TECNICATURA SUPERIOR EN Desarrollo del Software
Aproximación al Mundo del Trabajo **Profesor Mainero Alejandro**
05 de octubre de 2024

- **Título del trabajo: Monitoreo de Movimiento IoT**
- **Nombre de los estudiantes**
- **[Nahuel Argandoña] [Gastón Cané] [Eric Heredia]**

Índice

1. Introducción

El presente trabajo aborda el desarrollo de un sistema de monitoreo de movimiento utilizando un sensor PIR, un microcontrolador y una API para almacenar y recuperar datos en tiempo real. Se detalla la conexión a la base de datos y el proceso de envío de datos mediante solicitudes HTTP.

2. Objetivos

2.1. Objetivo General

Desarrollar un sistema de monitoreo que permita detectar movimiento y almacenar la información en una base de datos en la nube para su posterior análisis.

2.2. Objetivos Específicos

- **Implementar un microcontrolador con un sensor PIR para detectar movimiento.**
- **Configurar una API RESTful utilizando Flask para gestionar la interacción con la base de datos.**
- **Almacenar y recuperar datos desde una base de datos MySQL en Clever Cloud.**

3. Metodología

3.1. Diseño del Sistema

El sistema está compuesto por un microcontrolador que se conecta a una red Wi-Fi y utiliza un sensor PIR para detectar movimiento. Cuando se detecta movimiento, se envían datos a una API, que a su vez interactúa con una base de datos para almacenar la información.

3.2. Conexión a la Base de Datos

Se utiliza la biblioteca `mysql.connector` para gestionar la conexión con la base de datos y realizar operaciones de inserción y recuperación de datos.

4. Desarrollo

4.1. Código del Microcontrolador

```
from machine import Pin  # Librería para manejo de pines
import utime  # Librería para el manejo del tiempo
import network  # Librería para conectividad de red
import urequests  # Librería para hacer solicitudes HTTP

# Datos de Wi-Fi para Wokwi
SSID = 'Wokwi-GUEST'  # Red Wi-Fi simulada en Wokwi
PASSWORD = ''  # Sin contraseña en la red simulada

# Conectar a Wi-Fi
def conectar_wifi(ssid, password):
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)  # Activar el modo estación
```

```
    wlan.connect(ssid, password)  # Conectar a la red Wi-Fi

    # Esperar a que se conecte
    while not wlan.isconnected():
        print('Conectando a la red...')
        utime.sleep(1)

    print('Conectado a:', wlan.ifconfig())  # Mostrar dirección IP y
detalles de conexión

# Crear el objeto del PIR, LED y buzzer
pir = Pin(15, Pin.IN, Pin.PULL_DOWN)  # PIR conectado al pin 15 y GND
led_rojo = Pin(13, Pin.OUT)  # LED conectado al pin 13 (salida) y GND
buzzer = Pin(14, Pin.OUT)  # Buzzer conectado al pin 14 (salida) y GND

# Conectar a la red Wi-Fi simulada
conectar_wifi(SSID, PASSWORD)

# Función para enviar datos al servidor
def enviar_datos(sensor, valor, message):

    data = {'sensor': sensor, 'value': valor, 'message': message}
    print('Enviando datos:', data)

    try:
        response =
urequests.post('https://iot-lot-2.onrender.com/data', json=data)  #
Cambia la URL si es necesario
        print('Código de respuesta:', response.status_code)
        print('Respuesta JSON:', response.json())
    except Exception as e:
        print('Error en la solicitud:', e)

# Bucle principal
while True:
    estado = pir.value()  # Leer el valor del PIR (1 si detecta
movimiento, 0 si no)

    if estado == 0:  # Si no detecta movimiento
        led_rojo.value(0)  # Apagar el LED
        buzzer.value(1)  # Encender el buzzer
        print('No hay nadie en el área')
```

```
        enviar_datos('PIR', 0,'No se detecta movimiento cercano')  #
Enviar valor de 0 al servidor
    else:  # Si detecta movimiento
        led_rojo.value(1)  # Encender el LED
        buzzer.value(0)  # Apagar el buzzer
        print('Hay alguien en el área')
        enviar_datos('PIR', 1,'Se detecta movimiento cercano')  #
Enviar valor de 1 al servidor

    utime.sleep(2)  # Espera de 2 segundos
```

4.2. Código de la API con Flask

python

```python
from flask import Flask, request, jsonify
import mysql.connector

app = Flask(__name__)

# Configuración de la conexión a la base de datos
db_config = {
    "host": "bxy5ofa8ezud0x0caavs-mysql.services.clever-cloud.com",
    "user": "uupemiqze1zhijfn",
    "password": "XnExGlv7QzWuydrfjtLK",  # Reemplaza con tu contraseña
    "database": "bxy5ofa8ezud0x0caavs",
}

# Ruta para insertar datos
@app.route("/data", methods=["POST"])
def insert_data():
    if request.is_json:
        data = request.get_json()
        sensor = data.get("sensor")
        value = data.get("value")
        message = data.get("message")

        if sensor is None or value is None:
            return jsonify({"status": "error", "message": "Both
'sensor' and 'value' are required and 'message' are required"}), 400

        try:
```

