# 2025 TEEP Progress Report Week- 6

## Used Wokwi to complete the tasks for Labs 1 through 6

(Before creating the physical project, please use **Wokwi** for online simulation to complete the initial design of the circuit and code. Then, use **Thonny** together with the **ESP32** development board to build the actual project. Please **record a video of the project** in operation, upload it to YouTube, and provide the video link(LAB7) so we can better understand your learning progress.)

**Lab1- On Wokwi , complete the following tasks:**

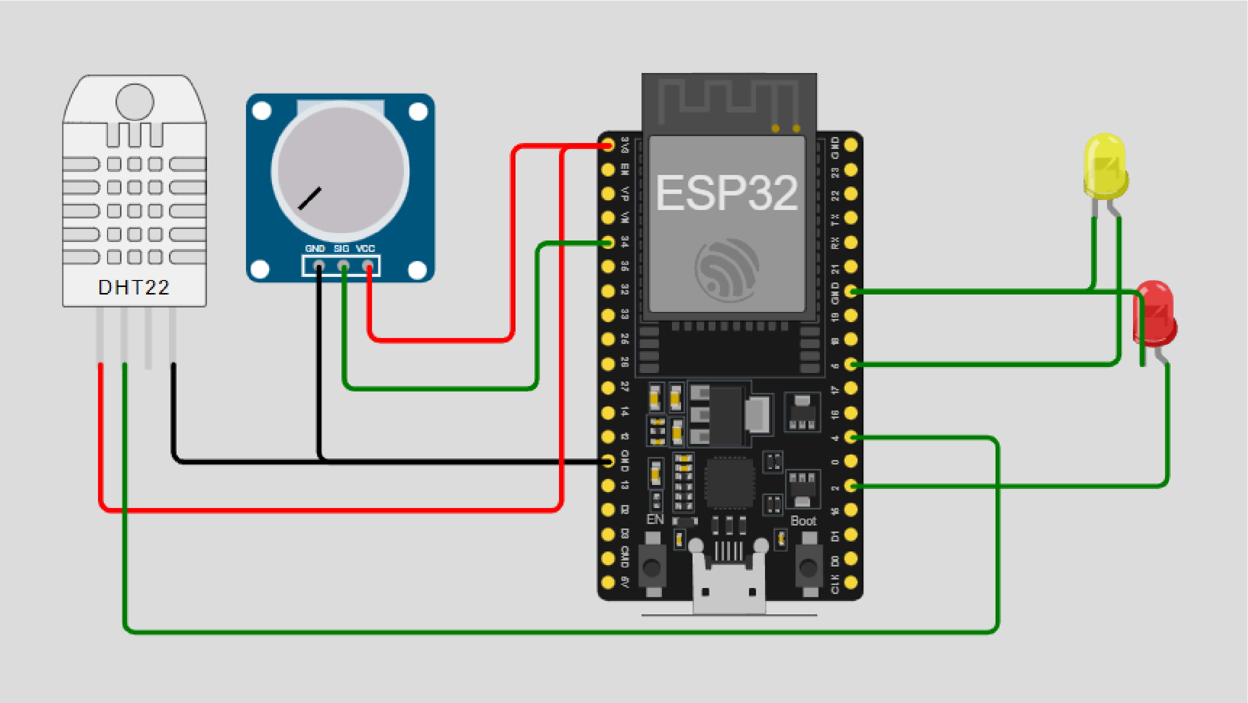
Read the voltage value on A0, DHT22 Temperature/humidity data and upload it to ThingSpeak:

1. Connect the potentiometer to the circuit consisting of the ADC to GPIO0, DHT22 to GPIOxxx, yellow LED (LED_Hum) to GPIOxxx and red LED (LED_Temp) to GPIOxxx. Complete the following steps:

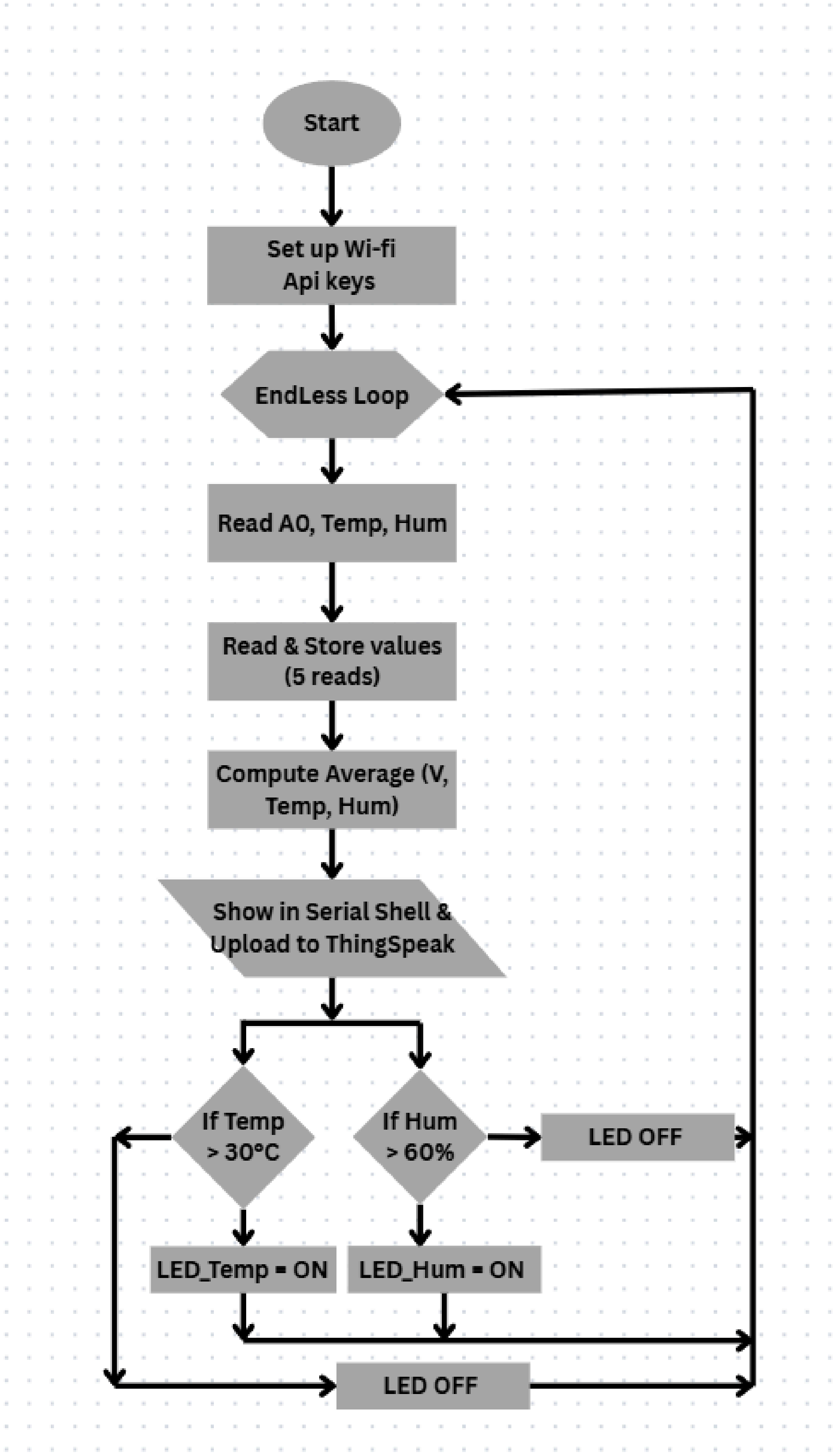
   - **Function 1**: When the program starts, it detects the voltage on ADC every 3 seconds (limited to 0~3.3V), obtains the temperature and humidity data of DHT22, and displays the average value in Shell after obtaining 5 records, and uploads it to ThingSpeak.
   - **Function 2**:When the temperature is above 30 degrees, LED_Temp lights up.
   - **Function 3:**When the humidity is higher than 60%, LED_Hum lights up.

**Connections :**

- Potentiometer → GPIO34 (ADC1_6, safe for analog)
- DHT22 → GPIO4
- Yellow LED (LED_Hum) → GPIO5
- Red LED (LED_Temp) → GPIO2



**FlowChart :**

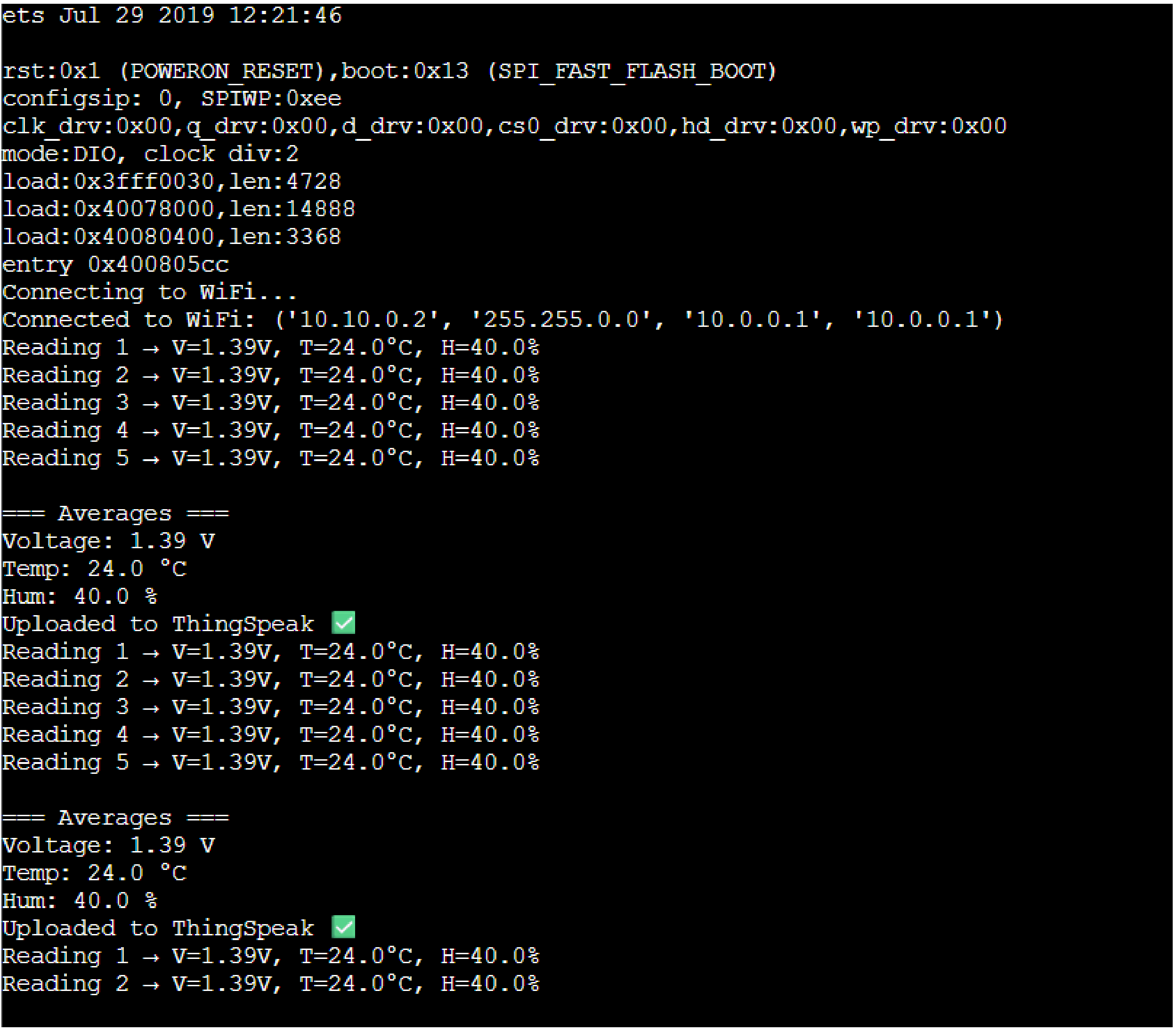

Start
Set up Wi-fi
Api keys
EndLess Loop
Read A0, Temp, Hum
Read & Store values
(5 reads)
Compute Average (V,
Temp, Hum)
Show in Serial Shell &
Upload to ThingSpeak
If Temp
> 30°C
If Hum
> 60%
LED OFF
LED_Temp = ON
LED_Hum = ON
LED OFF

