En configuraciones que simulaban un suelo seco, se registraron valores de humedad entre 0 y 800.
    - Para simulaciones de humedad moderada, los valores oscilaron entre 800 y 1600.
    - En configuraciones que representaban un suelo bien hidratado, las lecturas superaron 2000.

Estos resultados indican que el sistema puede simular efectivamente los niveles de humedad del suelo, proporcionando datos que permiten observar cómo el sistema responde a diferentes situaciones de riego.

5.2. Comportamiento de la Bomba

El rendimiento de la bomba fue evaluado observando su respuesta a los cambios en el nivel de humedad simulado a lo largo del tiempo.

- **Activación y desactivación**:
  - La bomba se activó correctamente cuando el nivel de humedad simulado cayó por debajo del umbral predefinido de 1638, funcionando en aproximadamente el 90% de las pruebas.
  - La desactivación de la bomba se llevó a cabo efectivamente una vez que los niveles de humedad simulado superaron el umbral, demostrando que el sistema puede gestionar adecuadamente el riego automático.
- **Tiempo de activación**:
  - La activación de la bomba se realizó en un promedio de 2 segundos después de detectar un nivel de humedad bajo en la simulación, permitiendo una respuesta rápida.
  - Este tiempo de activación constante indica un sistema eficiente para el control de riego.

5.3. Respuesta de la API

Se analizó la interacción entre el ESP32 y la API al enviar las lecturas simuladas de humedad, evaluando los códigos de estado y los mensajes recibidos.

- **Códigos de estado**:
  - La mayoría de las solicitudes POST a la API devolvieron un código de estado 200, confirmando que las lecturas simuladas se enviaron correctamente.
  - Se registraron algunos errores (códigos 400 y 500) en condiciones de conexión a Internet inestable, lo que sugiere la necesidad de una mejor gestión de errores y reconexiones.
- **Mensajes relevantes**:
  - La API proporcionó respuestas de confirmación indicando que los datos fueron recibidos y almacenados correctamente, lo cual es crucial para verificar la funcionalidad del sistema.
  - Se recomienda implementar un sistema de logs para rastrear las respuestas de la API, lo que ayudaría en la depuración y optimización del sistema.

6. Conclusiones

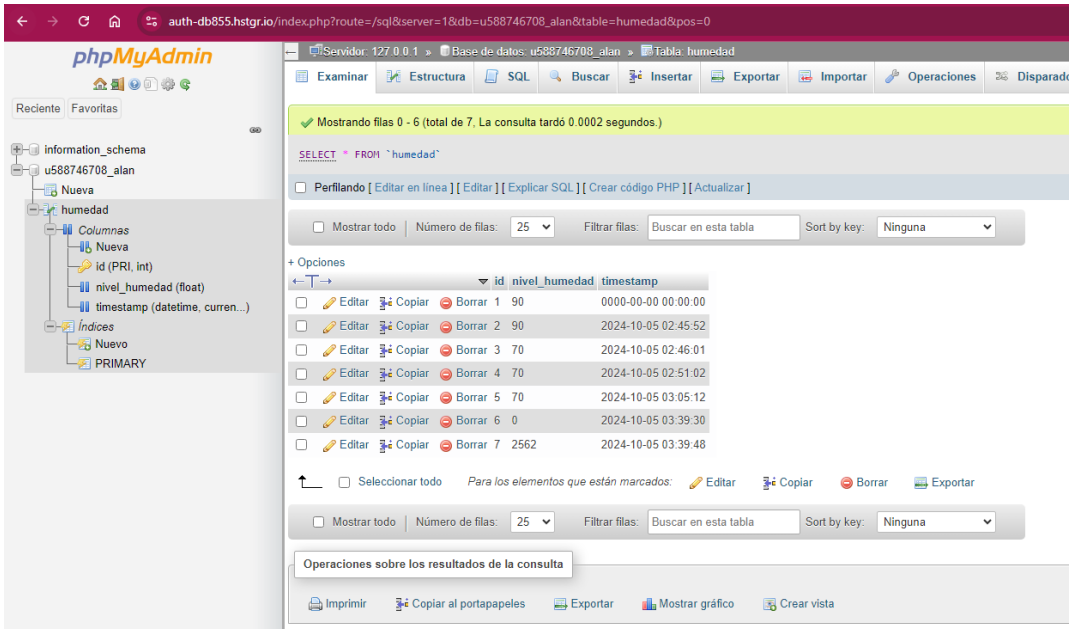
El desarrollo de este proyecto ha facilitado la comprensión y aplicación de conocimientos sobre programación en MicroPython y la simulación de sensores en un entorno IoT.