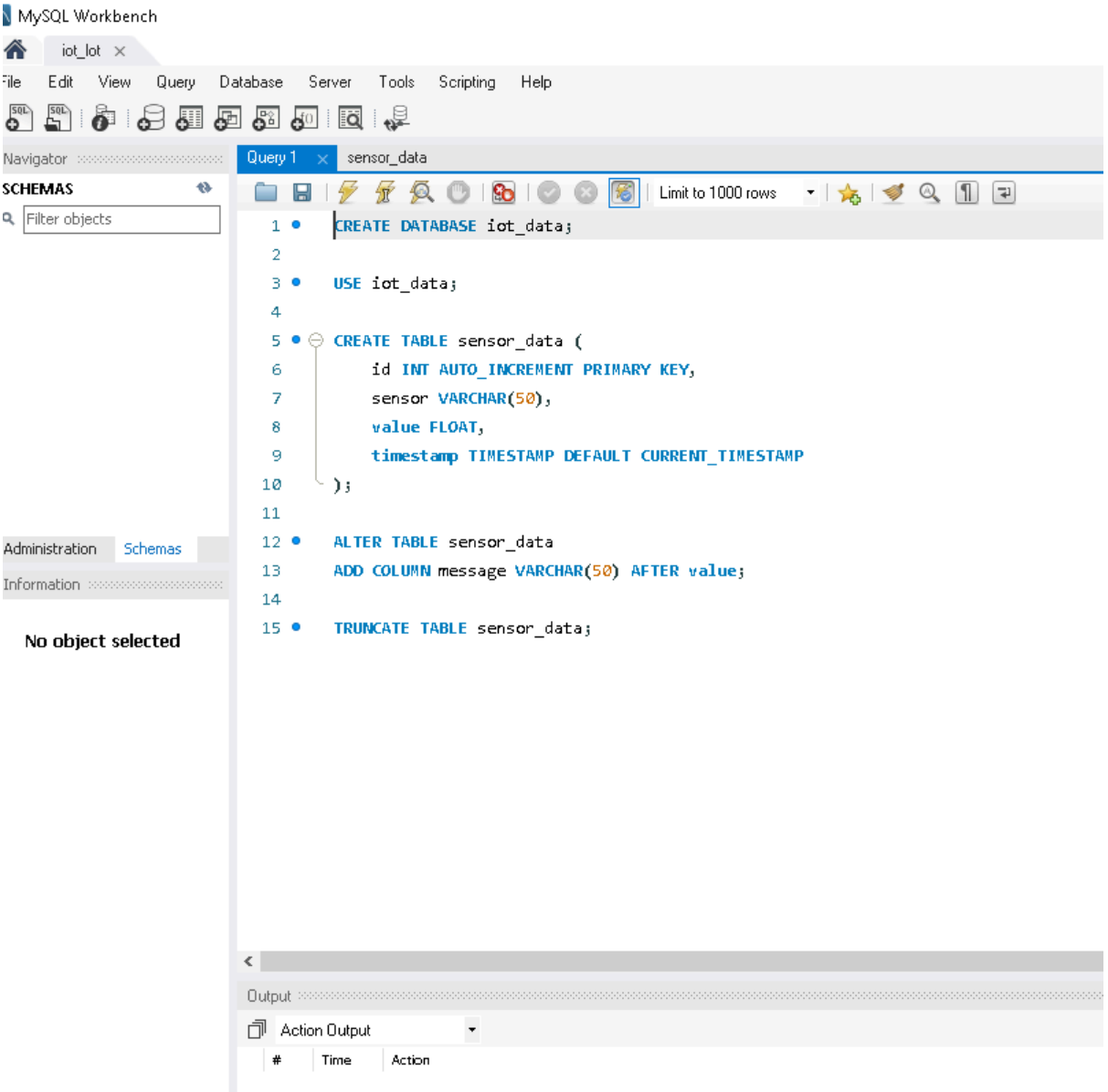
```
            conn = mysql.connector.connect(**db_config)
            cursor = conn.cursor()

            cursor.execute("INSERT INTO sensor_data (sensor, value,
message) VALUES (%s, %s, %s)", (sensor, value, message))
            conn.commit()

            cursor.close()
            conn.close()

            return jsonify({"status": "success"}), 201

        except mysql.connector.Error as err:
            return jsonify({"status": "error", "message": str(err)}),
500
    else:
        return jsonify({"status": "error", "message": "Request must be
JSON"}), 400


# Ruta para mostrar los datos almacenados
@app.route("/data", methods=["GET"])
def get_data():
    try:
        conn = mysql.connector.connect(**db_config)
        cursor = conn.cursor()

        # Ejecutar consulta para obtener todos los datos
        cursor.execute("SELECT id, sensor, value, message, timestamp
FROM sensor_data")
        rows = cursor.fetchall()

        # Cerrar el cursor y la conexión
        cursor.close()
        conn.close()

        # Formatear los datos como una lista de diccionarios
        results = []
        for row in rows:
            results.append({
                "id": row[0],
                "sensor": row[1],
                "value": row[2],
```

```
                "message":row[3],
                "timestamp": row[4].strftime("%Y-%m-%d %H:%M:%S")   #
Formato de fecha
            })

        return jsonify({"status": "success", "data": results}), 200

    except mysql.connector.Error as err:
        return jsonify({"status": "error", "message": str(err)}), 500


if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000)
```