### Source Code :

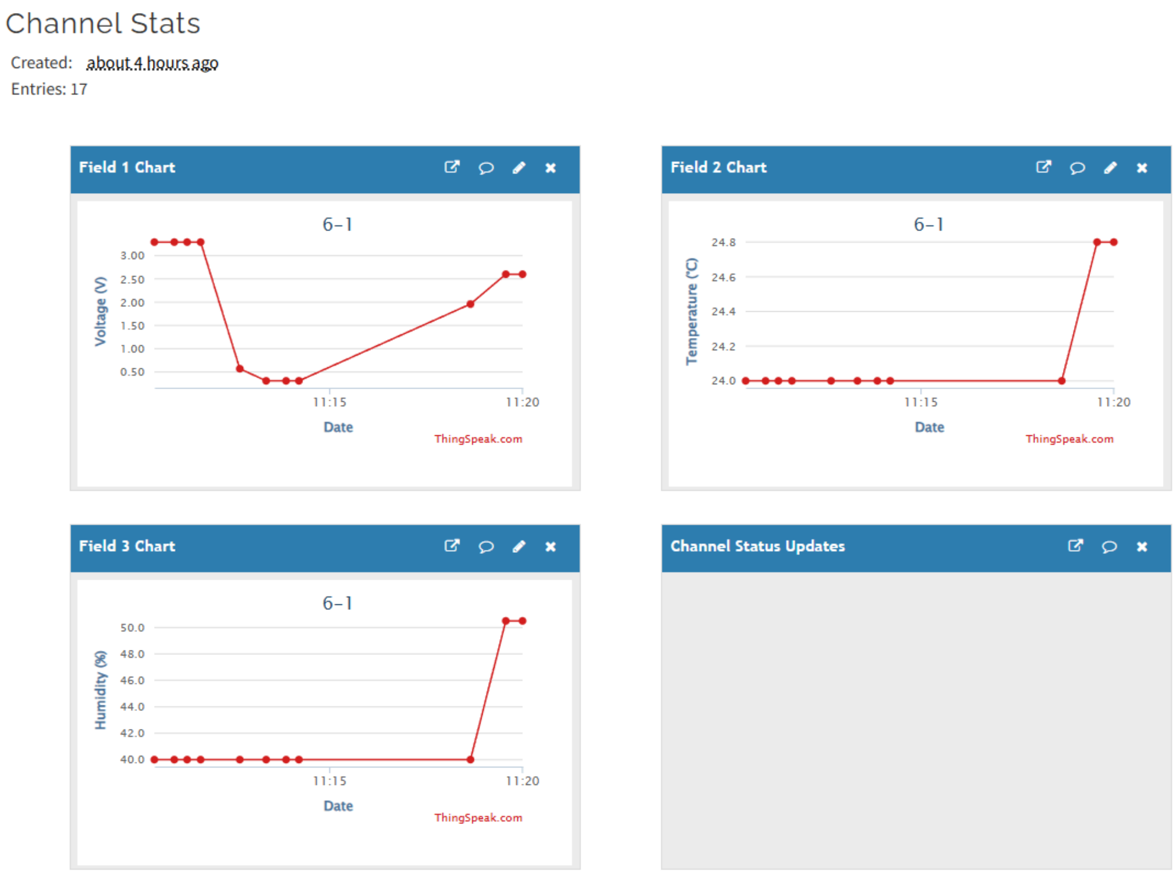
```python
from machine import Pin, ADC
import dht
import network
import time
# --- Pin setup ---
pot = ADC(Pin(34))             # Potentiometer (ADC on GPIO34)
pot.atten(ADC.ATTN_11DB)       # Full range: 0 - 3.3V
d = dht.DHT22(Pin(4))          # DHT22 on GPIO4
led_hum = Pin(5, Pin.OUT)      # Yellow LED
led_temp = Pin(2, Pin.OUT)     # Red LED (safe pin, not GPIO6!)
# --- WiFi setup ---
SSID = "Wokwi-GUEST"
PASSWORD = ""
print("Connecting to WiFi...")
wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect(SSID, PASSWORD)
while not wlan.isconnected():
    time.sleep(0.5)
print("Connected to WiFi:", wlan.ifconfig())
# --- ThingSpeak setup ---
THINGSPEAK_WRITE_API_KEY = "YOUR_API_KEY"
THINGSPEAK_CHANNEL_URL = "https://api.thingspeak.com/update"
# --- Data storage ---
voltages = []
temps = []
hums = []
# --- Memory-safe ThingSpeak upload function ---
def upload_data(v, t, h):
    try:
        import urequests
        url = f"{THINGSPEAK_CHANNEL_URL}?api_key={THINGSPEAK_WRITE_API_KEY}&field1={v:.2f}&field2={t:.1f}&field3={h:.1f}"
        res = urequests.get(url)
        res.close()  # free memory immediately
        print("Uploaded to ThingSpeak ")
    except Exception as e:
        print("Upload failed:", e)
```

```python
# --- Main loop ---
while True:
    # 1. Read potentiometer voltage (0–3.3V)
    adc_value = pot.read()
    voltage = (adc_value / 4095) * 3.3
    # 2. Read DHT22
    try:
        d.measure()
        temp = d.temperature()
        hum = d.humidity()
    except Exception as e:
        print("DHT22 error:", e)
        time.sleep(3)
        continue
    # 3. Save values
    voltages.append(voltage)
    temps.append(temp)
    hums.append(hum)
    print(f"Reading {len(voltages)} → V={voltage:.2f}V, T={temp:.1f}°C, H={hum:.1f}%")
    # 4. If 5 readings collected → average & upload
    if len(voltages) >= 5:
        avgV = sum(voltages) / len(voltages)
        avgT = sum(temps) / len(temps)
        avgH = sum(hums) / len(hums)
        print("\n=== Averages ===")
        print(f"Voltage: {avgV:.2f} V")
        print(f"Temp: {avgT:.1f} °C")
        print(f"Hum: {avgH:.1f} %")
        # Upload averages
        upload_data(avgV, avgT, avgH)
        # Clear lists for next cycle
        voltages.clear()
        temps.clear()
        hums.clear()
    # 5. LED control
    led_temp.value(1 if temp > 30 else 0)
    led_hum.value(1 if hum > 60 else 0)
    # 6. Wait 3 sec before next reading
    time.sleep(3)
```

**Test Records :** (The following test records from wokwi and the thingspeak)



```
ets Jul 29 2019 12:21:46

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0030,len:4728
load:0x40078000,len:14888
load:0x40080400,len:3368
entry 0x400805cc
Connecting to WiFi...
Connected to WiFi: ('10.10.0.2', '255.255.0.0', '10.0.0.1', '10.0.0.1')
Reading 1 → V=1.39V, T=24.0°C, H=40.0%
Reading 2 → V=1.39V, T=24.0°C, H=40.0%
Reading 3 → V=1.39V, T=24.0°C, H=40.0%
Reading 4 → V=1.39V, T=24.0°C, H=40.0%
Reading 5 → V=1.39V, T=24.0°C, H=40.0%

=== Averages ===
Voltage: 1.39 V
Temp: 24.0 °C
Hum: 40.0 %
Uploaded to ThingSpeak ✅
Reading 1 → V=1.39V, T=24.0°C, H=40.0%
Reading 2 → V=1.39V, T=24.0°C, H=40.0%
Reading 3 → V=1.39V, T=24.0°C, H=40.0%
Reading 4 → V=1.39V, T=24.0°C, H=40.0%
Reading 5 → V=1.39V, T=24.0°C, H=40.0%

=== Averages ===
Voltage: 1.39 V
Temp: 24.0 °C
Hum: 40.0 %
Uploaded to ThingSpeak ✅
Reading 1 → V=1.39V, T=24.0°C, H=40.0%
Reading 2 → V=1.39V, T=24.0°C, H=40.0%
```



Channel Stats

Created: about 4 hours ago

Entries: 17

**Note :** This was done using DHT22 in wowki sim only

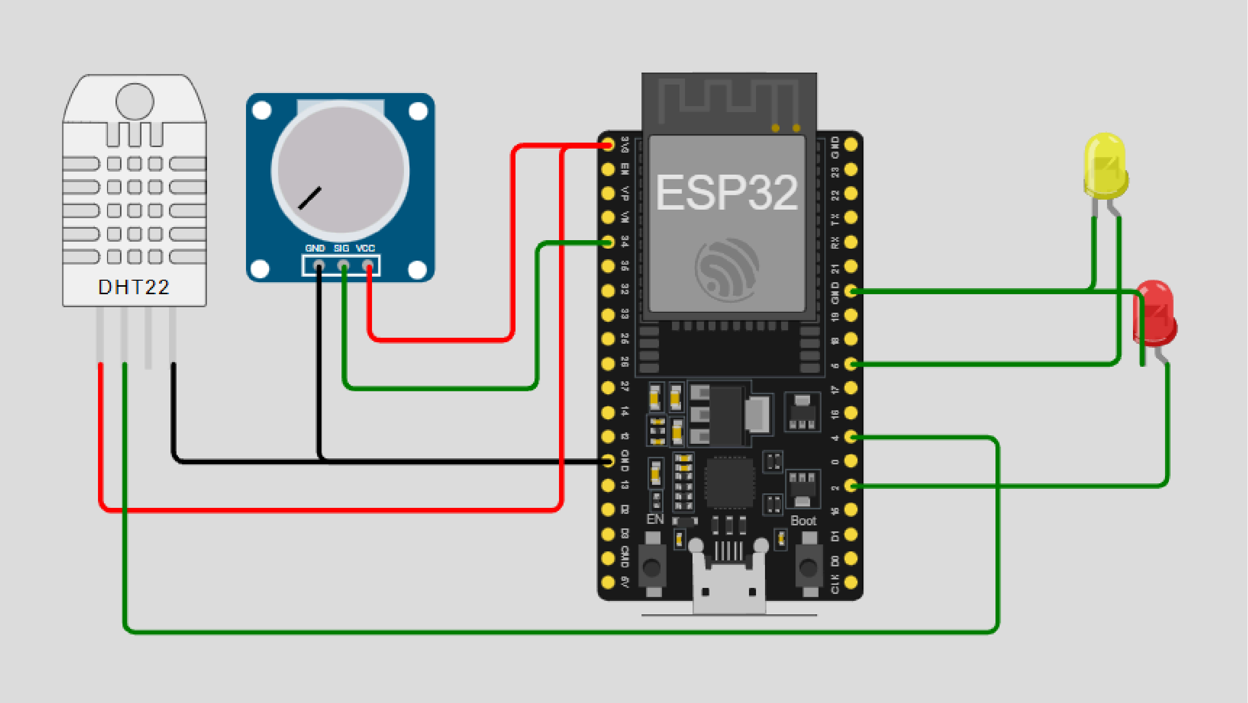
**Lab2 – On ESP32 , complete the following tasks:**

Read the voltage value on A0, DHT11 Temperature/humidity data and upload it to ThingSpeak:

1. Connect the potentiometer to the circuit consisting of the ADC to GPIO0, DHT11 to GPIOxxx, yellow LED (LED_Hum) to GPIOxxx, and red LED (LED_Temp) to GPIOxxx. Complete the following steps:
   - Function 1: When the program starts, it detects the voltage on ADC every 3 seconds (limited to 0~3.3V), obtains the temperature and humidity data of DHT11, and displays the average value in Shell after obtaining 5 records, and uploads it to ThingSpeak.
   - Function 2:When the temperature is above 30 degrees, LED_Temp lights up.
   - Function 3:When the humidity is higher than 60%, LED_Hum lights up.

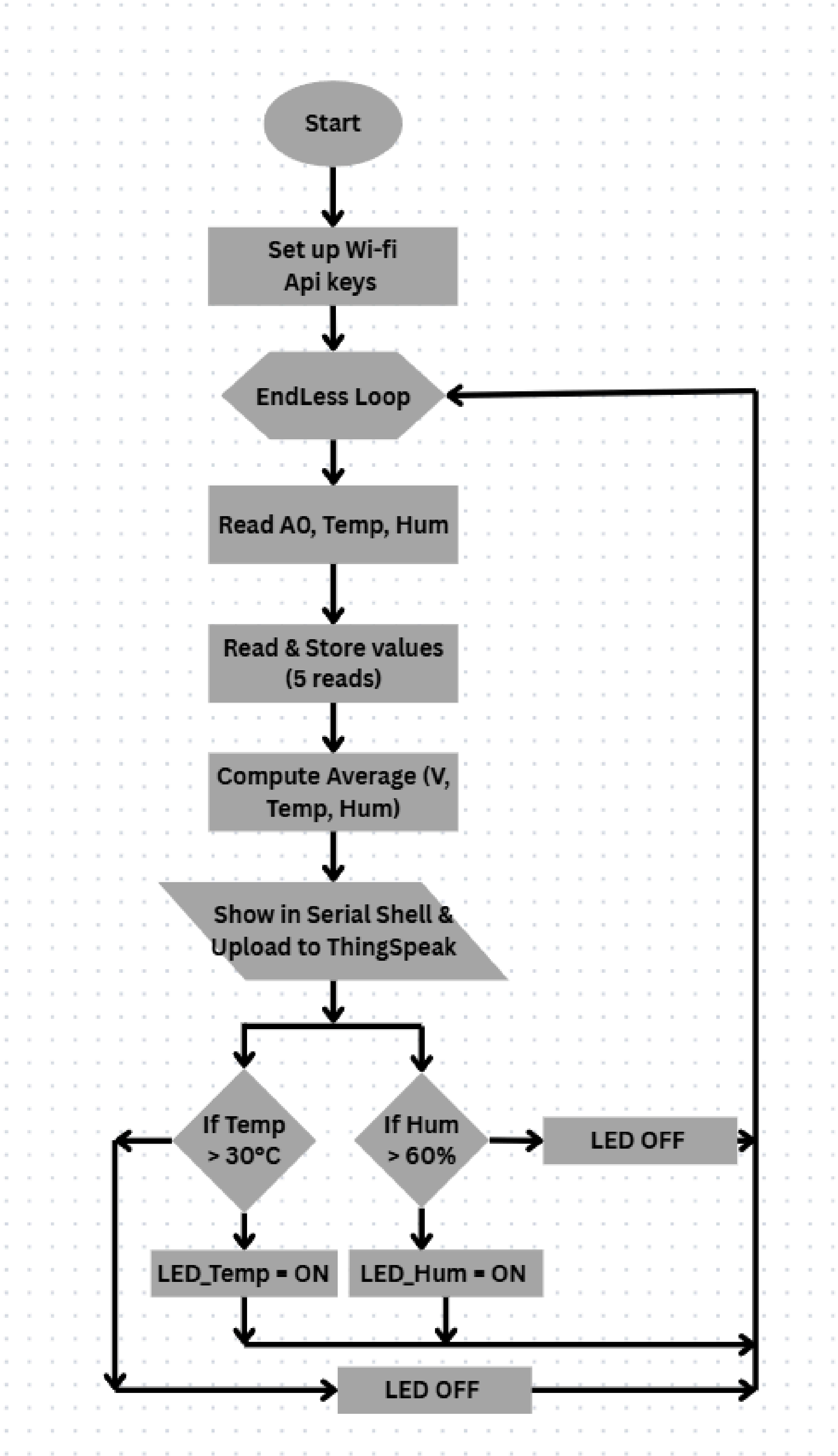
**Connections :**

- Potentiometer → ADC pin GPIO34
- DHT11 → GPIO4
- Yellow LED (LED_Hum) → GPIO5
- Red LED (LED_Temp) → GPIO2



