Internet Report

Team Name:

Mamamia

Faculty:

IET

Major:

Networks Engineering

Done By:
Youssef Ismail 58-12389
Omar Khaled 58-6794
Yehia Nader 58-18085

Supervised by:
Prof. Tallal El Shabrawy
Eng. Lucian Emad

Introduction and goal:

This project is used to monitor environmental applications that face us in daily life by measuring temperature and humidity and detecting if readings aren't in the appropriate range. It uses a DHT sensor to measure the needed readings and act according to them. The goal of the project is to implement a smart system that is useful in real-life applications where temperature and humidity readings need to be kept under control in a certain range.

Code with explanation:

Server code:

```
import socket
import threading

# Define normal temperature range
NORMAL_TEMP_RANGE = [24, 34]  # Example: Normal temperature range is 24 to 34 degrees
Celsius

# Function to handle communication with a single client
def handle_client(client_socket, client_address):
    print(f"Connection established with {client_address}")

    try:
        while True:
            # Receive message from the client
            message = client_socket.recv(1024).decode('utf-8')

            if not message:
                print(f"Connection closed by {client_address}")
                break

            print(f"Received data from {client_address}: {message}")

            try:
                # Assume the message contains temperature data
                temperature = int(message[0] + message[1])
                humidity = int(message[3] + message[4])
                if NORMAL_TEMP_RANGE[0] <= temperature <= NORMAL_TEMP_RANGE[1]:
                    t_response = "NORMAL"
                elif (temperature > NORMAL_TEMP_RANGE[1]):
                    t_response = "TOO HIGH"
                else:
                    t_response = "TOO LOW"
                if humidity > 40:
                    h_response = "Abnormal"
                else:
                    h_response = "Normal"

                # Send response to the client
                response = f"Temperature: {t_response} \n Humidity: {h_response}"
                client_socket.send(response.encode('utf-8'))

            except ValueError:
                print(f"Invalid data received from {client_address}: {message}")
                client_socket.send("ERROR: Invalid data".encode('utf-8'))

    except ConnectionResetError:
        print(f"Connection reset by {client_address}")

    finally:
        client_socket.close()
        print(f"Connection with {client_address} closed")

# Main server code
def start_server():
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind(('0.0.0.0', 5605))
    server_socket.listen(5)

    print("Server is listening on port 5605...")

    while True:
        try:
            # Accept new client connection
            client_socket, client_address = server_socket.accept()
            print(f"New connection from {client_address}")

            # Create a new thread to handle the client's connection
            client_thread = threading.Thread(target=handle_client, args=(client_socket,
client_address))
            client_thread.start()

        except KeyboardInterrupt:
            print("Server shutting down...")
            server_socket.close()
            break

if __name__ == "__main__":
    start_server()
```

The server listens from 2 clients. It first identifies the normal temperature range. Then, a method tries to receive a message from a client and decode it. If no message arrives, it closes the connection. Characters in index 0,1 represent the temperature and characters in index 3,4 represent humidity. Index 2 is ignored as it represents the ',' between temperature and humidity. Then, there are conditions for temperature and humidity that decide what the responses will be (t_response denoting temperature response and h_response denoting humidity response). The responses are encoded and sent back to the corresponding client. In the main method, the server creates a new socket, binds IP address and port number and listens for clients. A maximum queue size of 5 is allowed for clients that are waiting for the server. If more than that arrive, they will be discarded. The server accepts a connection from the client and starts listening. Threading is used to allow the server to handle multiple clients simultaneously. In case of a Keyboard Interrupt, it shuts down the connection.