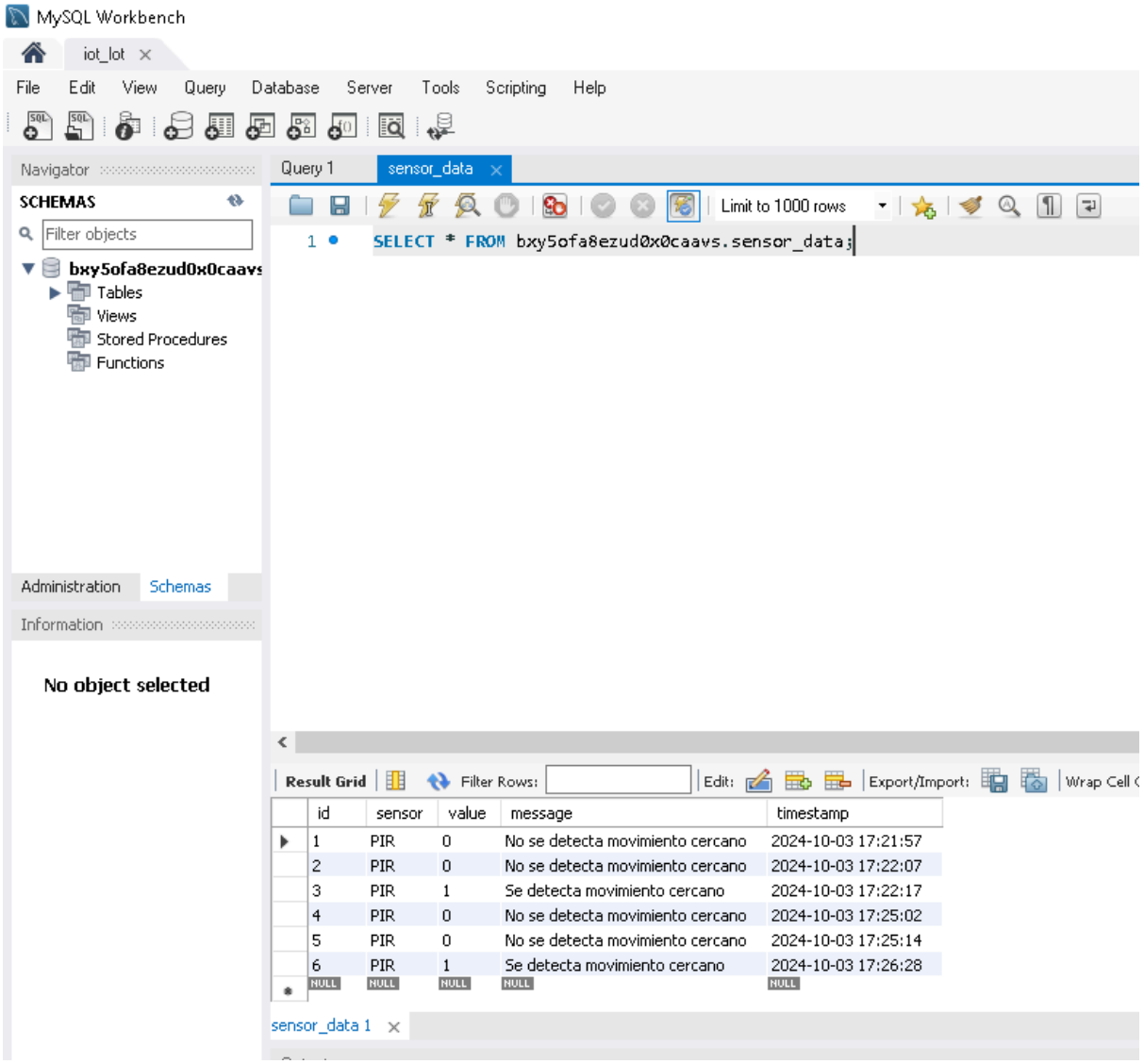
Requirements

```
Flask #Flask: El marco web que estás usando para desarrollar la API.
mysql-connector-python

# pip install -r requirements.txt para instalar las dependencias.

# Iniciar el Servidor Flask
# export FLASK_APP=app.py     # Para Linux o macOS
#set FLASK_APP=app.py        # Para Windows
# python app.py # vs code windows

#export FLASK_ENV=development  # (opcional) para habilitar el modo de
desarrollo
#flask run                    # Inicia el servidor
```

MYSQL DATA BASE



CONSULTA



5. Resultados

Los resultados del proyecto muestran la efectividad del sistema en la detección de movimiento y el correcto almacenamiento de datos en la base de datos.