(The following Circuit uses DHT22 in the wokwi instead of DHT11)

**FlowChart :**

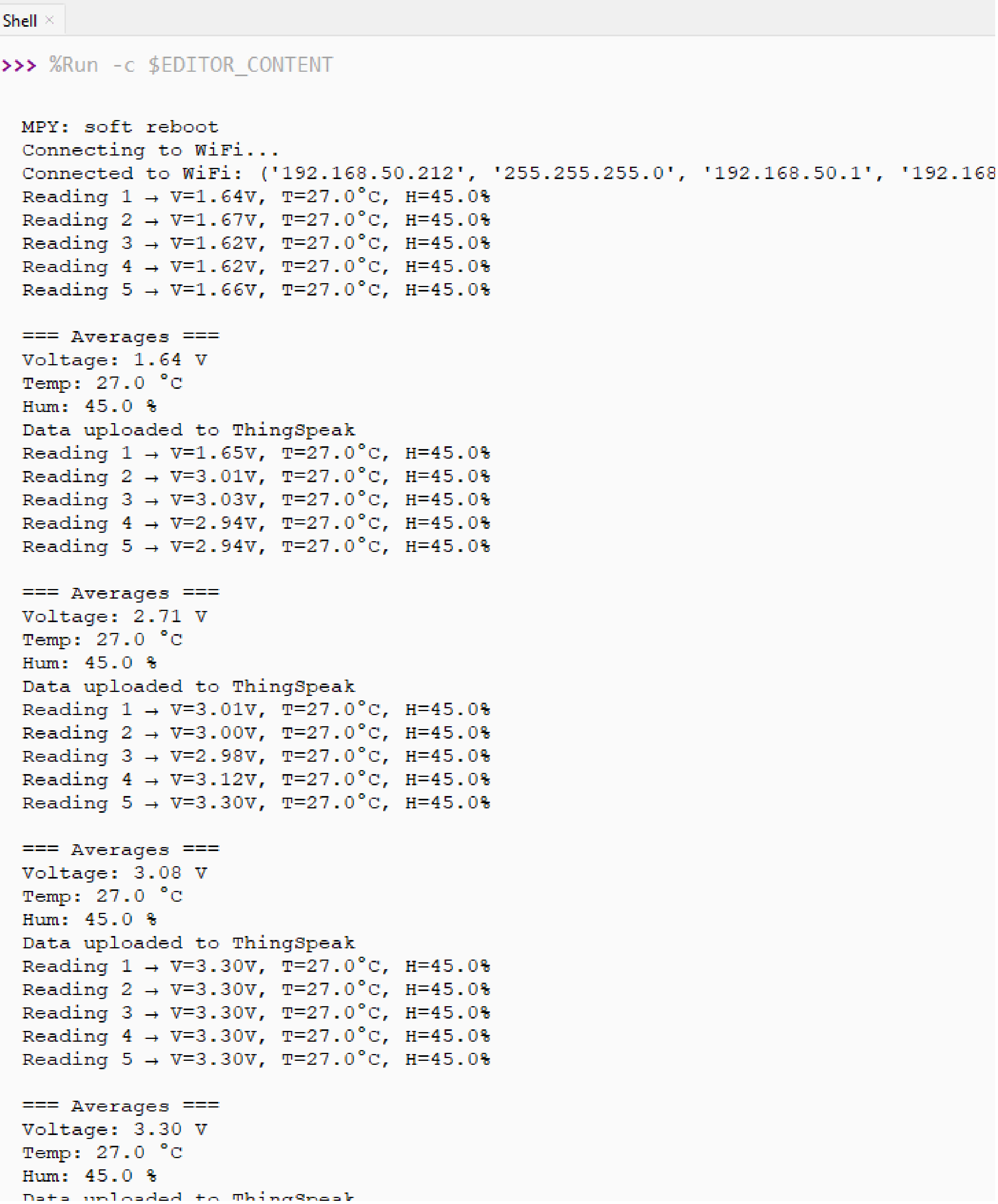

Start
Set up Wi-fi
Api keys
EndLess Loop
Read A0, Temp, Hum
Read & Store values
(5 reads)
Compute Average (V,
Temp, Hum)
Show in Serial Shell &
Upload to ThingSpeak
If Temp
> 30°C
If Hum
> 60%
LED OFF
LED_Temp = ON
LED_Hum = ON
LED OFF

**Source Code :**

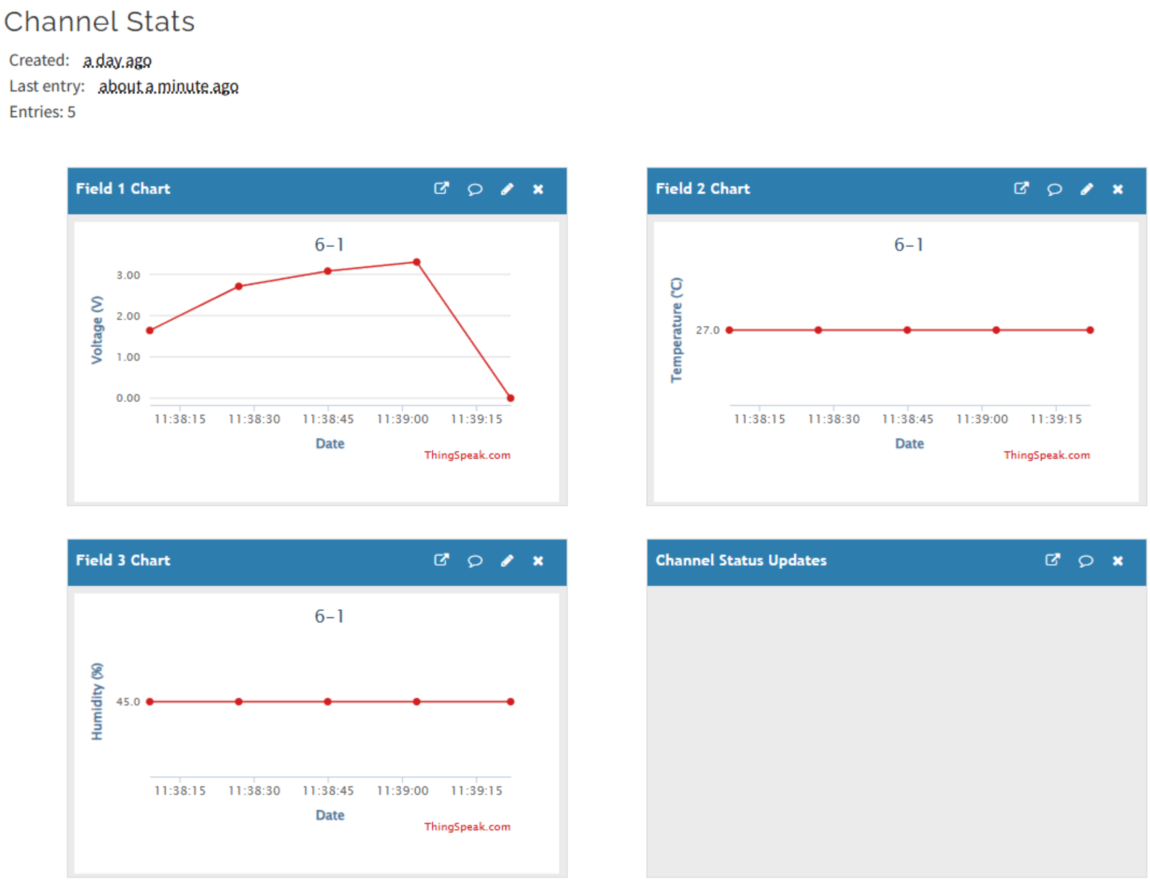
```
from machine import Pin, ADC
import dht
import network
import time
# --- Pin setup ---
pot = ADC(Pin(34))             # Potentiometer (ADC on GPIO34)
pot.atten(ADC.ATTN_11DB)       # Full range: 0 - 3.3V
d = dht.DHT11(Pin(4))          # DHT11 on GPIO4
led_hum = Pin(5, Pin.OUT)      # Yellow LED
led_temp = Pin(2, Pin.OUT)     # Red LED (safe pin, not GPIO6!)
# --- WiFi setup ---
SSID = "ISILAB CR"      #You can change the wifi details
PASSWORD = "isilab.ncut.CR"
print("Connecting to WiFi...")
wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect(SSID, PASSWORD)
while not wlan.isconnected():
    time.sleep(0.5)
print("Connected to WiFi:", wlan.ifconfig())
# --- ThingSpeak setup ---
THINGSPEAK_WRITE_API_KEY = "YOUR_API_KEY"  #give your Api key
THINGSPEAK_CHANNEL_URL = "https://api.thingspeak.com/update"
# --- Data storage ---
voltages = []
temps = []
hums = []
# --- Memory-safe ThingSpeak upload function ---
def upload_data(v, t, h):
    try:
        import urequests
        url =
f"{THINGSPEAK_CHANNEL_URL}?api_key={THINGSPEAK_WRITE_API_KEY}&field1={v:.2
f}&field2={t:.1f}&field3={h:.1f}"
        res = urequests.get(url)
        res.close()  # free memory immediately
        print("Uploaded to ThingSpeak ")
    except Exception as e:
        print("Upload failed:", e)
```

```python
# --- Main loop ---
while True:
    # 1. Read potentiometer voltage (0–3.3V)
    adc_value = pot.read()
    voltage = (adc_value / 4095) * 3.3
    # 2. Read DHT22
    try:
        d.measure()
        temp = d.temperature()
        hum = d.humidity()
    except Exception as e:
        print("DHT22 error:", e)
        time.sleep(3)
        continue
    # 3. Save values
    voltages.append(voltage)
    temps.append(temp)
    hums.append(hum)
    print(f"Reading {len(voltages)} → V={voltage:.2f}V, T={temp:.1f}°C, H={hum:.1f}%")
    # 4. If 5 readings collected → average & upload
    if len(voltages) >= 5:
        avgV = sum(voltages) / len(voltages)
        avgT = sum(temps) / len(temps)
        avgH = sum(hums) / len(hums)
        print("\n=== Averages ===")
        print(f"Voltage: {avgV:.2f} V")
        print(f"Temp: {avgT:.1f} °C")
        print(f"Hum: {avgH:.1f} %")
        # Upload averages
        upload_data(avgV, avgT, avgH)
        # Clear lists for next cycle
        voltages.clear()
        temps.clear()
        hums.clear()
    # 5. LED control
    led_temp.value(1 if temp > 30 else 0)
    led_hum.value(1 if hum > 60 else 0)
    # 6. Wait 3 sec before next reading
    time.sleep(3)
```

**Test Record :** (The following test records from Thonny and the thingspeak)



```
>>> %Run -c $EDITOR_CONTENT

  MPY: soft reboot
  Connecting to WiFi...
  Connected to WiFi: ('192.168.50.212', '255.255.255.0', '192.168.50.1', '192.168
  Reading 1 → V=1.64V, T=27.0°C, H=45.0%
  Reading 2 → V=1.67V, T=27.0°C, H=45.0%
  Reading 3 → V=1.62V, T=27.0°C, H=45.0%
  Reading 4 → V=1.62V, T=27.0°C, H=45.0%
  Reading 5 → V=1.66V, T=27.0°C, H=45.0%

  === Averages ===
  Voltage: 1.64 V
  Temp: 27.0 °C
  Hum: 45.0 %
  Data uploaded to ThingSpeak
  Reading 1 → V=1.65V, T=27.0°C, H=45.0%
  Reading 2 → V=3.01V, T=27.0°C, H=45.0%
  Reading 3 → V=3.03V, T=27.0°C, H=45.0%
  Reading 4 → V=2.94V, T=27.0°C, H=45.0%
  Reading 5 → V=2.94V, T=27.0°C, H=45.0%

  === Averages ===
  Voltage: 2.71 V
  Temp: 27.0 °C
  Hum: 45.0 %
  Data uploaded to ThingSpeak
  Reading 1 → V=3.01V, T=27.0°C, H=45.0%
  Reading 2 → V=3.00V, T=27.0°C, H=45.0%
  Reading 3 → V=2.98V, T=27.0°C, H=45.0%
  Reading 4 → V=3.12V, T=27.0°C, H=45.0%
  Reading 5 → V=3.30V, T=27.0°C, H=45.0%

  === Averages ===
  Voltage: 3.08 V
  Temp: 27.0 °C
  Hum: 45.0 %
  Data uploaded to ThingSpeak
  Reading 1 → V=3.30V, T=27.0°C, H=45.0%
  Reading 2 → V=3.30V, T=27.0°C, H=45.0%
  Reading 3 → V=3.30V, T=27.0°C, H=45.0%
  Reading 4 → V=3.30V, T=27.0°C, H=45.0%
  Reading 5 → V=3.30V, T=27.0°C, H=45.0%

  === Averages ===
  Voltage: 3.30 V
  Temp: 27.0 °C
  Hum: 45.0 %
  Data uploaded to ThingSpeak
```