—---

<u>Client code:</u>

```
import time
import socket
import network
import dht
from machine import Pin

# Wi-Fi credentials
SSID = "Galaxy A52 5G665E"
PASSWORD = "fmnn7937"

# Server address and port
SERVER_IP = "192.168.212.173"  # Replace with the server's IP address
SERVER_PORT = 5605

# GPIO pin assignments
DHT_SENSOR_PIN = 13  # Pin connected to the DHT sensor
LED_LOW_PIN = 4      # Pin connected to the LED for low temperature
```

```
LED_HIGH_PIN = 5    # Pin connected to the LED for high temperature

# Initialize the DHT sensor and LEDs
dht_sensor = dht.DHT11(Pin(DHT_SENSOR_PIN))
led_low = Pin(LED_LOW_PIN, Pin.OUT)
led_high = Pin(LED_HIGH_PIN, Pin.OUT)

# Connect to Wi-Fi
def connect_to_wifi():
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    wlan.connect(SSID, PASSWORD)

    print("Connecting to Wi-Fi...")
    while not wlan.isconnected():
        time.sleep(5)

    print("Connected to Wi-Fi", wlan.ifconfig())

# Send sensor readings to the server
def send_readings_to_server(client_socket):
    try:
        dht_sensor.measure()
        temperature = dht_sensor.temperature()
        humidity = dht_sensor.humidity()

        message = f"{temperature},{humidity}"
        client_socket.send(message.encode('utf-8'))
        print(f"Sent to server: {message}")

        # Receive response from the server
        response = client_socket.recv(1024).decode('utf-8')
        print(f"Server response: {response}")

        # Handle server response
        if "TOO LOW" in response:
            led_low.on()
            led_high.off()
        elif "TOO HIGH" in response:
            led_high.on()
            led_low.off()
        else:
            led_low.off()
            led_high.off()
    except Exception as e:
        print("Error reading sensor or sending data:", e)

# Main client code
def main():
    connect_to_wifi()

    # Connect to the server
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        client_socket.connect((SERVER_IP, SERVER_PORT))
        print("Connected to server")

        while True:
            send_readings_to_server(client_socket)
            time.sleep(3)  # Wait 3 seconds before sending the next reading

    except Exception as e:
        print("Connection error:", e)

    finally:
        client_socket.close()
        print("Socket closed")

if __name__ == "__main__":
    main()
```

In the client code, the Wi-Fi credentials of the local Wi-Fi network are specified. Server IP address and port to which client will connect are specified. Pins of ESP board that are connected to the sensor are specified. The client then attempts to connect to the Wi-Fi network every 5 seconds. When connected, it prints connection details. When connected, it measures temperature and humidity, encodes readings and sends them to the server. Then, it receives an appropriate response from the server according to readings and decodes it. According to the server response, it decides which LED will be turned on (high or low or neither if normal). In the main method, it attempts to connect to Wi-Fi then creates a TCP socket and attempts to connect to the server using IP address and port. When connected, it calls the method explained above that sends readings to the server and acts appropriately according to response. It waits for a time of 3 seconds after sending each reading to send the next one. It also handles if an error happens by throwing an exception and printing a message. Finally, it closes the connection.

<u>**Choices of timer and temperature ranges:**</u>

Regarding the timer, sending a reading every 3 seconds is reasonable as it gives the sensor enough time to measure properly without having effect from the previous measurement. Also, waiting for more than 3 seconds may cause some delays and long wait-time till receiving a response. For the temperature, a range of [24,34] is reasonable especially for an indoor environment as it is close to normal room temperature and also handles if a slight surprising increase in temperature occurs.