6. Conclusiones

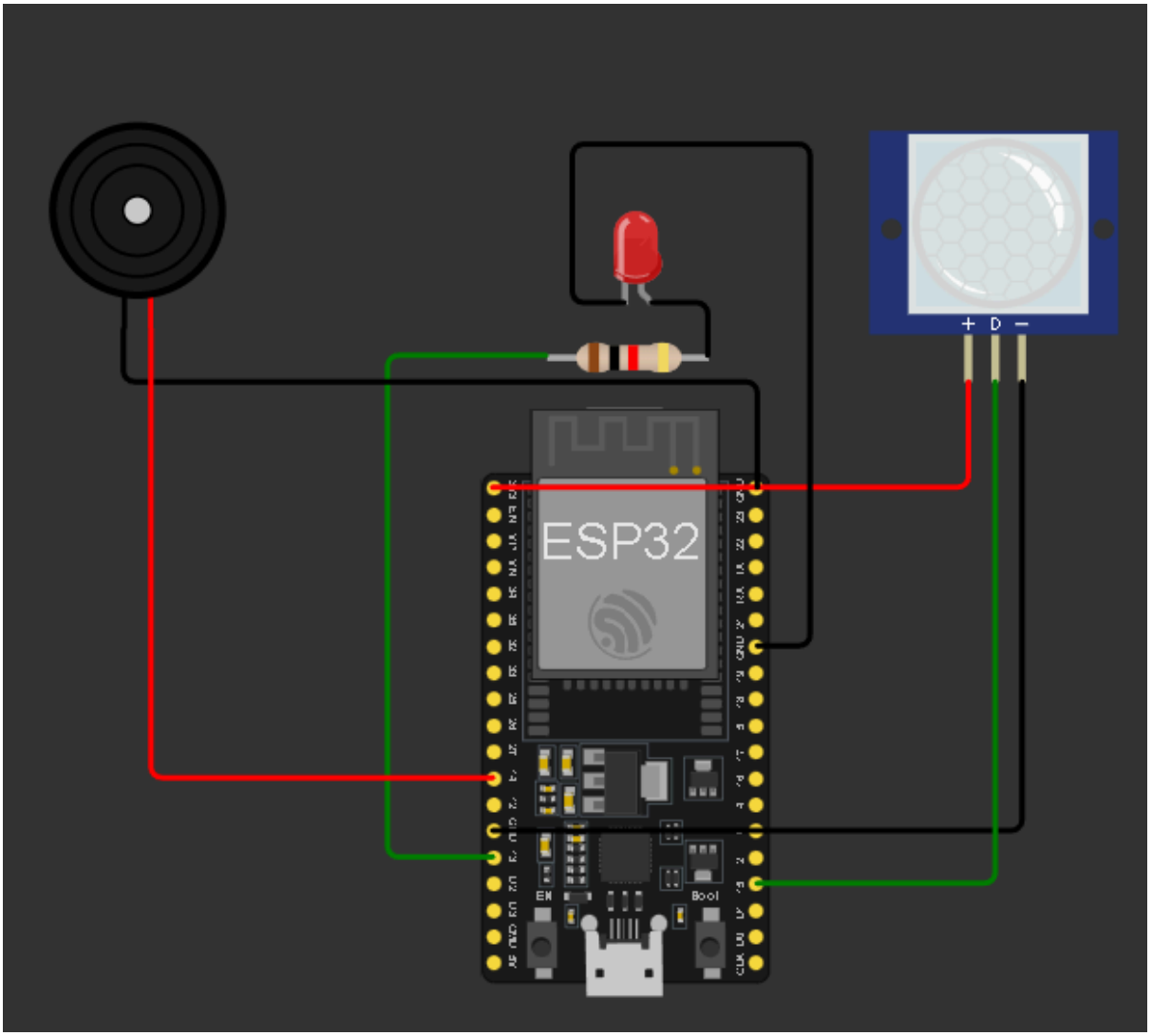
El sistema desarrollado cumple con los objetivos planteados, demostrando la viabilidad de soluciones IoT para el monitoreo en tiempo real.

7. Anexos

7.1. Diagramas del Sistema

```
{
  "version": 1,
  "author": "Nahuel Argandoña",
  "editor": "wokwi",
  "parts": [
    {
      "type": "board-esp32-devkit-c-v4",
      "id": "esp",
      "top": -57.6,
      "left": -71.96,
      "attrs": { "env": "micropython-20231227-v1.22.0" }
    },
    { "type": "wokwi-pir-motion-sensor", "id": "pir1", "top": -159.2,
"left": 69.42, "attrs": {} },
    { "type": "wokwi-led", "id": "led1", "top": -138, "left": -34.6,
"attrs": { "color": "red" } },
    {
      "type": "wokwi-resistor",
      "id": "r1",
      "top": -82.45,
      "left": -48,
      "attrs": { "value": "1000" }
    },
    {
      "type": "wokwi-buzzer",
      "id": "bz1",
      "top": -170.4,
      "left": -228.6,
      "attrs": { "volume": "0.1" }
    }
  ],
  "connections": [
    [ "esp:TX", "$serialMonitor:RX", "", [] ],
    [ "esp:RX", "$serialMonitor:TX", "", [] ],
    [ "pir1:VCC", "esp:3V3", "red", [ "v0" ] ],
    [ "pir1:GND", "esp:GND.1", "black", [ "v0" ] ],
```

```
    [ "pir1:OUT", "esp:15", "green", [ "v0" ] ],
    [ "led1:A", "r1:2", "black", [ "v0", "h38.4" ] ],
    [ "led1:C", "esp:GND.3", "black", [ "v0", "h-18.8", "v-57.6",
"h86.4", "v96" ] ],
    [ "r1:1", "esp:13", "green", [ "v0", "h-57.6", "v163.2" ] ],
    [ "bz1:1", "esp:GND.2", "black", [ "v19.2", "h240" ] ],
    [ "bz1:2", "esp:14", "red", [ "v0" ] ]
  ],
  "dependencies": {}
}
```