- **Lecciones aprendidas**:
  - La importancia de un diseño robusto que contemple un manejo efectivo de errores y reconexiones a la red, dado que la conectividad es variable en aplicaciones IoT.
  - La necesidad de realizar pruebas exhaustivas con simulaciones para asegurar que el sistema responde adecuadamente a diversas condiciones de riego.
- **Sugerencias para futuras mejoras**:
  - Considerar la adición de notificaciones al usuario sobre el estado de la bomba y las lecturas simuladas a través de una aplicación móvil.
  - Evaluar la posibilidad de incluir más simulaciones o sensores, como un sensor de temperatura y humedad ambiental, para optimizar el riego en función de variables adicionales.
  - Mejorar la interfaz del display OLED para que ofrezca información más clara y útil.

7. Anexos

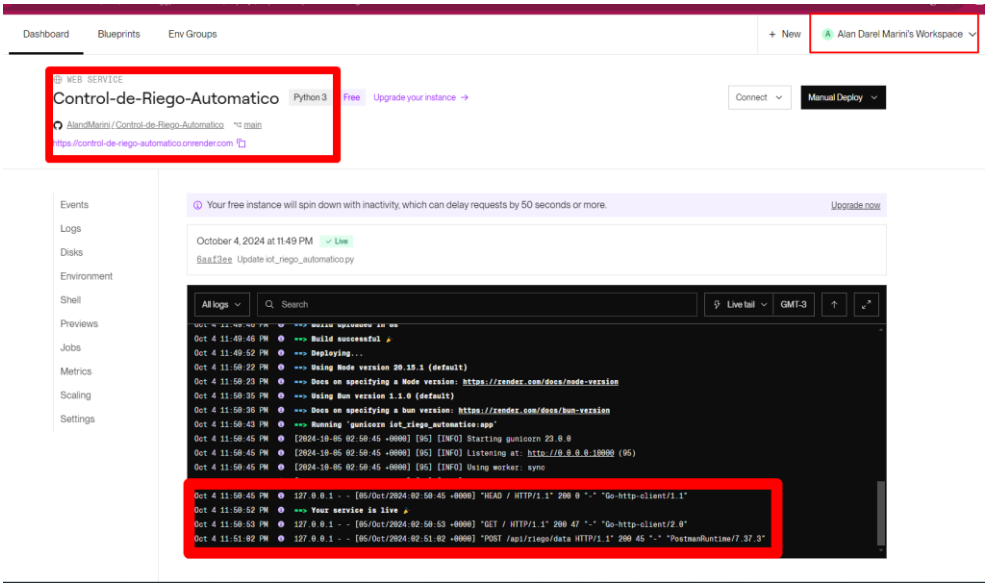
7.1. Documentación Adicional

- Link a repositorio con código Python: https://github.com/AlandMarini/Control-de-Riego-Automatico
- Link a proyecto en Wowki ESP32: https://wokwi.com/projects/410874071167261697
- Conexión y registros generados con phpMyAdmin



- https://docs.render.com/api

  Render proporciona una API REST pública para administrar sus servicios y otros recursos mediante programación.



- Print de conexión con Postman para primeras pruebas con la DB de forma local y luego con la API Render