Diagram of Circuit of Client 1

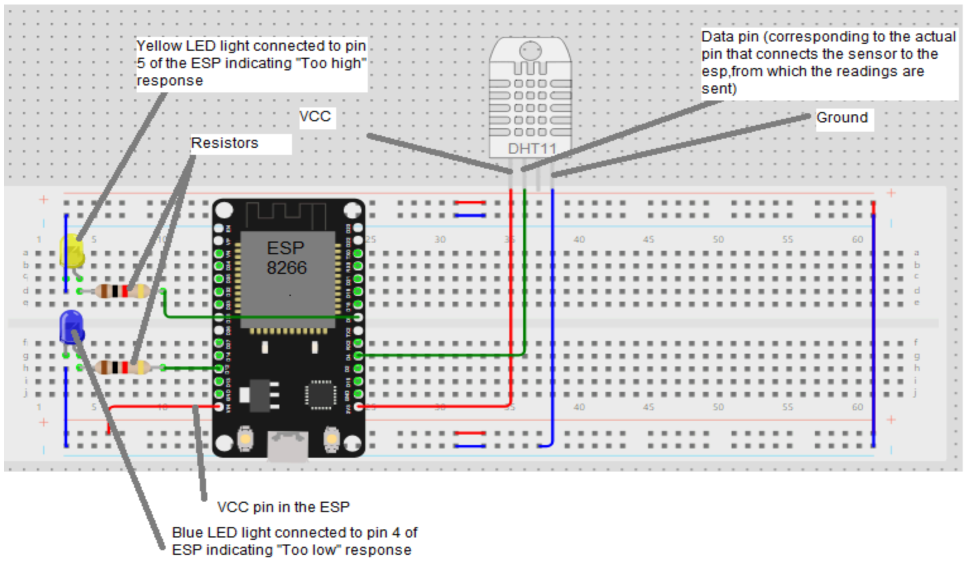
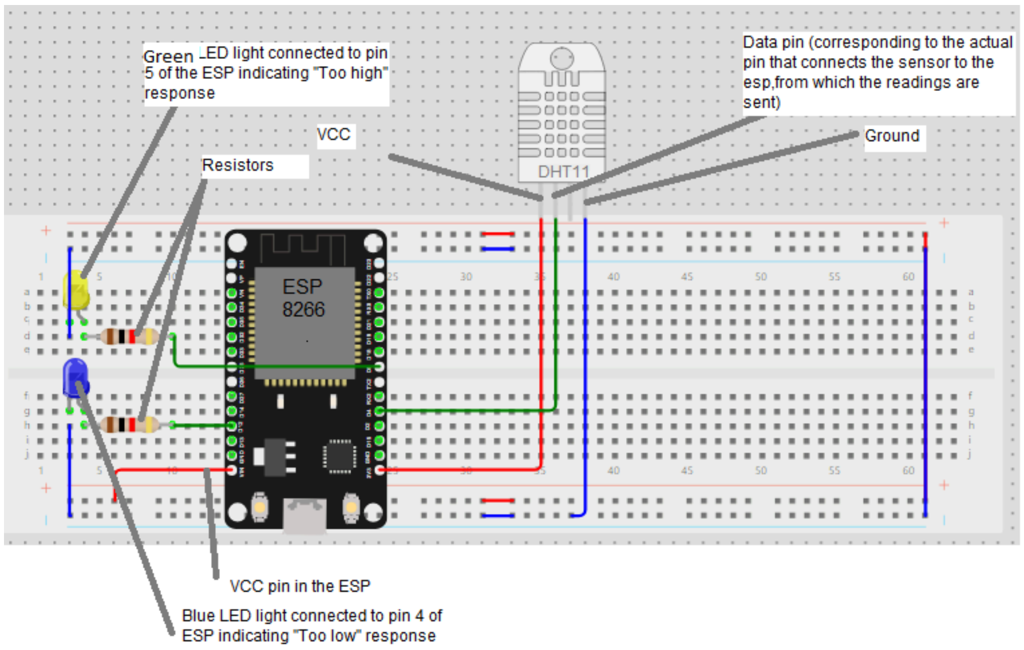

Yellow LED light connected to pin 5 of the ESP indicating "Too high" response
VCC
Resistors
DHT11
Data pin (corresponding to the actual pin that connects the sensor to the esp,from which the readings are sent)
Ground
ESP 8266
VCC pin in the ESP
Blue LED light connected to pin 4 of ESP indicating "Too low" response

Diagram of Circuit of Client 2


Green LED light connected to pin 5 of the ESP indicating "Too high" response
VCC
Resistors
DHT11
Data pin (corresponding to the actual pin that connects the sensor to the esp,from which the readings are sent)
Ground
ESP 8266
VCC pin in the ESP
Blue LED light connected to pin 4 of ESP indicating "Too low" response

