2025 TEEP Progress Report Week-9

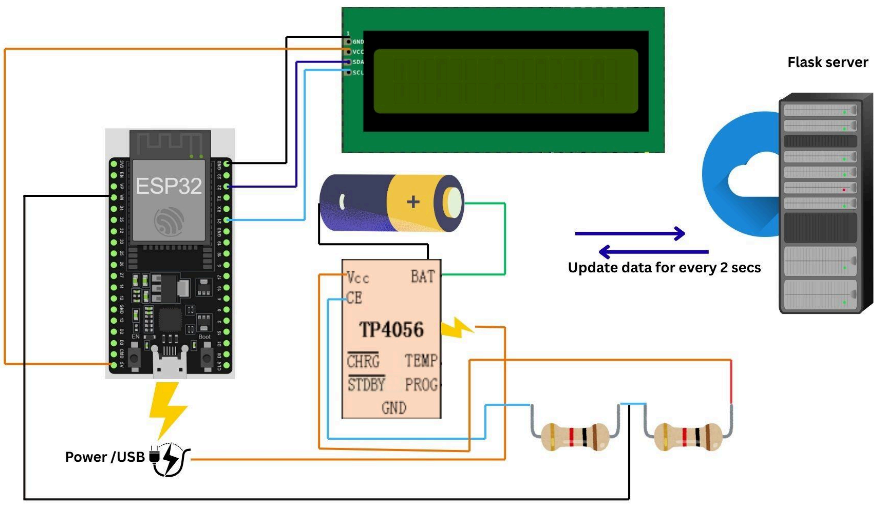
Used Wokwi to complete the tasks for Labs 1 and 2

(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 ESP32, complete the following tasks:

This project focuses on optimizing the single-cell battery charging monitoring system completed last week, with the goal of improving the overall stability of measurement and display. Each step of the experimental process must be thoroughly documented, compiled into the project report, and also presented in video format to ensure completeness and traceability of the research process.

Please record the experimental process in the Experimental Record, which



Note:
In this project, I chose to use my **own Flask-based server** for monitoring the battery system instead of relying on third-party platforms such as ThingSpeak. Hosting my own server allowed better **integration with ESP32**, real-time **data communication through APIs**, and more flexibility in **visualization** using custom dashboards built with HTML, CSS, and Chart.js. This approach also avoids the limitations of external cloud services (such as restricted data points, internet dependency, or account setup) and gives full control over the system for future modifications and scaling.

Experimental Record

Date: 22 / sep / 2025 – 26 / sep / 2025

1. Experiment Title

Battery Monitoring System with Web Dashboard

2. Objective

The objective of this experiment is to build a simple real-time battery monitoring system that:

- Reads battery voltage from ESP32 using MicroPython.
- Calculates approximate battery percentage.
- Sends live data to a Flask API.
- Displays the battery level and status on a web dashboard with charts.