7.2. Documentación Adicional

https://www.clever-cloud.com/product/mysql/
https://docs.render.com/deploy-flask
https://wokwi.com/projects/406401817182924801
https://github.com/Aubar48/iot_lot_2

Tecnicatura Superior en Desarrollo del Software

E.C.: **Aproximación al Mundo del Trabajo**

Evidencia de Aprendizaje 2: Sensor de luminosidad / Fotoresistor

Profesor: **Mainero Alejandro**

Nombre de los estudiantes: **Nahuel Argandoña, Gastón Cané, Eric Heredia, Sofía Auzqui**

Índice

1. Introducción

El presente trabajo aborda el desarrollo del Escalado de Plataforma IoT desde la Capa Física a la Capa de Transporte de Datos con Almacenamiento en MySQL.

2. Objetivos

2.1. Objetivo General

Escalar una plataforma IoT que actualmente opera exclusivamente en la capa física, utilizando microcontroladores ESP32 conectados a sensores y actuadores, para incorporar una capa de transporte de datos que permita la transmisión eficiente de información hacia un servidor con una base de datos MySQL. Los datos sensorizados por los dispositivos ESP32 se transmiten utilizando tecnologías de transporte como HTTP.

2.2. Objetivos Específicos

Implementar la capa de transporte de datos en los dispositivos ESP32:

- Configurar los microcontroladores ESP32 para que transmitan los datos sensorizados utilizando una o varias tecnologías de transporte, como WiFi, MQTT, LoRa, NarrowBand, Zigbee o HTTP, según sea apropiado para la aplicación.

Desarrollar un protocolo de comunicación eficiente para la transmisión de datos:

- Definir y configurar el protocolo más adecuado para la transmisión de datos desde los ESP32 hacia el servidor, optimizando el rendimiento y minimizando el consumo de energía en los dispositivos.

Configurar un servidor para la recepción y almacenamiento de datos:

- Implementar un servidor que reciba la información transmitida por los ESP32 a través de la red, y configurar una base de datos MySQL para almacenar los datos de manera estructurada y eficiente.

Establecer conexiones directas a la base de datos MySQL:

- Evitar el uso de PHPMyAdmin y establecer conexiones directas entre los ESP32 y el servidor MySQL utilizando bibliotecas o frameworks adecuados para integrar dispositivos IoT con bases de datos de forma segura y escalable.

Garantizar la seguridad y confiabilidad en la transmisión de datos:

- Implementar mecanismos de seguridad, como encriptación de datos y autenticación, para garantizar la integridad y confidencialidad de los datos sensorizados durante la transmisión hacia el servidor.

Optimizar el proceso de almacenamiento y procesamiento de datos en la base de datos:

- Diseñar una estructura de base de datos eficiente que permita un almacenamiento óptimo de grandes volúmenes de datos, con posibilidad de consulta y procesamiento en tiempo real o diferido.

Realizar pruebas de escalabilidad y eficiencia de la plataforma IoT:

- Evaluar el rendimiento del sistema a medida que se conectan más dispositivos y se incremente el volumen de datos, asegurando que la plataforma pueda escalar sin afectar la eficiencia.

3. Metodología

3.1. Diseño del Sistema

El sistema está compuesto por un microcontrolador que se conecta a una red Wi-Fi y utiliza un fotoresitor para el control de luminosidad. El led se prende y apaga, al detectar si hay luz en el ambiente, se envían datos a una API, que a su vez interactúa con una base de datos para almacenar la información.

3.2. Conexión a la Base de Datos

Se utiliza la biblioteca mysql.connector para gestionar la conexión con la base de datos y realizar operaciones de inserción y recuperación de datos.