<u>Circuit Construction:</u>

To begin with, our project mainly consisted of 2 circuits(one for each client). Furthermore, the components used for each circuit were as follows:

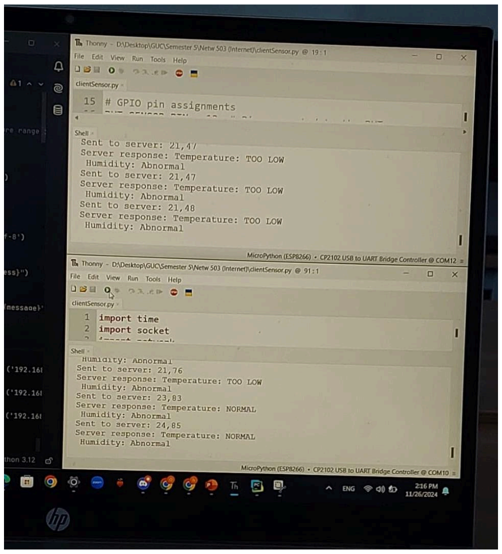
- 1 ESP8266
- 1 DHT11(temperature and humidity sensor)
- 2 LEDs of different colours(colours used were blue, green and yellow)
- 2 resistors
- Micro USB cable
- Wires
- 1 breadboard

<u>Connection:</u>

The connections for each circuit were as follows:
Firstly, the ESP was inserted on the breadboard and the GND pin was connected to the ground of the breadboard and the v3 pin (responsible to provide power to the ESP, which runs on 3 volts) in the ESP was connected to the positive rails of the breadboard. Next, 2 LEDs were also inserted in the breadboard. Furthermore, each LED has a short leg that is connected to the ground(negative rails of breadboard) and long leg that is connected to the ESP through pins 4 and 5 (LED connected on Pin 4 is responsible for the “Too Low response” while the LED connected on pin 5 is responsible for the “Too High” response”). Between each Long leg of the LED and the Pins of the ESP, a resistor is placed inorder to prevent the LEDs from overheating and burning. Then, the DHT11 is inserted in the breadboard. The DHT11 has 3 pins, GND pin that is connected to the ground(negative rail of the breadboard), VCC pin connected to the positive rail of the breadboard and finally the data pin that is connected to the ESP through pin 13. Finally, in order to power up our circuit, it was connected to the computer through a micro USB cable inserted in the port of the ESP (the power source was the computer). Moreover, the connections stated above were repeated for circuit 2.

<u>Implementation and test cases:</u>

Firstly, we tested the connection between the LEDs and the ESP through running an implemented code Known as ‘Pin tester’ in order to make sure the connections were successful. Next, make sure our connections and code were correctly functioning we carried out some test cases. Secondly, to test that the DHT11 was actually functioning we ran the client code and observed the readings being displayed and we applied some external changes in order to cause variations in the temperatures,such as blowing hot air onto sensor to increase the temp and leaving the sensor near cold water to decrease temp. to make sure that the sensor was changing values.



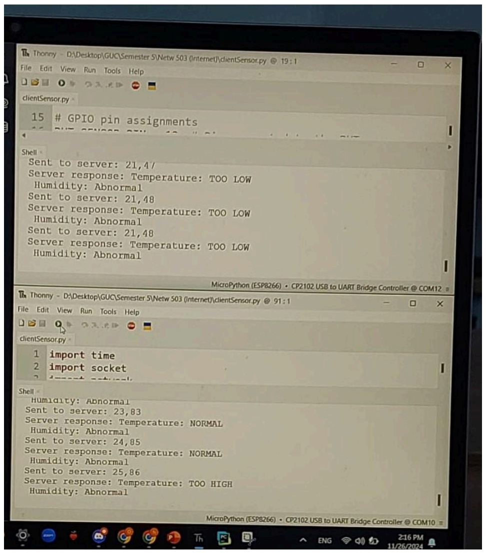
```
# GPIO pin assignments
```

```
Sent to server: 21,47
Server response: Temperature: TOO LOW
 Humidity: Abnormal
Sent to server: 21,47
Server response: Temperature: TOO LOW
 Humidity: Abnormal
Sent to server: 21,48
Server response: Temperature: TOO LOW
 Humidity: Abnormal
```

```
import time
import socket
```

```
 Humidity: Abnormal
Sent to server: 21,76
Server response: Temperature: TOO LOW
 Humidity: Abnormal
Sent to server: 23,83
Server response: Temperature: NORMAL
 Humidity: Abnormal
Sent to server: 24,85
Server response: Temperature: NORMAL
 Humidity: Abnormal
```