3. Materials and Equipment

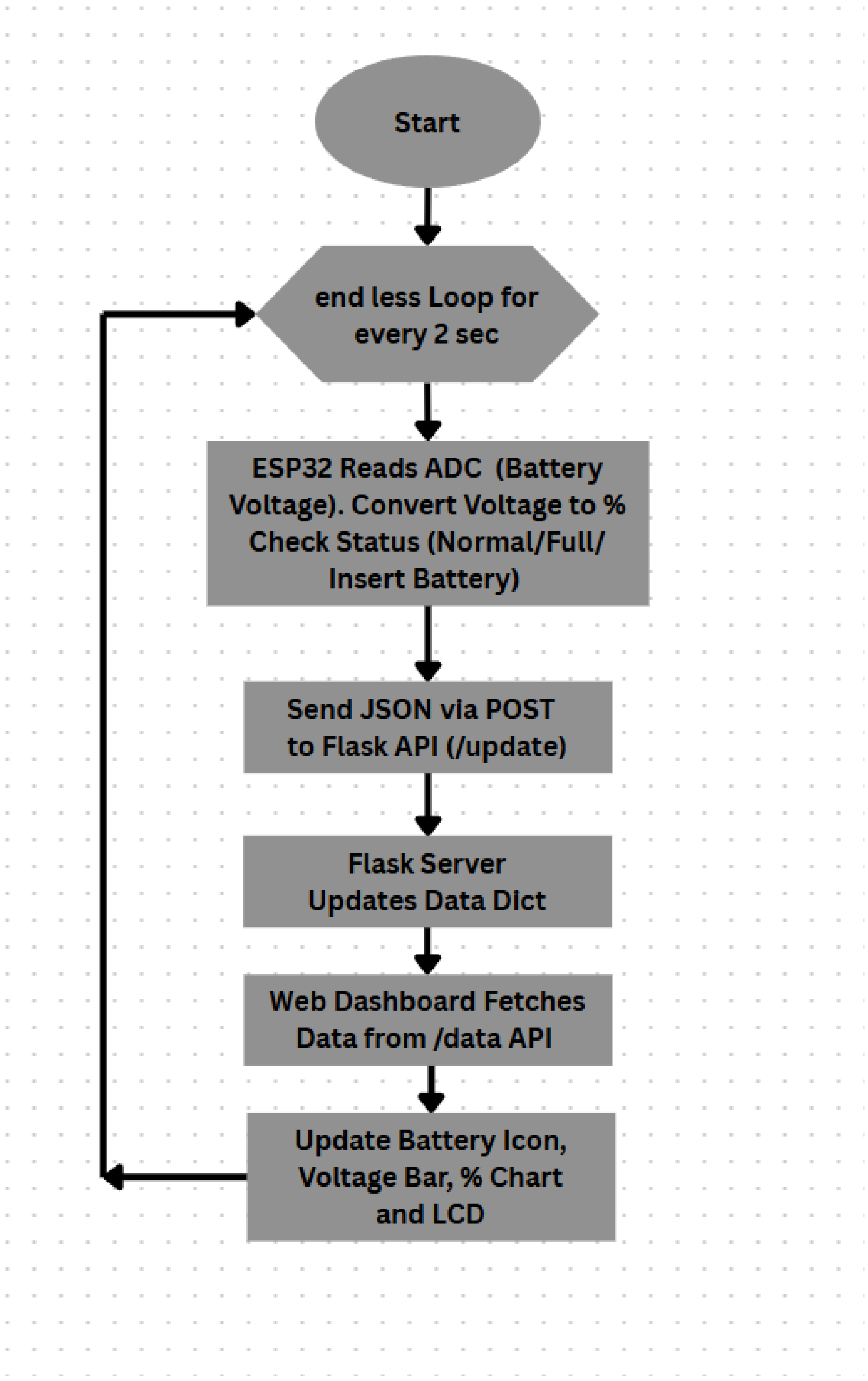
- **Hardware**:
    - ESP32 microcontroller
    - Li-Ion batteries (1S setup)
    - TP4056 charging module
    - Resistors (for voltage divider, to scale voltage to ADC range)
    - USB cable for ESP32 programming
- **Software/Tools**:
    - Thonny IDE (for MicroPython on ESP32)

- MicroPython firmware for ESP32
- Python 3.x (for Flask server)
- Flask (`pip install flask`)
- HTML, CSS, JavaScript (Chart.js) for dashboard
- Web browser (Chrome/Firefox)

4. Procedure

1. **Circuit Setup**
   - Connect Li-Ion battery to TP4056 charging module.
   - Connect ESP32 ADC pin to battery output via a resistor divider..
2. **ESP32 MicroPython Programming (via Thonny)**
   - Write a MicroPython script to:
     - Read battery voltage from ADC.
     - Convert voltage into battery percentage.
     - Determine charging status (Charging / Discharging / Please insert battery).
     - Send battery data (JSON) to Flask server using `urequests.post()`.
3. **Flask Server Setup**
   - Create a Flask app with endpoints:
     - `/` → serves the HTML dashboard.
     - `/update` → receives battery data from ESP32.
     - `/data` → provides latest battery info for frontend updates.
   - Run server using `python app.py`.
4. **Web Dashboard**
   - Design HTML page with:
     - Battery icon showing level and color-coded status.
     - Line chart for percentage history.
     - Bar chart for voltage level.
   - Use JavaScript `fetch('/data')` to update dashboard every 2 seconds.
5. **System Execution**
   - Run Flask server on PC.
   - Start ESP32 MicroPython script from Thonny.
   - Open browser → `http://<PC-IP>:5000` to view live dashboard.