4. Desarrollo

4.1. Código del Microcontrolador

```
from machine import Pin, ADC, PWM  # Librería para manejo de pines, ADC
y PWM
import utime  # Librería para manejo del tiempo
import network  # Librería para conectividad de red

# Datos de Wi-Fi para Wokwi
SSID = 'Wokwi-GUEST'  # Red Wi-Fi simulada en Wokwi
PASSWORD = ''  # Sin contraseña en la red simulada

# Conectar a Wi-Fi
def conectar_wifi(ssid, password):
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)  # Activar el modo estación
    wlan.connect(ssid, password)  # Conectar a la red Wi-Fi

    # Esperar a que se conecte
    while not wlan.isconnected():
        print('Conectando a la red...')
        utime.sleep(1)

     print('Conectado a:', wlan.ifconfig())  # Mostrar dirección IP y
detalles de conexión


# Crear el objeto del LED con PWM y sensor LDR
led_rojo = PWM(Pin(13), freq=1000)  # LED en el pin 13 con una
frecuencia de 1000 Hz
ldr = ADC(Pin(34))  # LDR conectado al pin 34 (entrada analógica)
ldr.atten(ADC.ATTN_11DB)  # Configurar el rango de lectura del LDR
(0-3.3V)

# Umbrales para determinar los niveles de luz
umbral_bajo = 1000  # Ajusta este valor según el entorno
umbral_medio = 3000

# Conectar a la red Wi-Fi simulada
conectar_wifi(SSID, PASSWORD)

while True:
    valor_ldr = ldr.read()  # Leer el valor del LDR (rango de 0 a 4095)
```

```
    print(f"Valor LDR: {valor_ldr}")   # Mostrar el valor del LDR en
consola para debug

    if valor_ldr > umbral_medio:  # Si hay mucha luz, apagar el LED
        led_rojo.duty(1023)  # LED apagado
        print("Luz alta - LED apagado")

    elif umbral_bajo < valor_ldr <= umbral_medio:  # Si hay luz media,
ajustar brillo bajo del LED
         led_rojo.duty(512)   # LED con brillo bajo (rango PWM de 0 a
1023)
        print("Luz media - LED con brillo bajo")

    else:  # Si hay poca luz o está oscuro, ajustar brillo alto del LED
        led_rojo.duty(0)  # LED con brillo alto (rango PWM de 0 a 1023)
        print("Luz baja - LED con brillo alto")

    utime.sleep(1)  # Espera de 1 segundo entre lecturas
```

4.2. Código de la API con Flask

python

```
You, hace 3 horas | 1 author (You)
from flask import Flask, request, jsonify
import mysql.connector

app = Flask(__name__)

# Configuración de la conexión a la base de datos
db_config = {
    "host": "bczmbgnehppfn51vsosj-mysql.services.clever-cloud.com",
    "user": "uivkycfd3e4dnsvj",
    "password": "DLAbOBEZTX1NYrAu6eB5",  # Reemplaza con tu contraseña
    "database": "bczmbgnehppfn51vsosj",
}
```

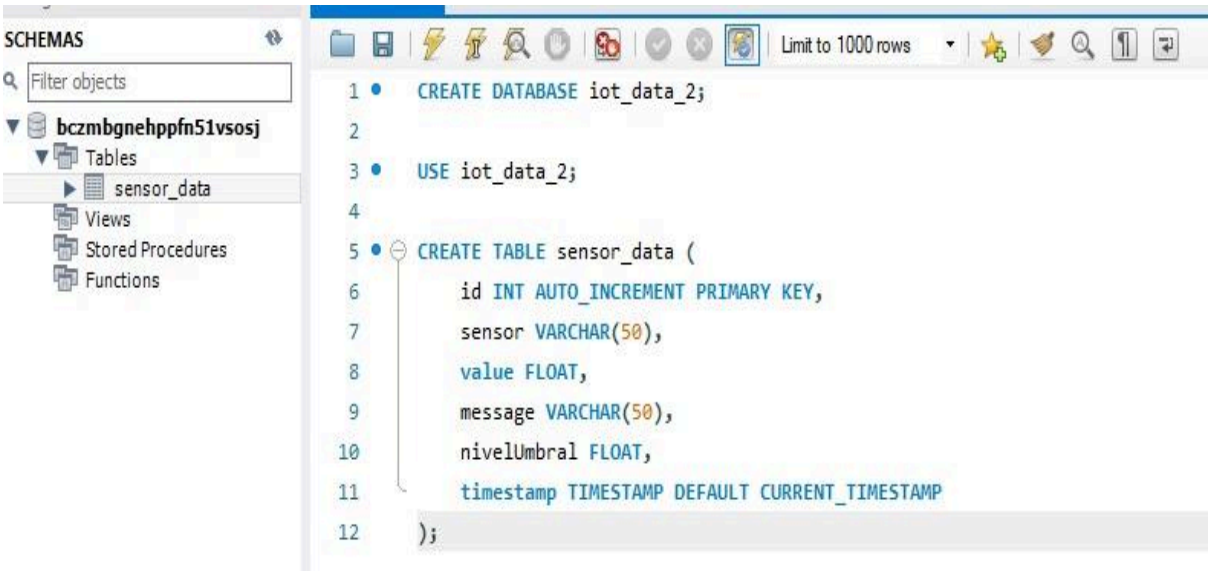
```python
# Ruta para insertar datos
@app.route("/data", methods=["POST"])      You, hace 31 minutos • uploading
def insert_data():
    if request.is_json:
        data = request.get_json()
        sensor = data.get("sensor")
        value = data.get("value")
        message = data.get("message")
        nivelUmbral = data.get("nivelUmbral")

        if sensor is None or value is None:
            return jsonify({"status": "error", "message": "Both 'sensor' and 'value' are required and 'message' are required and 'nivelUmbral' are required"}), 400

        try:
            conn = mysql.connector.connect(**db_config)
            cursor = conn.cursor()

            cursor.execute("INSERT INTO sensor_data (sensor, value, message,nivelUmbral) VALUES (%s, %s, %s,%s)", (sensor, value, message,nivelUmbral))
            conn.commit()

            cursor.close()
            conn.close()

            return jsonify({"status": "success"}), 201

        except mysql.connector.Error as err:
            return jsonify({"status": "error", "message": str(err)}), 500
    else:
```

```python

    # Ruta para mostrar los datos almacenados
    @app.route("/data", methods=["GET"])
    def get_data():
        try:
            conn = mysql.connector.connect(**db_config)
            cursor = conn.cursor()

            # Ejecutar consulta para obtener todos los datos
            cursor.execute("SELECT id, sensor, value, message,nivelUmbral, timestamp FROM sensor_data")      You, hace 3 horas • uploading
            rows = cursor.fetchall()

            # Cerrar el cursor y la conexión
            cursor.close()
            conn.close()

            # Formatear los datos como una lista de diccionarios
            results = []
            for row in rows:
                results.append({
                    "id": row[0],
                    "sensor": row[1],
                    "value": row[2],
                    "message":row[3],
                    "nivelUmbral": row[4],
                    "timestamp": row[5].strftime("%Y-%m-%d %H:%M:%S")  # Formato de fecha
                })

            return jsonify({"status": "success", "data": results}), 200

        except mysql.connector.Error as err:
            return jsonify({"status": "error", "message": str(err)}), 500


    if __name__ == "__main__":
        app.run(host="0.0.0.0", port=5000)