Channel Stats

Created: a day ago

Last entry: about a minute ago

Entries: 5

**Note :** This task was done practically  by using DHT11 in Thonny s/w

**Lab3 – On Wokwi, complete the following tasks:**

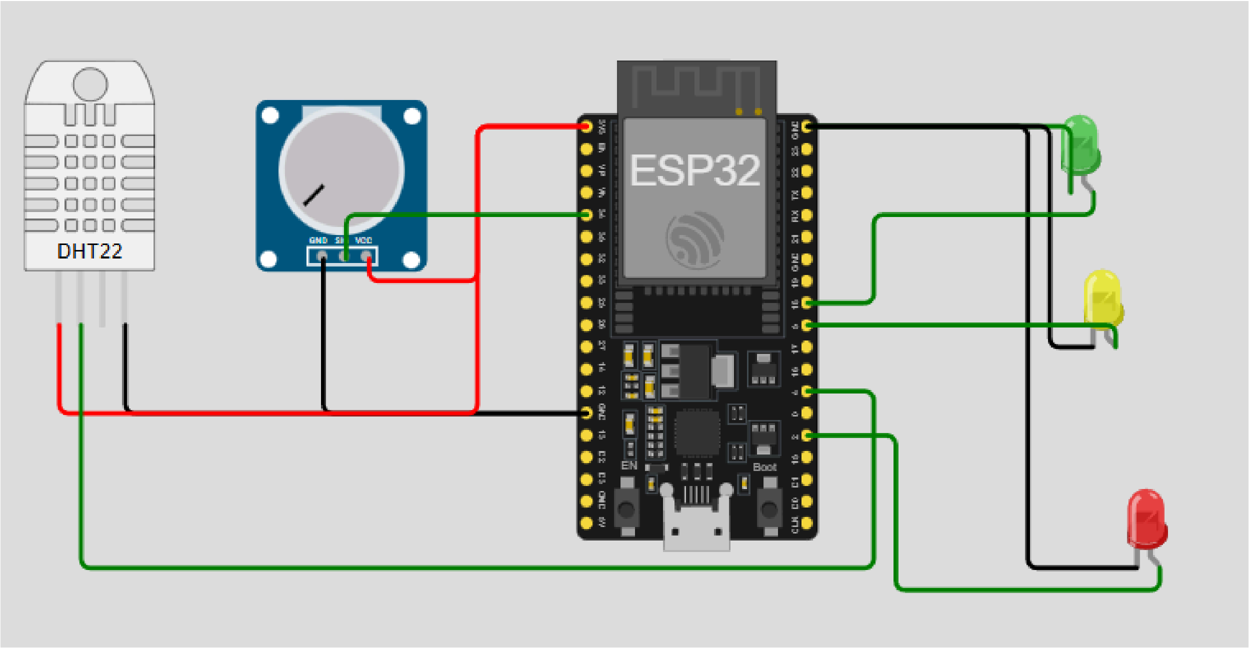
1. Create a circuit for the green LED (LED_ADC), yellow LED (LED_Hum), and red LED (LED_Temp) on the Wokwi. Complete the following steps:

When the program is started, it captures the ADC (Field 1), DHT22 temperature data (Field 2), and DHT22 humidity data (Field 3) on ThingSpeak every 3 seconds and displays them in the shell.

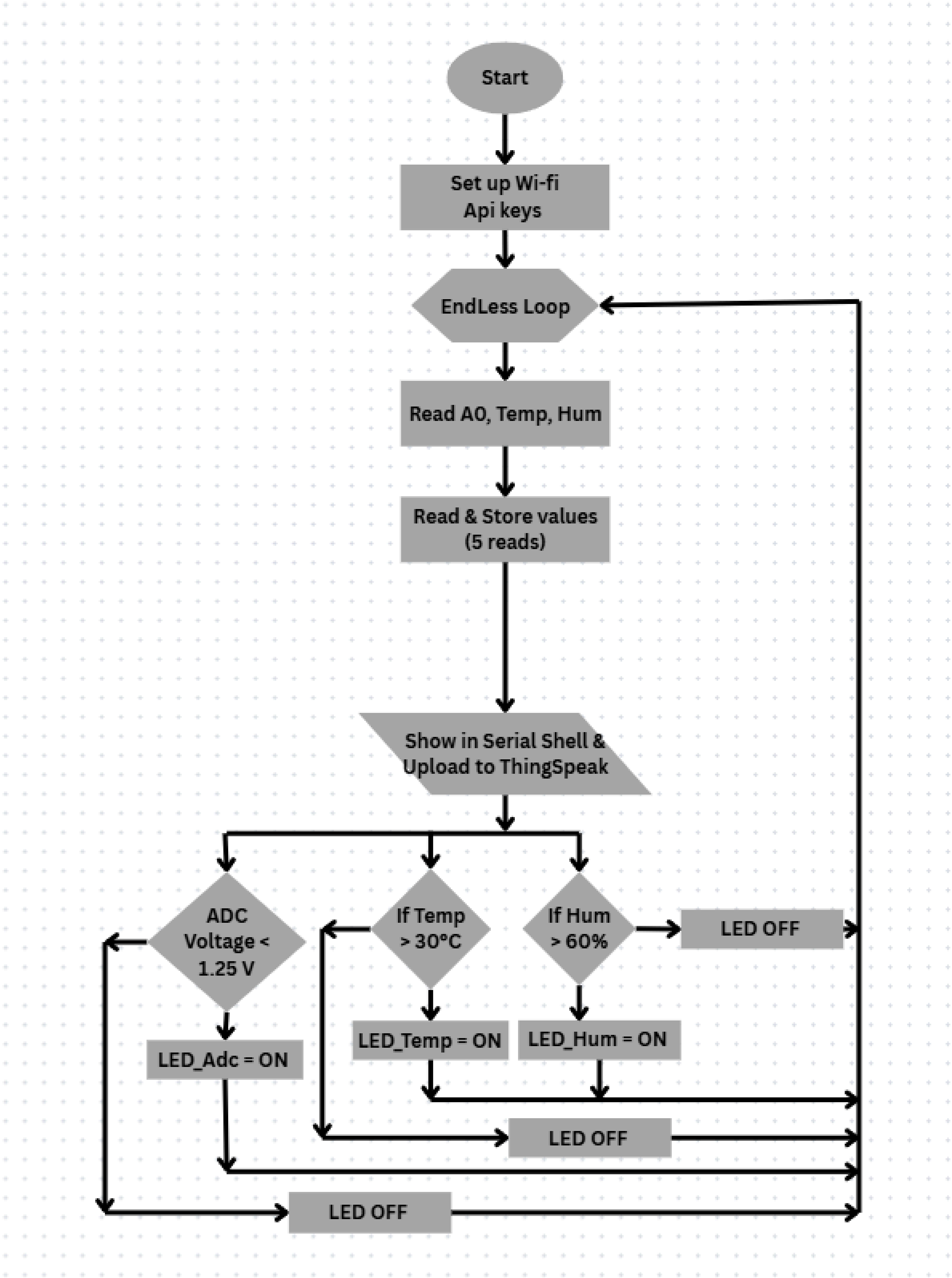
- Function 1:When the ADC voltage is lower than 1.25V, LED_ADC lights up.
- Function 2:When the temperature is above 30 degrees, LED_Temp lights .up
- Function 3:When the humidity is higher than 60%, LED_Hum lights up.

**Connection :**

- Potentiometer → GPIO34 (ADC pin)
- DHT22 → GPIO4
- Green LED (LED_ADC) → GPIO18
- Yellow LED (LED_Hum) → GPIO5
- Red LED (LED_Temp) → GPIO2



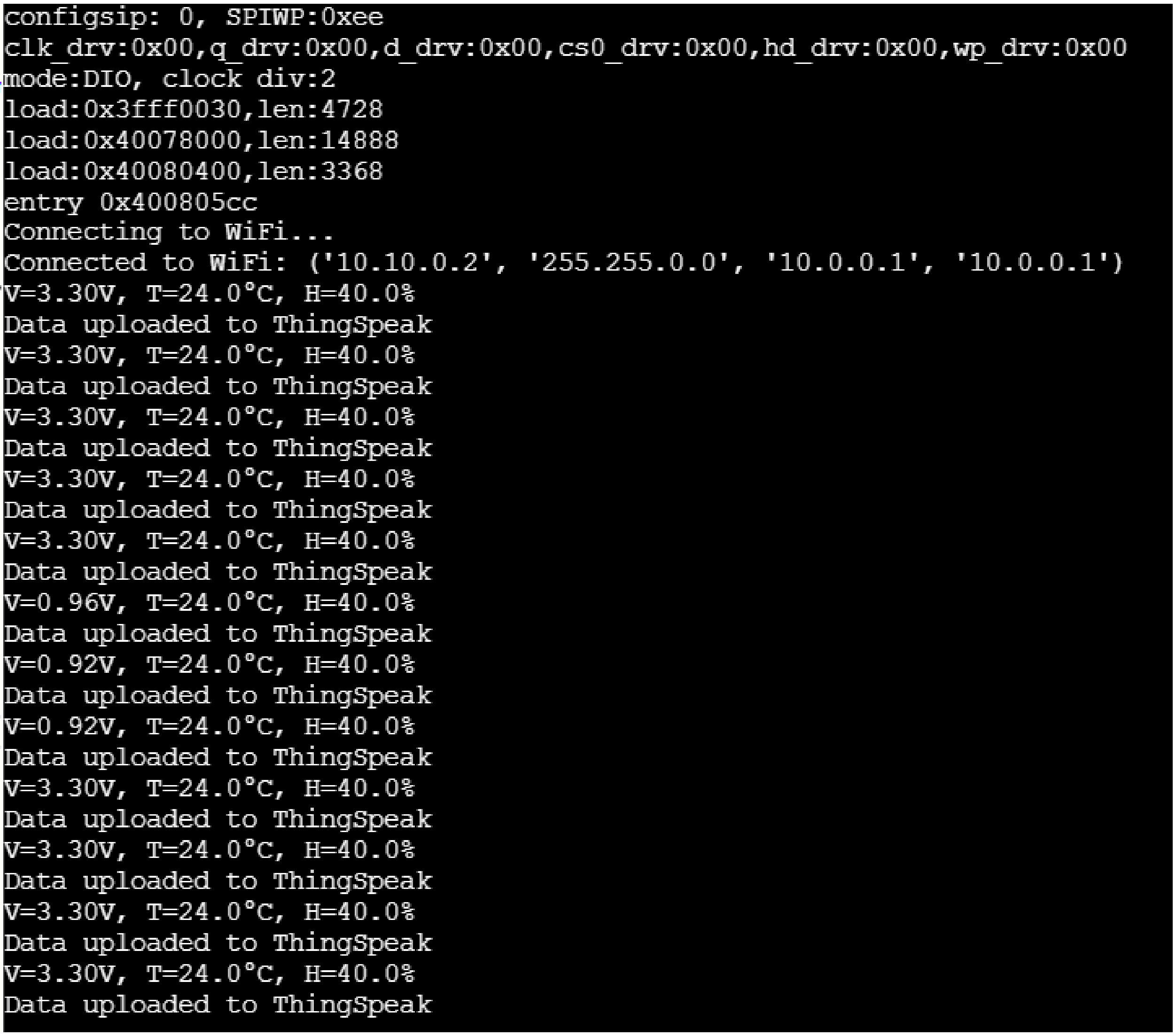
**Flow Chart :**


Start
Set up Wi-fi
Api keys
EndLess Loop
Read A0, Temp, Hum
Read & Store values
(5 reads)
Show in Serial Shell &
Upload to ThingSpeak
ADC
Voltage <
1.25 V
If Temp
> 30°C
If Hum
> 60%
LED OFF
LED_Adc = ON
LED_Temp = ON
LED_Hum = ON
LED OFF
LED OFF

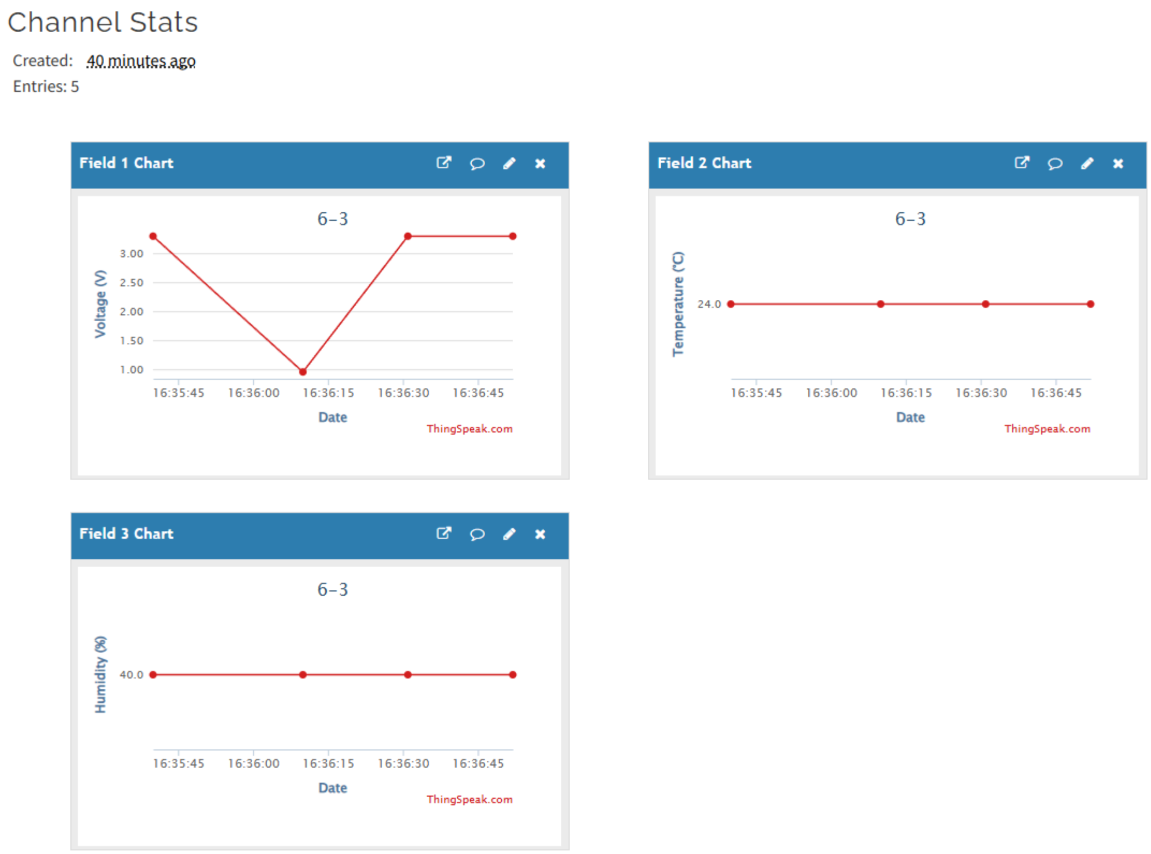
**Source Code :**

```
from machine import Pin, ADC
import dht
import network
import time

# --- Pin setup ---
pot = ADC(Pin(34))              # Potentiometer ADC
pot.atten(ADC.ATTN_11DB)        # Range: 0-3.3V
d = dht.DHT22(Pin(4))           # DHT22 on GPIO4

led_adc = Pin(18, Pin.OUT)      # Green LED
led_hum = Pin(5, Pin.OUT)       # Yellow LED
led_temp = Pin(2, Pin.OUT)      # Red LED

# --- WiFi setup ---
SSID = "Wokwi-GUEST"
PASSWORD = ""

print("Connecting to WiFi...")
wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect(SSID, PASSWORD)

while not wlan.isconnected():
    time.sleep(0.5)
print("Connected to WiFi:", wlan.ifconfig())

# --- ThingSpeak setup ---
THINGSPEAK_WRITE_API_KEY = "MIDF8JA8TV1R6Y1C"
THINGSPEAK_CHANNEL_URL =
"https://thingspeak.mathworks.com/channels/3056080"

# --- Upload function ---
def upload_data(v, t, h):
    import urequests
    url =
"https://api.thingspeak.com/update?api_key={}&field1={:.2f}&field2={:.1f}&
field3={:.1f}".format(
        THINGSPEAK_WRITE_API_KEY, v, t, h
    )
    try:
        res = urequests.get(url)   # Send request
        res.close()                # Free memory immediately
        print("Data uploaded to ThingSpeak ")
    except Exception as e:
        print("Upload failed:", e)


# --- Main loop ---
while True:
    # 1. Read ADC voltage
    adc_value = pot.read()
    voltage = (adc_value / 4095) * 3.3

    # 2. Read DHT22
    try:
        d.measure()
        temp = d.temperature()
        hum = d.humidity()
    except Exception as e:
        print("DHT22 error:", e)
        time.sleep(3)
        continue

    # 3. Display readings
    print(f"V={voltage:.2f}V, T={temp:.1f}°C, H={hum:.1f}%")

    # 4. Upload to ThingSpeak
    upload_data(voltage, temp, hum)

    # 5. LED control
    led_adc.value(1 if voltage < 1.25 else 0)   # Green LED
    led_temp.value(1 if temp > 30 else 0)       # Red LED
    led_hum.value(1 if hum > 60 else 0)         # Yellow LED

    # 6. Wait 3 seconds
    time.sleep(3)
```