Thonny - D:\Desktop\GUC\Semester 5\Netw 503 (Internet)\clientSensor.py @ 19:1
File Edit View Run Tools Help
clientSensor.py
Shell
MicroPython (ESP8266) • CP2102 USB to UART Bridge Controller @ COM12
Thonny - D:\Desktop\GUC\Semester 5\Netw 503 (Internet)\clientSensor.py @ 91:1
MicroPython (ESP8266) • CP2102 USB to UART Bridge Controller @ COM10
2:16 PM
11/26/2024



```
# GPIO pin assignments
```

```
Sent to server: 21,47
Server response: Temperature: TOO LOW
 Humidity: Abnormal
Sent to server: 21,48
Server response: Temperature: TOO LOW
 Humidity: Abnormal
Sent to server: 21,48
Server response: Temperature: TOO LOW
 Humidity: Abnormal
```

```
import time
import socket
```

```
 Humidity: Abnormal
Sent to server: 23,83
Server response: Temperature: NORMAL
 Humidity: Abnormal
Sent to server: 24,85
Server response: Temperature: NORMAL
 Humidity: Abnormal
Sent to server: 25,86
Server response: Temperature: TOO HIGH
 Humidity: Abnormal
```

Thonny - D:\Desktop\GUC\Semester 5\Netw 503 (Internet)\clientSensor.py @ 19:1
File Edit View Run Tools Help
clientSensor.py
Shell
MicroPython (ESP8266) • CP2102 USB to UART Bridge Controller @ COM12
Thonny - D:\Desktop\GUC\Semester 5\Netw 503 (Internet)\clientSensor.py @ 91:1
MicroPython (ESP8266) • CP2102 USB to UART Bridge Controller @ COM10
2:16 PM
11/26/2024

Furthermore, to test that the code was working in parallel with the hardware components, we tested both circuits while changing the temperature of each circuit alone and observed the changes in both the readings and the light bulbs of each circuit.

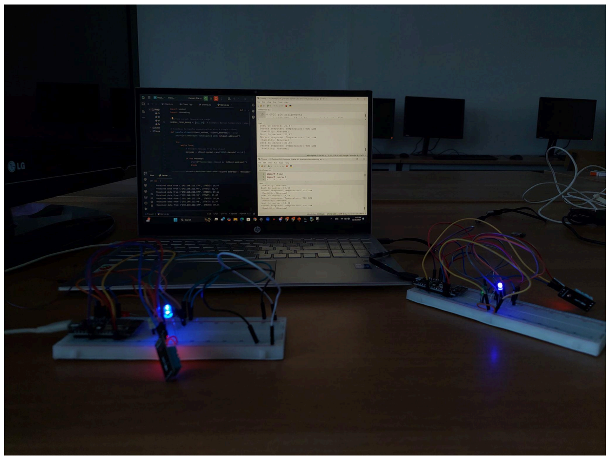


Fig 1: both circuits have their LEDs that indicate low temperatures lit, also shown in the readings