```

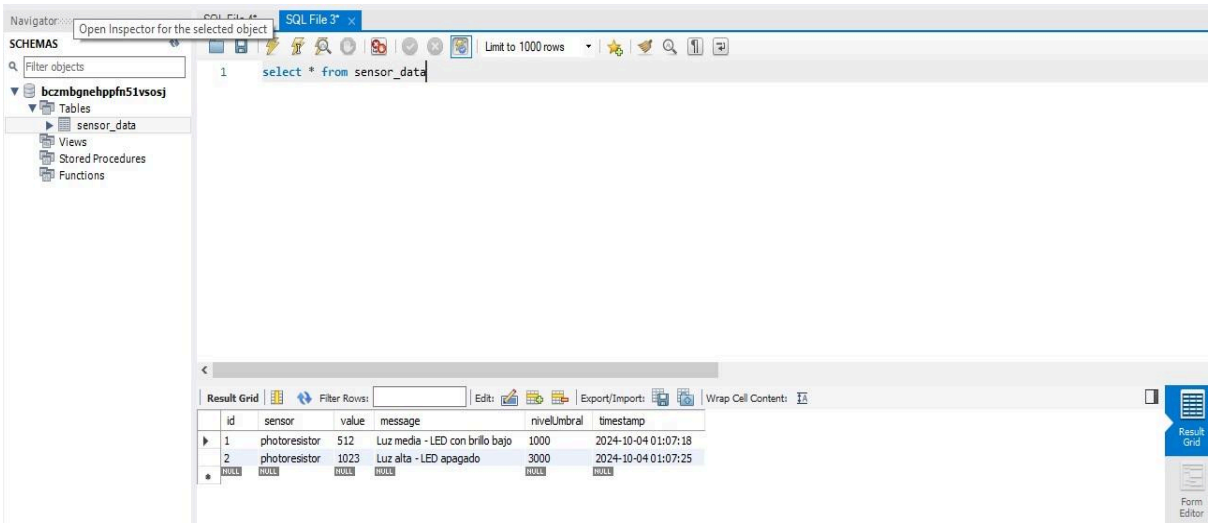
Requirements

```
Flask #Flask: El marco web que estás usando para desarrollar la API.
mysql-connector-python

# pip install -r requirements.txt para instalar las dependencias.

# Iniciar el Servidor Flask
# export FLASK_APP=app.py    # Para Linux o macOS
#set FLASK_APP=app.py       # Para Windows
# python app.py # vs code windows

#export FLASK_ENV=development   # (opcional) para habilitar el modo de desarrollo
#flask run                  # Inicia el servidor
```

MYSQL DATA BASE



CONSULTA




```
// 20241003220736
// https://iot-75r2.onrender.com/data

{
  "data": [
    {
      "id": 1,
      "message": "Luz media - LED con brillo bajo",
      "nivelUmbral": 1000.0,
      "sensor": "photoresistor",
      "timestamp": "2024-10-04 01:07:18",
      "value": 512.0
    },
    {
      "id": 2,
      "message": "Luz alta - LED apagado",
      "nivelUmbral": 3000.0,
      "sensor": "photoresistor",
      "timestamp": "2024-10-04 01:07:25",
      "value": 1023.0
    }
  ],
  "status": "success"
}
```

5. Resultados

Los resultados del proyecto muestran la efectividad del sistema en el control de luminosidad. El led se prende y apaga, al detectar si hay luz en el ambiente. Y el correcto almacenamiento de datos en la base de datos.