**Test Records :** (The following test records from Wokwi and ThingSpeak)



```
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0030,len:4728
load:0x40078000,len:14888
load:0x40080400,len:3368
entry 0x400805cc
Connecting to WiFi...
Connected to WiFi: ('10.10.0.2', '255.255.0.0', '10.0.0.1', '10.0.0.1')
V=3.30V, T=24.0°C, H=40.0%
Data uploaded to ThingSpeak
V=3.30V, T=24.0°C, H=40.0%
Data uploaded to ThingSpeak
V=3.30V, T=24.0°C, H=40.0%
Data uploaded to ThingSpeak
V=3.30V, T=24.0°C, H=40.0%
Data uploaded to ThingSpeak
V=3.30V, T=24.0°C, H=40.0%
Data uploaded to ThingSpeak
V=0.96V, T=24.0°C, H=40.0%
Data uploaded to ThingSpeak
V=0.92V, T=24.0°C, H=40.0%
Data uploaded to ThingSpeak
V=0.92V, T=24.0°C, H=40.0%
Data uploaded to ThingSpeak
V=3.30V, T=24.0°C, H=40.0%
Data uploaded to ThingSpeak
V=3.30V, T=24.0°C, H=40.0%
Data uploaded to ThingSpeak
V=3.30V, T=24.0°C, H=40.0%
Data uploaded to ThingSpeak
V=3.30V, T=24.0°C, H=40.0%
Data uploaded to ThingSpeak
```



Channel Stats

Created: 40 minutes ago

Entries: 5

**Note :** This was done using DHT22 in wowki sim only

**Lab4– On ESP32, complete the following tasks:**

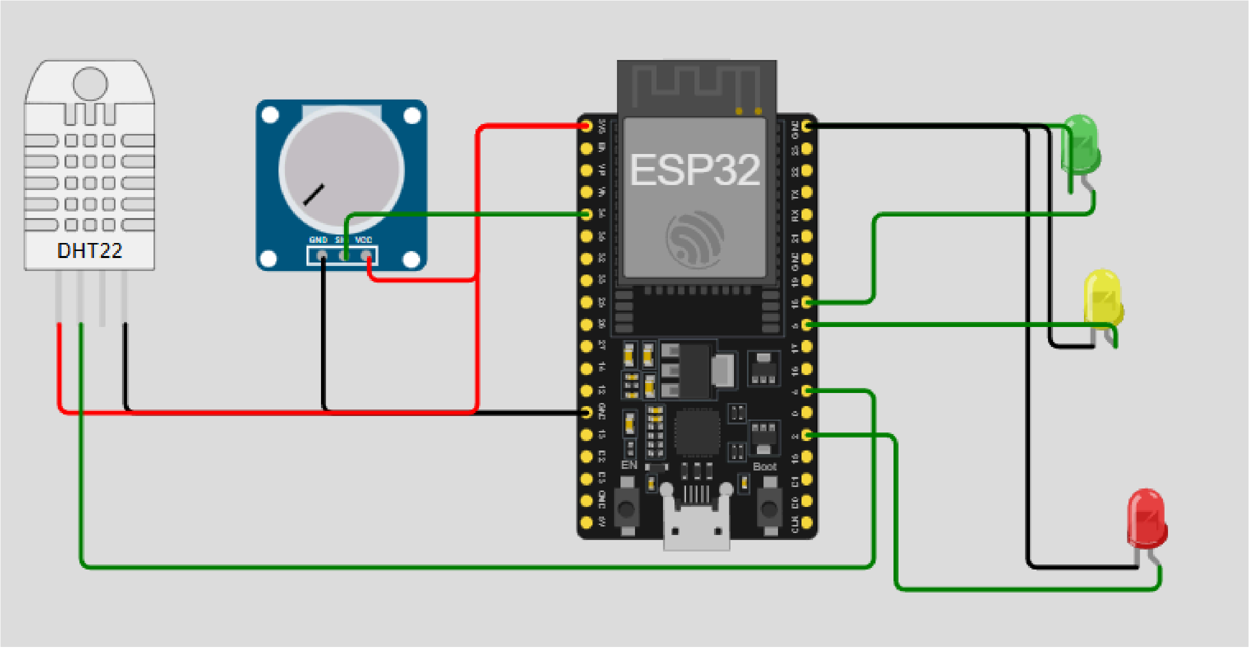
1. Create a circuit for the green LED (LED_ADC), yellow LED (LED_Hum), and red LED (LED_Temp) on the ESP32. Complete the following steps:

   When the program is started, it captures the ADC (Field 1), DHT11 temperature data (Field 2), and DHT11 humidity data (Field 3) on ThingSpeak every 3 seconds and displays them in the shell.

   - Function 1:When the ADC voltage is lower than 1.25V, LED_ADC lights up.
   - Function 2:When the temperature is above 30 degrees, LED_Temp lights up.
   - Function 3:When the humidity is higher than 60%, LED_Hum lights up.

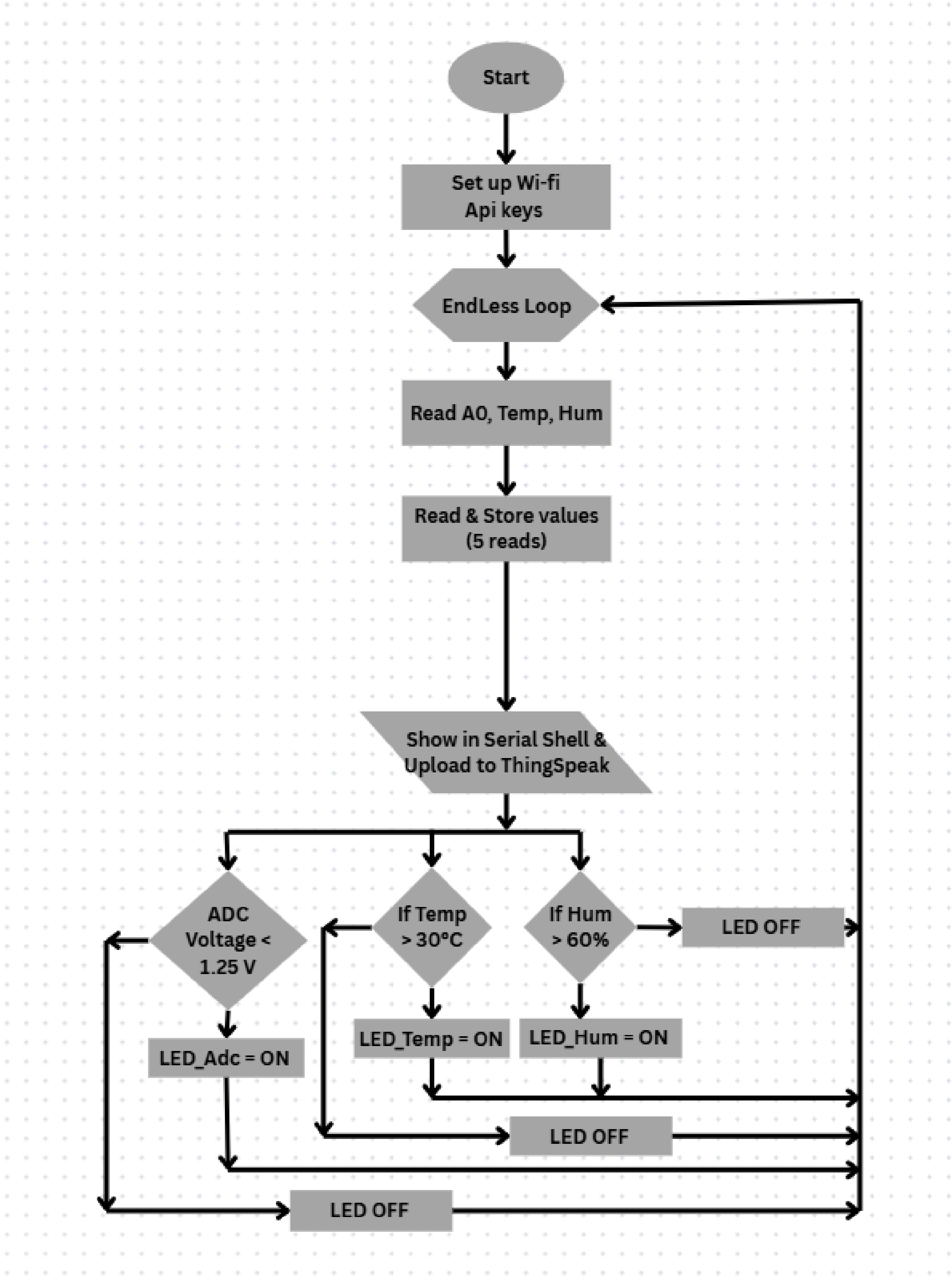
**Connections :**

- Potentiometer → GPIO34 (ADC pin)
- DHT11 → GPIO4
- Green LED (LED_ADC) → GPIO18
- Yellow LED (LED_Hum) → GPIO5
- Red LED (LED_Temp) → GPIO2



The following circuit is blueprint only in practically i use DHT11 instead of DHT22

## Flow Chart :

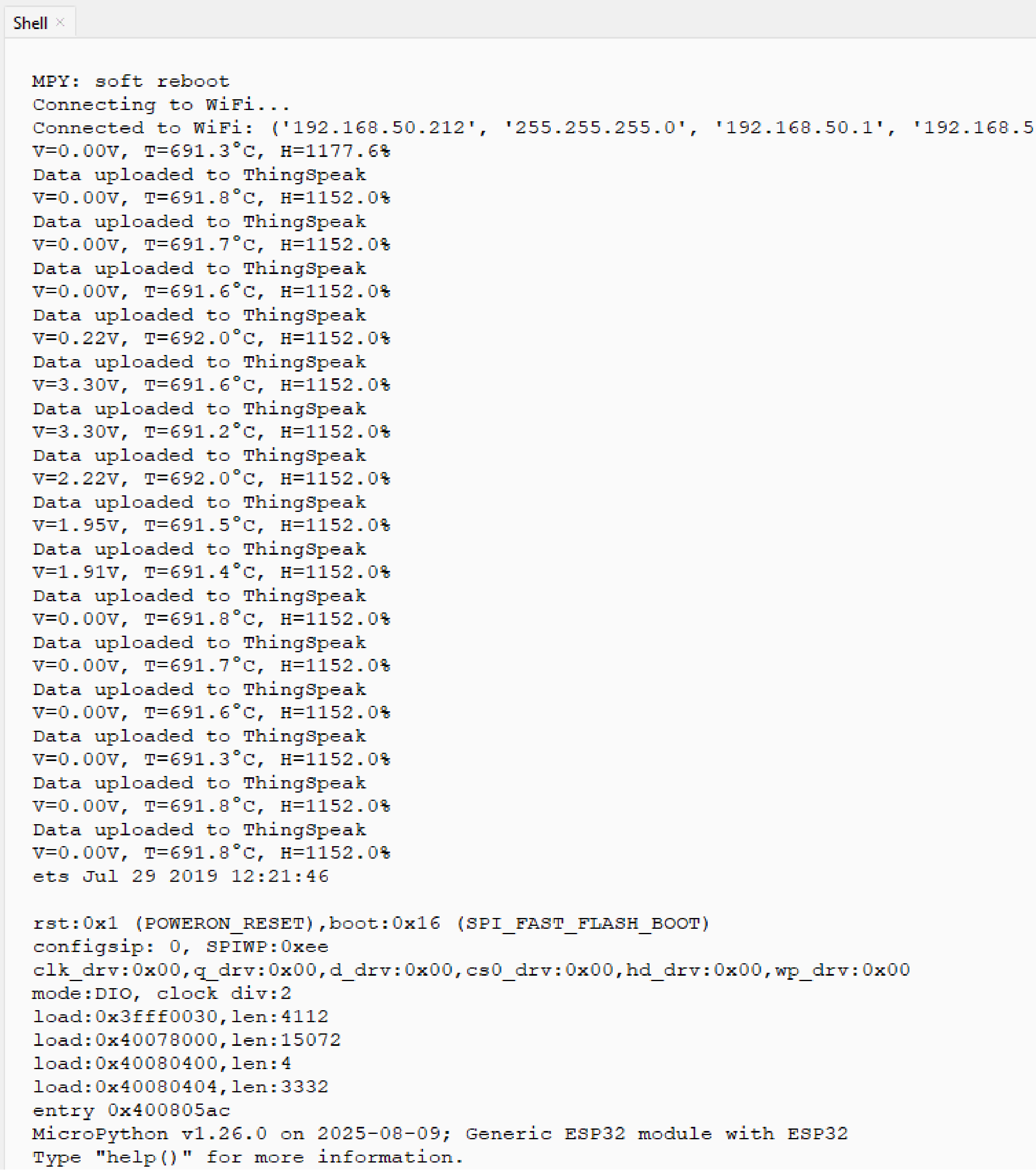


Start
Set up Wi-fi
Api keys
EndLess Loop
Read A0, Temp, Hum
Read & Store values
(5 reads)
Show in Serial Shell &
Upload to ThingSpeak
ADC
Voltage <
1.25 V
If Temp
> 30°C
If Hum
> 60%
LED OFF
LED_Adc = ON
LED_Temp = ON
LED_Hum = ON
LED OFF
LED OFF