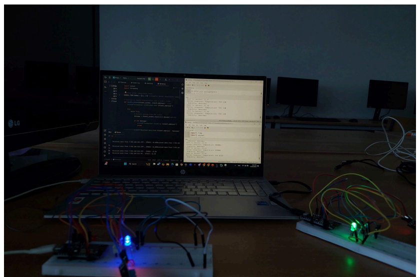


Fig 2: circuit 1 has a blue LED indicating low temp while circuit 2 has green LED indicating high temp, also shown through readings gathered

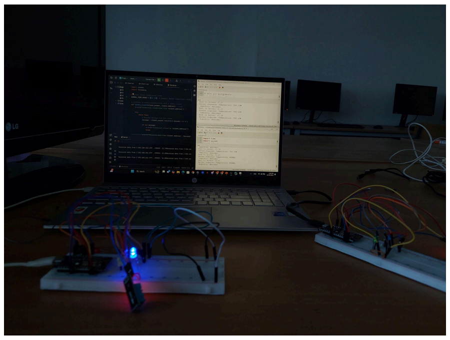


Fig 3:circuit 1 has a blue LED indicating low temp while circuit 2 has no LEDs lit indicating normal temperature, also shown through readings gathered

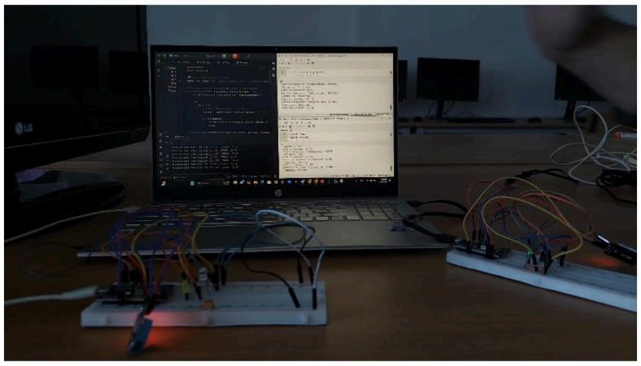


Fig 4:both circuit 1 and 2 have no LEDs lit indicating normal Temperature, also shown through readings gathered

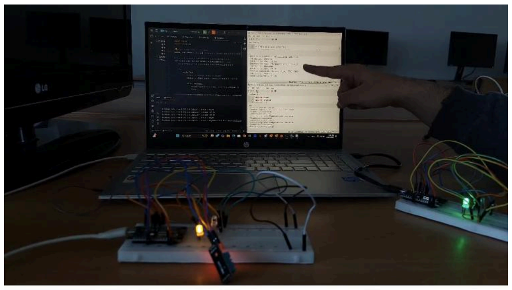


Fig 2:both circuits have their LEDs that indicate high temperatures lit, also shown through readings gathered

Testing the project:

The project was tested using both hardware and software. The ESP board was connected to the laptop and server code was run to initiate the connection. Then client code was run and the sensor started measuring temperature while the server was sending back its responses to the client. Based on these responses, the appropriate LED was lighted on. To make sure that this process works correctly, it was necessary to check on pins to which components are corrected to the ESP board and place them in code. Also, we needed to make sure that the sensor was reading correctly and in correct time intervals. Regarding software, it was mandatory to check that the server applies threading correctly to be able to handle multiple clients simultaneously and that the client initially connects to the network using appropriate credentials. In addition, some exceptions had to be thrown to handle errors when testing.

Use-cases (Real-life applications):

This project has multiple real life applications. It may be used to track temperature and humidity readings indoor where the LEDs act as warning if they light (if temperature goes above or below expected) indicating that a certain action should be taken to get temperature back to normal range for instance. This may be used in places where temperature readings going higher or lower than expected may cause fatal damage such as hospitals for patient care and Intensive Care Units. A smart use may also be to preserve food such as dairy products where temperature needs to stay low to prevent food poisoning (In this application, the range [24,34] won't be smart to use, a much smaller range is needed like [0,4]. We will mainly care about when temperature gets higher than expected). A practical application for humidity is placing this system in historic artifacts and museums to detect if humidity increases more than expected. High humidity may cause wooden artifacts to crack and paintings to fade.