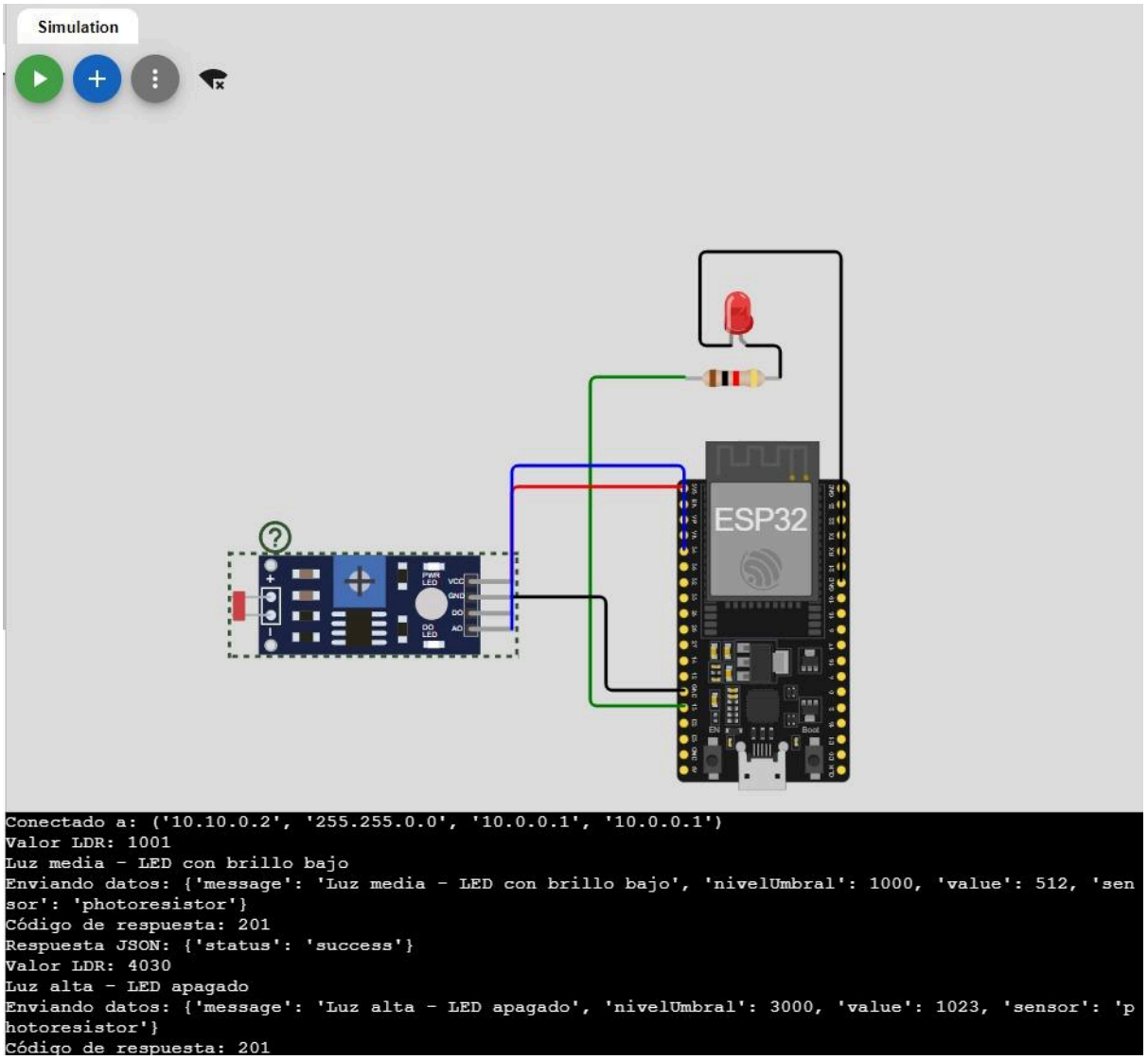
6. Conclusiones

El sistema desarrollado cumple con los objetivos planteados, demostrando la viabilidad de soluciones IoT para el monitoreo en tiempo real.

7. Anexos

7.1. Diagramas del Sistema


```
{
  "version": 1,
  "author": "Nahuel Argandoña",
  "editor": "wokwi",
  "parts": [
    {
      "type": "board-esp32-devkit-c-v4",
      "id": "esp",
      "top": -38.4,
      "left": -52.76,
      "attrs": { "env": "micropython-20231227-v1.22.0" }
    },
    { "type": "wokwi-led", "id": "led1", "top": -138, "left": -34.6, "attrs": { "color": "red
    {
      "type": "wokwi-resistor",
      "id": "r1",
      "top": -82.45,
      "left": -48,
      "attrs": { "value": "1000" }
    },
    { "type": "wokwi-photoresistor-sensor", "id": "ldr1", "top": 32, "left": -325.6, "attrs":
  ],
  "connections": [
    [ "esp:TX", "$serialMonitor:RX", "", [] ],
    [ "esp:RX", "$serialMonitor:TX", "", [] ],
    [ "led1:A", "r1:2", "black", [ "v0", "h38.4" ] ],
    [ "led1:C", "esp:GND.3", "black", [ "v0", "h-18.8", "v-57.6", "h86.4", "v96" ] ],
    [ "r1:1", "esp:13", "green", [ "v0", "h-57.6", "v163.2" ] ],
    [ "ldr1:VCC", "esp:3V3", "red", [ "h0", "v-163.2" ] ],
    [ "ldr1:GND", "esp:GND.1", "black", [ "h57.6", "v-38.8" ] ],
    [ "ldr1:AO", "esp:34", "blue", [ "h0", "v-100", "h200" ] ]
  ],
  "dependencies": {}
}
```



7.2. Documentación Adicional

https://www.clever-cloud.com/product/mysql/
https://docs.render.com/deploy-flask
https://wokwi.com/projects/410768705395252225
https://github.com/GasmauC/IoT