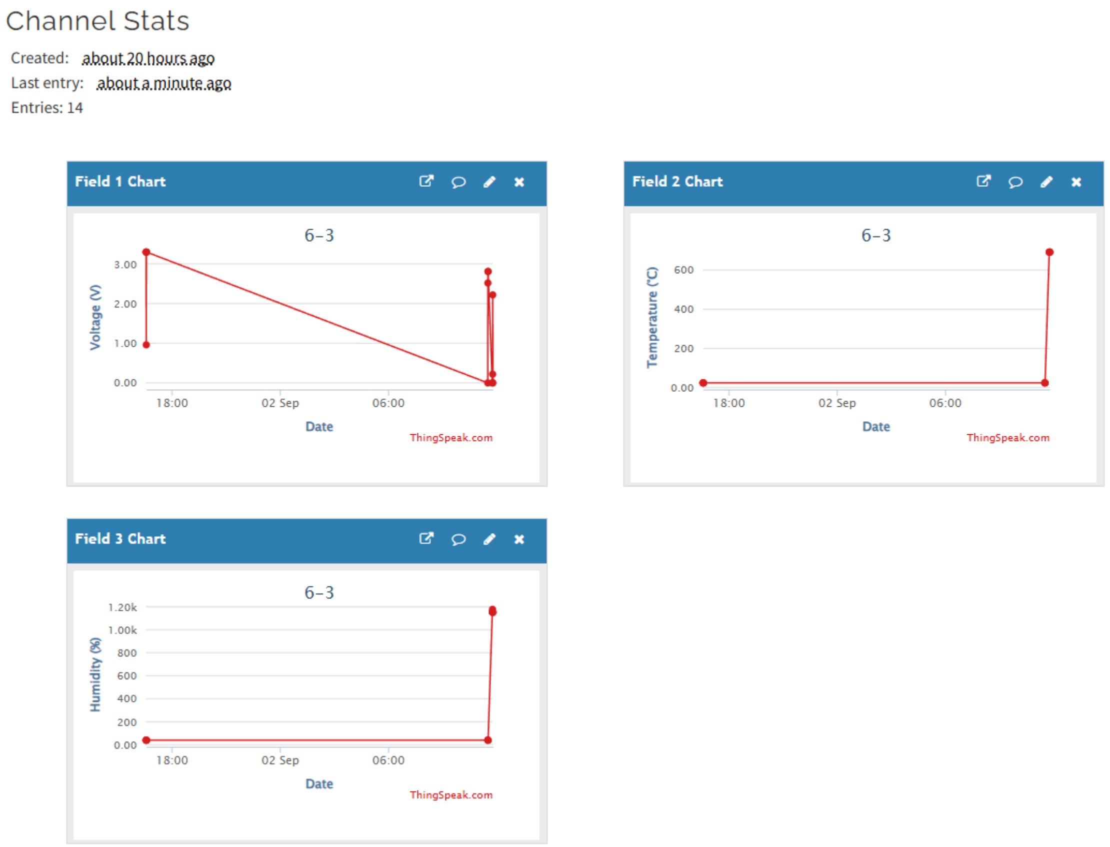
**Source Code :**

```
from machine import Pin, ADC
import dht
import network
import time

# --- Pin setup ---
pot = ADC(Pin(34))              # Potentiometer ADC
pot.atten(ADC.ATTN_11DB)        # Range: 0-3.3V
d = dht.DHT22(Pin(4))           # DHT22 on GPIO4

led_adc = Pin(18, Pin.OUT)      # Green LED
led_hum = Pin(5, Pin.OUT)       # Yellow LED
led_temp = Pin(2, Pin.OUT)      # Red LED

# --- WiFi setup ---
SSID = "Wokwi-GUEST"
PASSWORD = ""

print("Connecting to WiFi...")
wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect(SSID, PASSWORD)

while not wlan.isconnected():
    time.sleep(0.5)
print("Connected to WiFi:", wlan.ifconfig())

# --- ThingSpeak setup ---
THINGSPEAK_WRITE_API_KEY = "MIDF8JA8TV1R6Y1C"
THINGSPEAK_CHANNEL_URL =
"https://thingspeak.mathworks.com/channels/3056080"

# --- Upload function ---
def upload_data(v, t, h):
    import urequests
    url =
"https://api.thingspeak.com/update?api_key={}&field1={:.2f}&field2={:.1f}&
field3={:.1f}".format(
        THINGSPEAK_WRITE_API_KEY, v, t, h
    )
    try:
        res = urequests.get(url)    # Send request
        res.close()                 # Free memory immediately
        print("Data uploaded to ThingSpeak ")
    except Exception as e:
        print("Upload failed:", e)


# --- Main loop ---
while True:
    # 1. Read ADC voltage
    adc_value = pot.read()
    voltage = (adc_value / 4095) * 3.3

    # 2. Read DHT22
    try:
        d.measure()
        temp = d.temperature()
        hum = d.humidity()
    except Exception as e:
        print("DHT22 error:", e)
        time.sleep(3)
        continue

    # 3. Display readings
    print(f"V={voltage:.2f}V, T={temp:.1f}°C, H={hum:.1f}%")

    # 4. Upload to ThingSpeak
    upload_data(voltage, temp, hum)

    # 5. LED control
    led_adc.value(1 if voltage < 1.25 else 0)   # Green LED
    led_temp.value(1 if temp > 30 else 0)       # Red LED
    led_hum.value(1 if hum > 60 else 0)         # Yellow LED

    # 6. Wait 3 seconds
    time.sleep(3)
```

## Test Records :



```
MPY: soft reboot
Connecting to WiFi...
Connected to WiFi: ('192.168.50.212', '255.255.255.0', '192.168.50.1', '192.168.5
V=0.00V, T=691.3°C, H=1177.6%
Data uploaded to ThingSpeak
V=0.00V, T=691.8°C, H=1152.0%
Data uploaded to ThingSpeak
V=0.00V, T=691.7°C, H=1152.0%
Data uploaded to ThingSpeak
V=0.00V, T=691.6°C, H=1152.0%
Data uploaded to ThingSpeak
V=0.22V, T=692.0°C, H=1152.0%
Data uploaded to ThingSpeak
V=3.30V, T=691.6°C, H=1152.0%
Data uploaded to ThingSpeak
V=3.30V, T=691.2°C, H=1152.0%
Data uploaded to ThingSpeak
V=2.22V, T=692.0°C, H=1152.0%
Data uploaded to ThingSpeak
V=1.95V, T=691.5°C, H=1152.0%
Data uploaded to ThingSpeak
V=1.91V, T=691.4°C, H=1152.0%
Data uploaded to ThingSpeak
V=0.00V, T=691.8°C, H=1152.0%
Data uploaded to ThingSpeak
V=0.00V, T=691.7°C, H=1152.0%
Data uploaded to ThingSpeak
V=0.00V, T=691.6°C, H=1152.0%
Data uploaded to ThingSpeak
V=0.00V, T=691.3°C, H=1152.0%
Data uploaded to ThingSpeak
V=0.00V, T=691.8°C, H=1152.0%
Data uploaded to ThingSpeak
V=0.00V, T=691.8°C, H=1152.0%
ets Jul 29 2019 12:21:46

rst:0x1 (POWERON_RESET),boot:0x16 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0030,len:4112
load:0x40078000,len:15072
load:0x40080400,len:4
load:0x40080404,len:3332
entry 0x400805ac
MicroPython v1.26.0 on 2025-08-09; Generic ESP32 module with ESP32
Type "help()" for more information.
```



Channel Stats

Created: about 20 hours ago

Last entry: about a minute ago

Entries: 14

**Note :** This task was done practically  by using DHT11 in Thonny s/w

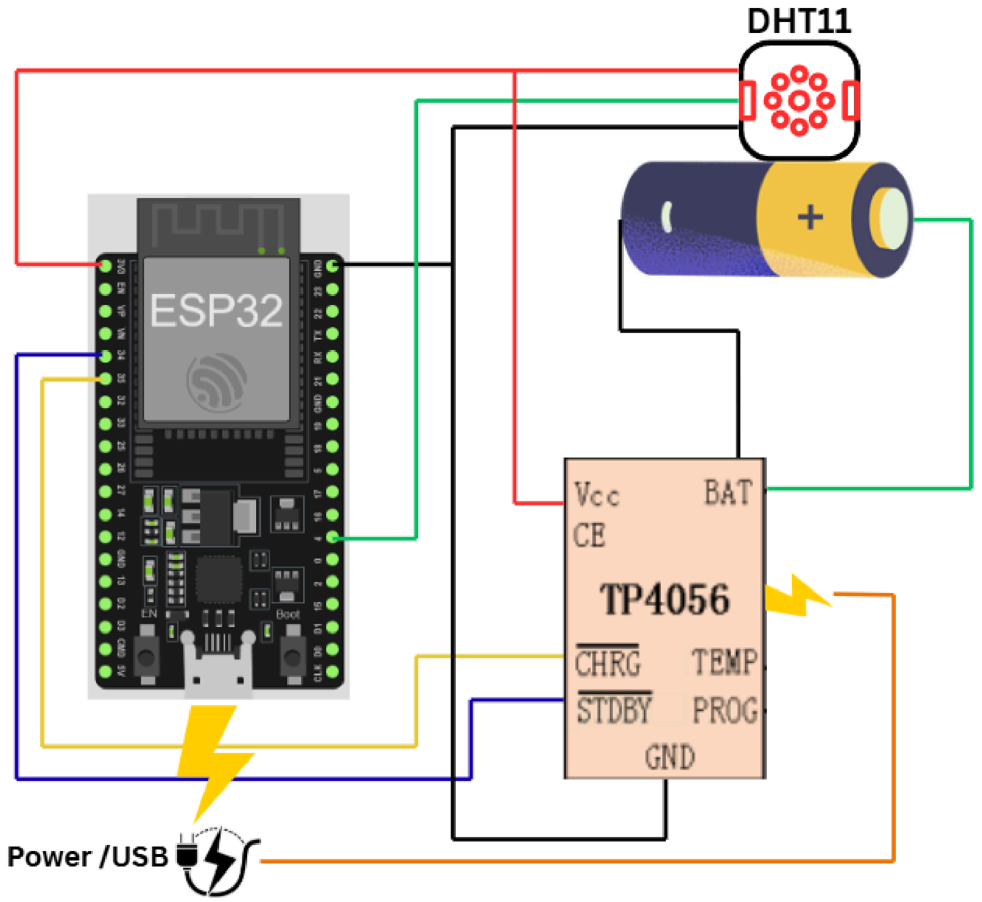
**Lab5 – On ESP32, complete the following tasks:**

Please design a power supply system based on 18650 lithium-ion batteries, including the following requirements:

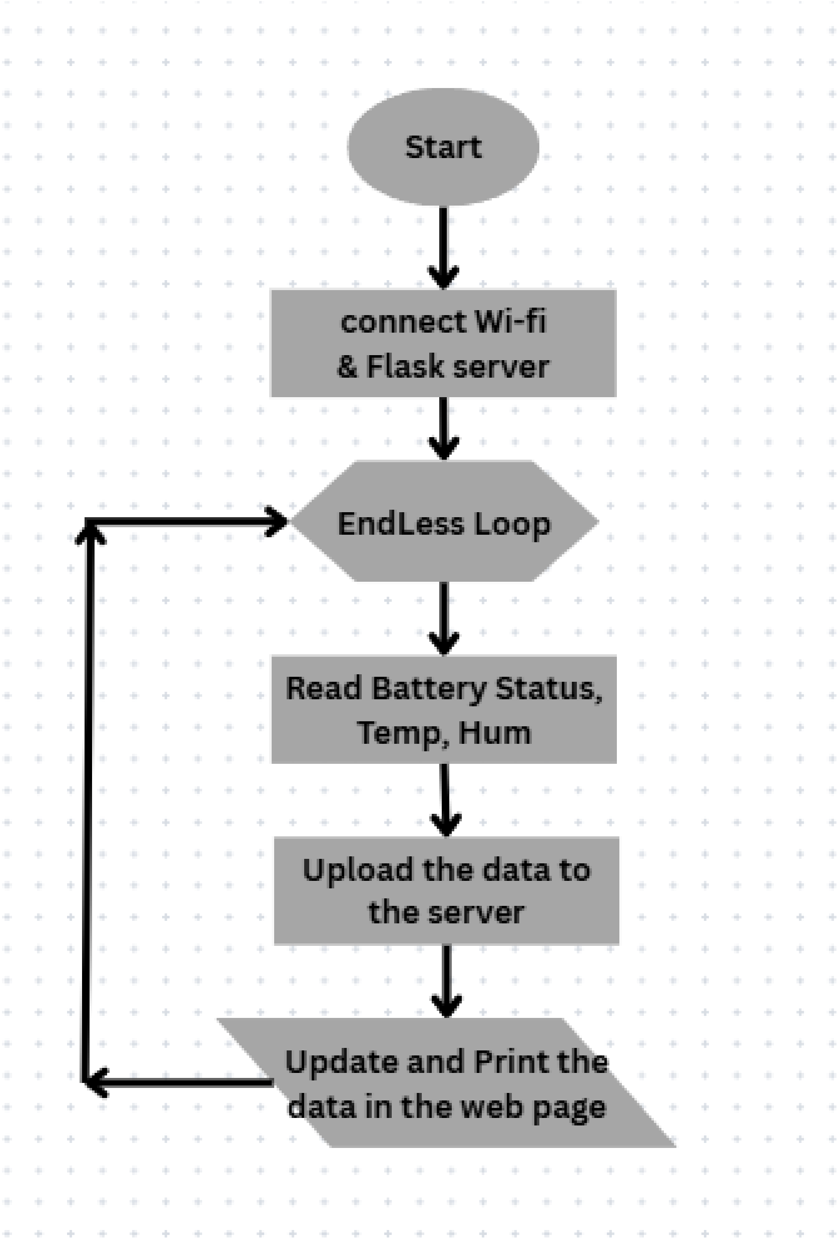
1. Battery monitoring and charging circuit design to ensure safety and stability.
2. 3D modeling of the battery enclosure to enhance structural integrity and heat dissipation performance.
3. A comprehensive design proposal that considers battery capacity, charging time, temperature, and current protection mechanisms.

**Connections:**

- TP4056 IN+ / IN- → USB 5V input
- TP4056 BAT+ / BAT- → Li-ion battery
- TP4056 OUT+ / OUT- → ESP32 VIN / GND
- TP4056 CHRG → ESP32 GPIO 34
- TP4056 STDBY → ESP32 GPIO 35
- DHT11 DATA → ESP32 GPIO 4
- DHT11 VCC/GND → ESP32 3.3V / GND



**Flow Chart :**


Start
connect Wi-fi & Flask server
EndLess Loop
Read Battery Status, Temp, Hum
Upload the data to the server
Update and Print the data in the web page

**Source code :**

**main.py** (micropython code for ESP32 programming)

```
import network
import time
import urequests
import ujson
from machine import Pin
import dht
# ---------------- Wi-Fi Setup ----------------
SSID = "ISILAB CR"
PASSWORD = "isilab.ncut.CR"
def connect_wifi():
    sta = network.WLAN(network.STA_IF)
    sta.active(True)
    if not sta.isconnected():
        print("Connecting to Wi-Fi...")
        sta.connect(SSID, PASSWORD)
        for _ in range(50):  # ~5 seconds max
            if sta.isconnected():
                break
            time.sleep_ms(100)
    if sta.isconnected():
        print("Connected! IP:", sta.ifconfig()[0])
        return True
    else:
        print("Wi-Fi connection failed")
        return False
# ---------------- Sensors ----------------
dht_sensor = dht.DHT11(Pin(4))  # DHT11 on GPIO4
# Dummy TP4056 data (replace with real pins if available)
def read_tp4056():
    charging = True   # simulate charging
    full = False      # simulate not full
    return charging, full
# ---------------- Flask API ----------------
FLASK_URL = "http://192.168.50.214:5000/update"  # Use your PC IP

# ---------------- Main Loop ----------------
if connect_wifi():
    try:
        while True:
            # Read DHT11
            try:
                dht_sensor.measure()
                temp = dht_sensor.temperature()
                hum = dht_sensor.humidity()
            except Exception as e:
                print("DHT11 error:", e)
                temp, hum = None, None

            # Read TP4056 (dummy)
            charging, full = read_tp4056()

            # Prepare data
            data = {"temperature": temp, "humidity": hum, "charging":
charging, "full": full}
            print("Sending:", data)

            # POST to Flask
            try:
                res = urequests.post(FLASK_URL,
                                     data=ujson.dumps(data),
                                     headers={"Content-Type":
"application/json"})
                print("Response:", res.text)
                res.close()
            except Exception as e:
                print("POST error:", e)

            # Short sleep allows Thonny to interrupt safely
            for _ in range(20):  # 20 x 0.1s = 2s total
                time.sleep_ms(100)

    except KeyboardInterrupt:
        print("Interrupted by user")
else:
    print("ESP32 could not connect to Wi-Fi. Reset to retry.")
```

app.py (Code for creating a flask server)

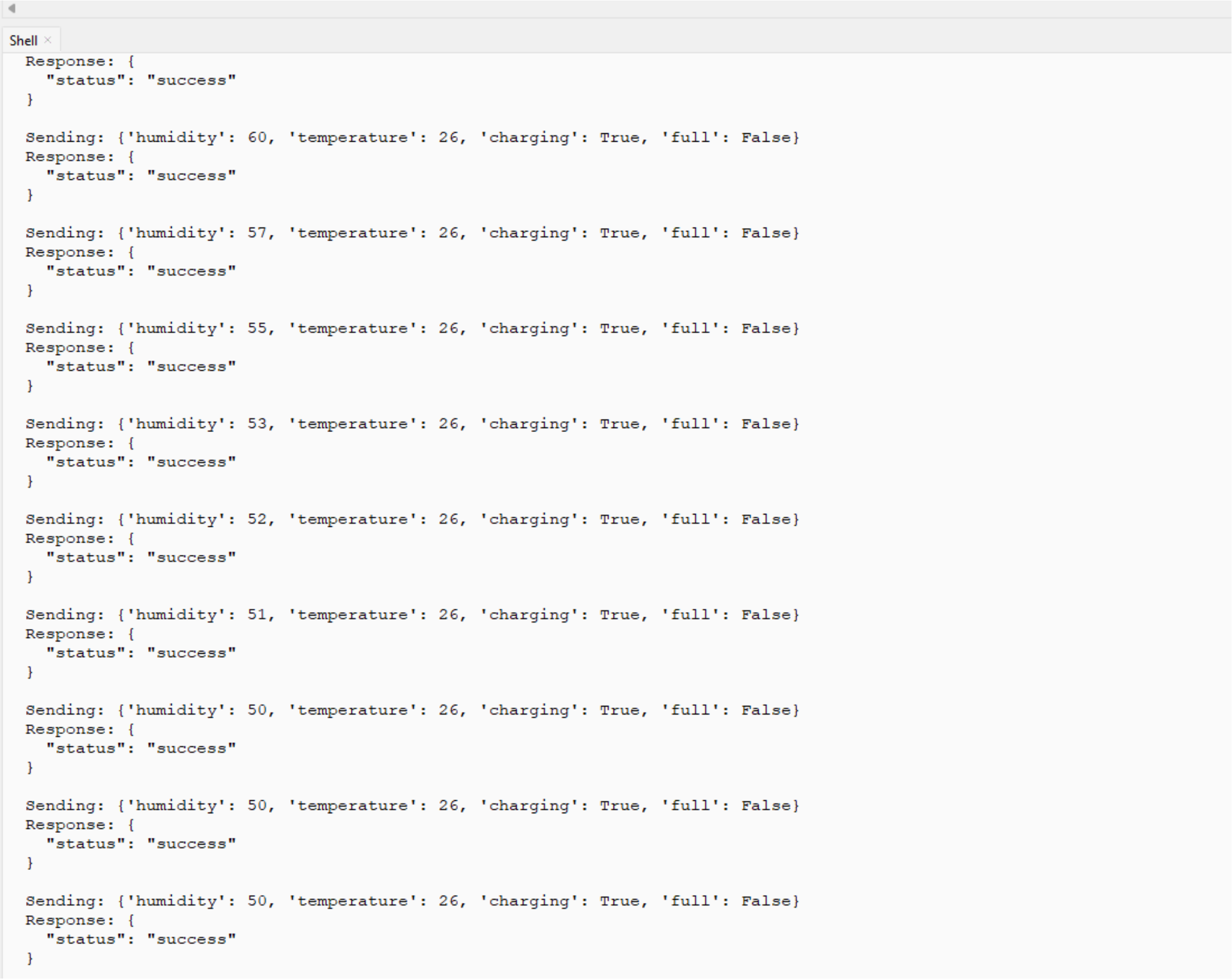
```python
from flask import Flask, request, jsonify, render_template
app = Flask(__name__)
# Store latest sensor data in memory
sensor_data = {
    "temperature": None,
    "humidity": None,
    "charging": None,
    "full": None
}
# Endpoint for ESP32 to POST data
@app.route('/update', methods=['POST'])
def update_data():
    global sensor_data
    data = request.json
    if not data:
        return jsonify({"status": "error", "message": "No JSON received"}), 400
    sensor_data.update(data)
    return jsonify({"status": "success"})
# Endpoint for frontend to GET latest data
@app.route('/data', methods=['GET'])
def get_data():
    return jsonify(sensor_data)
# Website home page
@app.route('/')
def index():
    return render_template('index.html')  # HTML page we will create

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

index.html (Code for webpage)

```html
<!doctype html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Battery Dashboard</title>
<style>
body { font-family: sans-serif; margin: 20px; }
.kpi { margin-bottom: 10px; }
.alert { color: red; font-weight: bold; }
</style>
</head>
<body>
<h1>Battery Dashboard</h1>
<div class="kpi">Temperature: <span id="temp">--</span> °C</div>
<div class="kpi">Humidity: <span id="hum">--</span> %</div>
<div class="kpi">Charging: <span id="charging">--</span></div>
<div class="kpi">Full: <span id="full">--</span></div>
<div class="alert" id="alert"></div>

<script>
async function fetchData() {
    const res = await fetch('/data');
    const data = await res.json();
    document.getElementById('temp').textContent = data.temperature ?? "--";
    document.getElementById('hum').textContent = data.humidity ?? "--";
    document.getElementById('charging').textContent = data.charging;
    document.getElementById('full').textContent = data.full;

    let alertMsg = "";
    if (data.temperature >= 45) alertMsg += " High temperature! ";
    if (data.full) alertMsg += " Battery full!";
    document.getElementById('alert').textContent = alertMsg;
}
setInterval(fetchData, 2000); // refresh every 2 sec
fetchData();
</script>
</body>
</html>
```

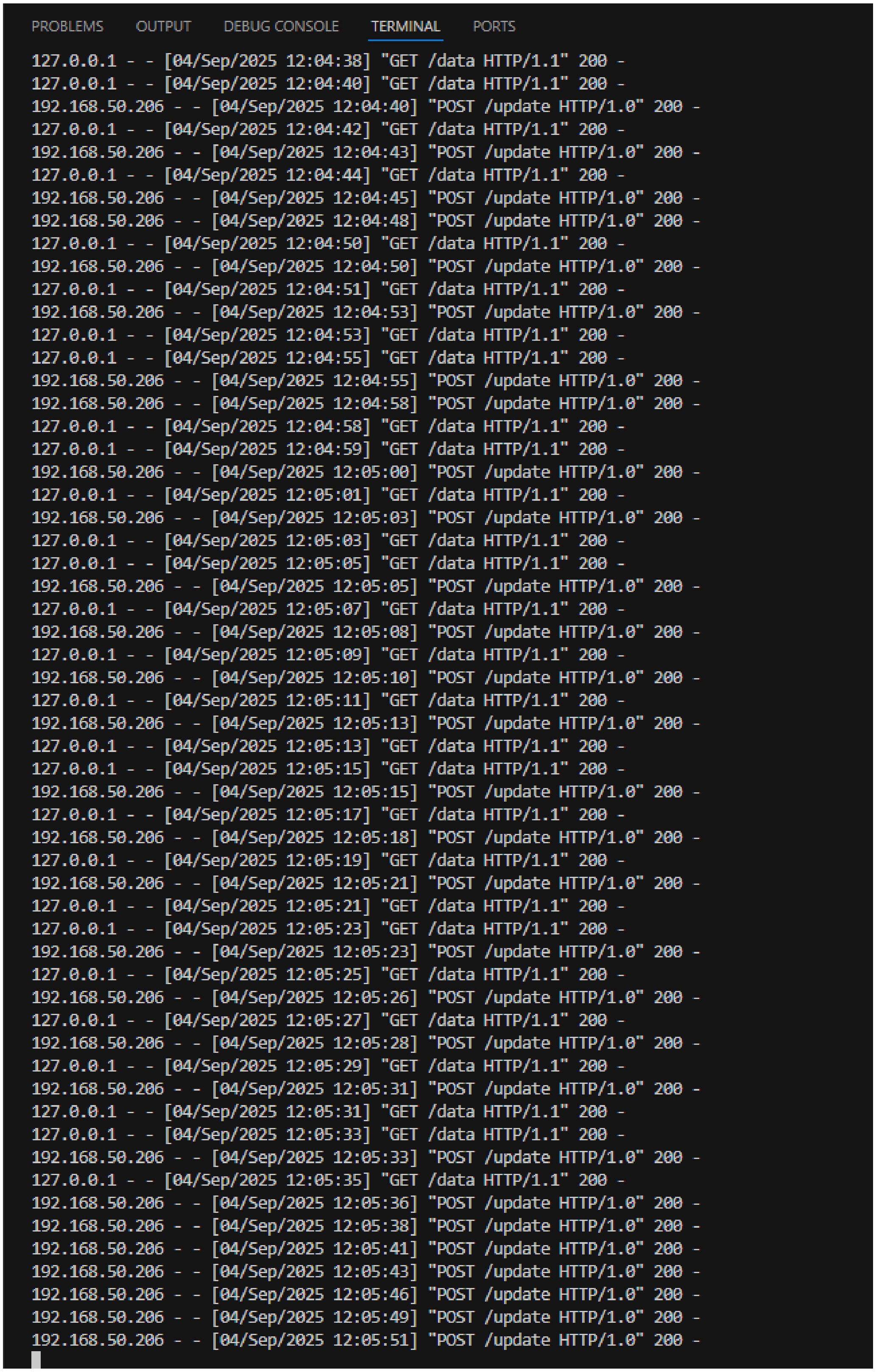
**Test records :**

Reading the data from ESP32 and sending to Web server(flask)



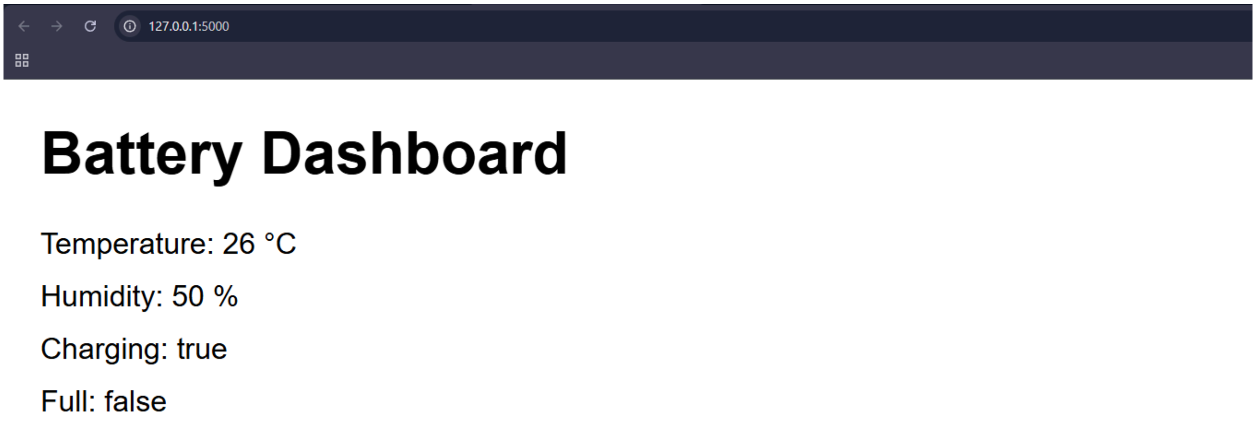
```
Response: {
  "status": "success"
}

Sending: {'humidity': 60, 'temperature': 26, 'charging': True, 'full': False}
Response: {
  "status": "success"
}

Sending: {'humidity': 57, 'temperature': 26, 'charging': True, 'full': False}
Response: {
  "status": "success"
}

Sending: {'humidity': 55, 'temperature': 26, 'charging': True, 'full': False}
Response: {
  "status": "success"
}

Sending: {'humidity': 53, 'temperature': 26, 'charging': True, 'full': False}
Response: {
  "status": "success"
}

Sending: {'humidity': 52, 'temperature': 26, 'charging': True, 'full': False}
Response: {
  "status": "success"
}

Sending: {'humidity': 51, 'temperature': 26, 'charging': True, 'full': False}
Response: {
  "status": "success"
}

Sending: {'humidity': 50, 'temperature': 26, 'charging': True, 'full': False}
Response: {
  "status": "success"
}

Sending: {'humidity': 50, 'temperature': 26, 'charging': True, 'full': False}
Response: {
  "status": "success"
}

Sending: {'humidity': 50, 'temperature': 26, 'charging': True, 'full': False}
Response: {
  "status": "success"
}
```

Gathering data from thonny and Sending data to webpage



```
127.0.0.1 - - [04/Sep/2025 12:04:38] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [04/Sep/2025 12:04:40] "GET /data HTTP/1.1" 200 -
192.168.50.206 - - [04/Sep/2025 12:04:40] "POST /update HTTP/1.0" 200 -
127.0.0.1 - - [04/Sep/2025 12:04:42] "GET /data HTTP/1.1" 200 -
192.168.50.206 - - [04/Sep/2025 12:04:43] "POST /update HTTP/1.0" 200 -
127.0.0.1 - - [04/Sep/2025 12:04:44] "GET /data HTTP/1.1" 200 -
192.168.50.206 - - [04/Sep/2025 12:04:45] "POST /update HTTP/1.0" 200 -
192.168.50.206 - - [04/Sep/2025 12:04:48] "POST /update HTTP/1.0" 200 -
127.0.0.1 - - [04/Sep/2025 12:04:50] "GET /data HTTP/1.1" 200 -
192.168.50.206 - - [04/Sep/2025 12:04:50] "POST /update HTTP/1.0" 200 -
127.0.0.1 - - [04/Sep/2025 12:04:51] "GET /data HTTP/1.1" 200 -
192.168.50.206 - - [04/Sep/2025 12:04:53] "POST /update HTTP/1.0" 200 -
127.0.0.1 - - [04/Sep/2025 12:04:53] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [04/Sep/2025 12:04:55] "GET /data HTTP/1.1" 200 -
192.168.50.206 - - [04/Sep/2025 12:04:55] "POST /update HTTP/1.0" 200 -
192.168.50.206 - - [04/Sep/2025 12:04:58] "POST /update HTTP/1.0" 200 -
127.0.0.1 - - [04/Sep/2025 12:04:58] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [04/Sep/2025 12:04:59] "GET /data HTTP/1.1" 200 -
192.168.50.206 - - [04/Sep/2025 12:05:00] "POST /update HTTP/1.0" 200 -
127.0.0.1 - - [04/Sep/2025 12:05:01] "GET /data HTTP/1.1" 200 -
192.168.50.206 - - [04/Sep/2025 12:05:03] "POST /update HTTP/1.0" 200 -
127.0.0.1 - - [04/Sep/2025 12:05:03] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [04/Sep/2025 12:05:05] "GET /data HTTP/1.1" 200 -
192.168.50.206 - - [04/Sep/2025 12:05:05] "POST /update HTTP/1.0" 200 -
127.0.0.1 - - [04/Sep/2025 12:05:07] "GET /data HTTP/1.1" 200 -
192.168.50.206 - - [04/Sep/2025 12:05:08] "POST /update HTTP/1.0" 200 -
127.0.0.1 - - [04/Sep/2025 12:05:09] "GET /data HTTP/1.1" 200 -
192.168.50.206 - - [04/Sep/2025 12:05:10] "POST /update HTTP/1.0" 200 -
127.0.0.1 - - [04/Sep/2025 12:05:11] "GET /data HTTP/1.1" 200 -
192.168.50.206 - - [04/Sep/2025 12:05:13] "POST /update HTTP/1.0" 200 -
127.0.0.1 - - [04/Sep/2025 12:05:13] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [04/Sep/2025 12:05:15] "GET /data HTTP/1.1" 200 -
192.168.50.206 - - [04/Sep/2025 12:05:15] "POST /update HTTP/1.0" 200 -
127.0.0.1 - - [04/Sep/2025 12:05:17] "GET /data HTTP/1.1" 200 -
192.168.50.206 - - [04/Sep/2025 12:05:18] "POST /update HTTP/1.0" 200 -
127.0.0.1 - - [04/Sep/2025 12:05:19] "GET /data HTTP/1.1" 200 -
192.168.50.206 - - [04/Sep/2025 12:05:21] "POST /update HTTP/1.0" 200 -
127.0.0.1 - - [04/Sep/2025 12:05:21] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [04/Sep/2025 12:05:23] "GET /data HTTP/1.1" 200 -
192.168.50.206 - - [04/Sep/2025 12:05:23] "POST /update HTTP/1.0" 200 -
127.0.0.1 - - [04/Sep/2025 12:05:25] "GET /data HTTP/1.1" 200 -
192.168.50.206 - - [04/Sep/2025 12:05:26] "POST /update HTTP/1.0" 200 -
127.0.0.1 - - [04/Sep/2025 12:05:27] "GET /data HTTP/1.1" 200 -
192.168.50.206 - - [04/Sep/2025 12:05:28] "POST /update HTTP/1.0" 200 -
127.0.0.1 - - [04/Sep/2025 12:05:29] "GET /data HTTP/1.1" 200 -
192.168.50.206 - - [04/Sep/2025 12:05:31] "POST /update HTTP/1.0" 200 -
127.0.0.1 - - [04/Sep/2025 12:05:31] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [04/Sep/2025 12:05:33] "GET /data HTTP/1.1" 200 -
192.168.50.206 - - [04/Sep/2025 12:05:33] "POST /update HTTP/1.0" 200 -
127.0.0.1 - - [04/Sep/2025 12:05:35] "GET /data HTTP/1.1" 200 -
192.168.50.206 - - [04/Sep/2025 12:05:36] "POST /update HTTP/1.0" 200 -
192.168.50.206 - - [04/Sep/2025 12:05:38] "POST /update HTTP/1.0" 200 -
192.168.50.206 - - [04/Sep/2025 12:05:41] "POST /update HTTP/1.0" 200 -
192.168.50.206 - - [04/Sep/2025 12:05:43] "POST /update HTTP/1.0" 200 -
192.168.50.206 - - [04/Sep/2025 12:05:46] "POST /update HTTP/1.0" 200 -
192.168.50.206 - - [04/Sep/2025 12:05:49] "POST /update HTTP/1.0" 200 -
192.168.50.206 - - [04/Sep/2025 12:05:51] "POST /update HTTP/1.0" 200 -
```

The web Page updates for every 2 secs automatically



127.0.0.1:5000

# Battery Dashboard

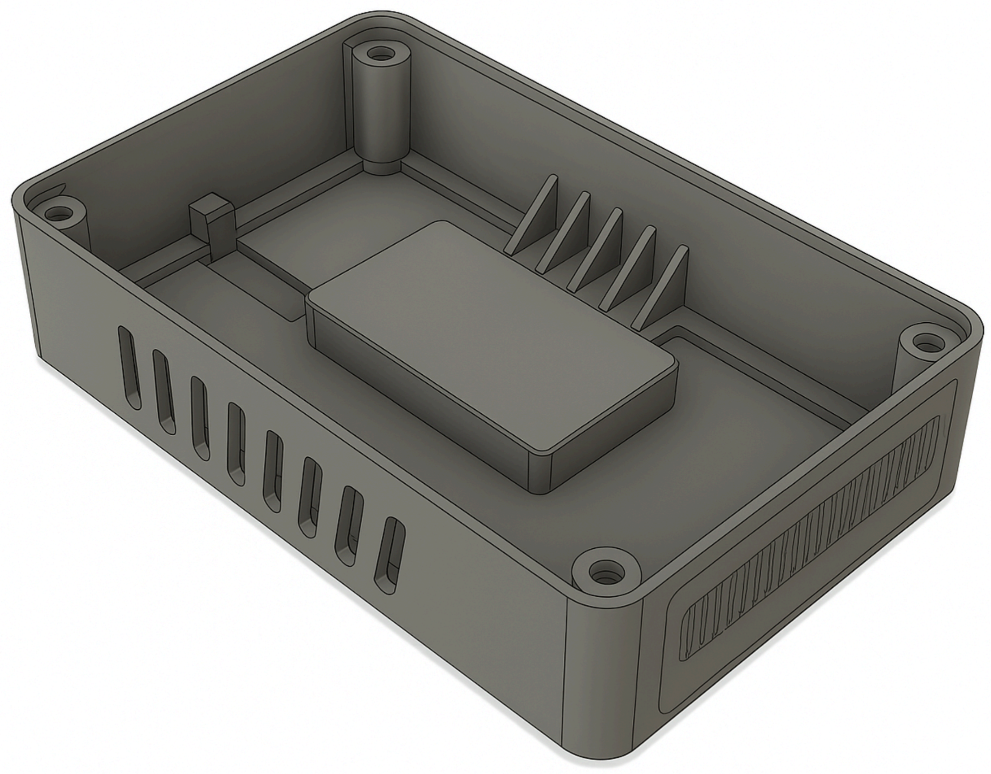
Temperature: 26 °C

Humidity: 50 %

Charging: true

Full: false

**The 3D model :** (Example) The model for a single Lithium Battery ,  the ESP32 as well as TP4056 and the DHT11 near the Battery



**Lab6 learning reflection or comments(100 words)**

The content of Learning reflection or comments can include your thoughts on the course, such as whether it was too difficult or too easy. You may also share something memorable that happened during the week. Overall, it is similar to a weekly journal, focusing on your learning experiences and personal feelings.

**Learning Reflection or FeedBack :**

Overall, I found the tasks both challenging and rewarding because they pushed me to apply theoretical knowledge into a practical project. Compared with earlier labs, the complexity increased, but I feel this was a natural progression that allowed me to connect multiple concepts together in a meaningful way.

One of the most memorable parts of this lab was the troubleshooting process. At first, I encountered difficulties when trying to read sensor data reliably and upload it to the platform. The errors were frustrating, but they also forced me to carefully review my wiring, check the code, and even revisit documentation. In the end, solving these small issues gave me a stronger sense of confidence in handling real-world hardware and software problems.

I also appreciated that this lab emphasized data handling and communication with cloud services. It made me realize how important it is to not only capture data but also to process and present it in a useful format. Being able to visualize values such as voltage, temperature, and humidity on ThingSpeak highlighted the potential of IoT systems in everyday applications like smart homes and environmental monitoring.

The Final task which was monitoring the Battery Status take most of the time to complete , And also i feel a bit tricky to complete . After completing the Battery monitoring task gave me a confidence to propagate with more tricky problem statements.

**Lab7 Video Link**

Record a video of approximately 3 minutes to explain your research process. The video should include the following:

1. A description of the tools or methods used during your research process.
2. A presentation of the results and outcomes from LAB1 to LAB7.

Link : https://youtu.be/vVqDBZ3lNxk