5. Flow Chart :


Start
end less Loop for every 2 sec
ESP32 Reads ADC (Battery Voltage). Convert Voltage to % Check Status (Normal/Full/ Insert Battery)
Send JSON via POST to Flask API (/update)
Flask Server Updates Data Dict
Web Dashboard Fetches Data from /data API
Update Battery Icon, Voltage Bar, % Chart and LCD

6. Source Code :

main.py

```
from machine import Pin, ADC, I2C
import network
import urequests
import utime
from i2c_lcd import I2cLcd

# --- WiFi Credentials ---
SSID = "ISILAB CR"
PASSWORD = "isilab.ncut.CR"

# --- Flask Server URL ---
SERVER_URL = "http://192.168.50.205:5000/update"  # Replace with your PC LAN IP

# --- ESP32 Pin Definitions ---
BATTERY_VOLTAGE_PIN = 36

# --- Battery specs ---
FULL_CHARGE_VOLTAGE = 4.15
VOLTAGE_CALIBRATION = 2.0

# --- LCD setup ---
i2c = I2C(0, scl=Pin(22), sda=Pin(21), freq=400000)
lcd = I2cLcd(i2c, 0x27, 2, 16)

# --- ADC setup ---
adc = ADC(Pin(BATTERY_VOLTAGE_PIN))
adc.atten(ADC.ATTN_11DB)
adc.width(ADC.WIDTH_12BIT)

# --- WiFi connect ---
def connect_wifi():
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    wlan.connect(SSID, PASSWORD)
    lcd.clear()
    lcd.putstr("Connecting WiFi")
    timeout = 20
    while not wlan.isconnected() and timeout > 0:
        utime.sleep(1)
        timeout -= 1
    lcd.clear()
    if wlan.isconnected():
        lcd.putstr("WiFi OK")
        print("Wi-Fi connected:", wlan.ifconfig())
        utime.sleep(2)
    else:
        lcd.putstr("WiFi Fail")
        print("Wi-Fi connection failed!")
        utime.sleep(2)
    return wlan

# --- Battery reading ---
def read_voltage(samples=10):
    total = 0
    for _ in range(samples):
        total += adc.read()
        utime.sleep_ms(10)
    avg = total / samples
    voltage = round((avg / 4095 * 3.3) * VOLTAGE_CALIBRATION, 3)
    return voltage

def calculate_percentage(voltage):
    if voltage >= 4.15: return 100
    elif voltage >= 4.05: return 90 + ((voltage - 4.05) / 0.1) * 10
    elif voltage >= 3.95: return 70 + ((voltage - 3.95) / 0.1) * 20
    elif voltage >= 3.85: return 50 + ((voltage - 3.85) / 0.1) * 20
    elif voltage >= 3.70: return 25 + ((voltage - 3.7) / 0.15) * 25
    elif voltage >= 3.30: return 5 + ((voltage - 3.3) / 0.4) * 20
    else: return 0

# --- Upload to Flask ---
def upload(voltage, pct, status):
    try:
        data = {"voltage": voltage, "percentage": pct, "status": status}
        res = urequests.post(SERVER_URL, json=data)
        print("Flask response:", res.text)
        res.close()
    except Exception as e:
        print("Upload failed:", e)

# --- Main ---
wlan = connect_wifi()
last_cloud_update = utime.ticks_ms()
prev_voltage = 0

while True:
    voltage = read_voltage()
    pct = calculate_percentage(voltage)
    status = "FULL" if voltage >= FULL_CHARGE_VOLTAGE else "NORMAL"

    # Update LCD only if significant change
    if abs(voltage - prev_voltage) >= 0.02:
        lcd.move_to(0,0)
        lcd.putstr("{:.2f}V {:>3}%".format(voltage, int(pct)) + "     ")
        lcd.move_to(0,1)
        lcd.putstr(status + " "*(16-len(status)))
        prev_voltage = voltage

    # Upload to Flask every 10 seconds
    if utime.ticks_diff(utime.ticks_ms(), last_cloud_update) > 10000 and
wlan.isconnected():
        upload(voltage, pct, status)
        last_cloud_update = utime.ticks_ms()

    utime.sleep(1)
```

Lcd_api.py (save this esp32)

```
class LcdApi:
    # LCD commands
    LCD_CLR = 0x01
    LCD_HOME = 0x02
    LCD_ENTRY_MODE = 0x04
    LCD_ENTRY_INC = 0x02
    LCD_ENTRY_SHIFT = 0x01

    LCD_ON_CTRL = 0x08
    LCD_ON_DISPLAY = 0x04
    LCD_ON_CURSOR = 0x02
    LCD_ON_BLINK = 0x01

    LCD_MOVE = 0x10
    LCD_MOVE_DISP = 0x08
    LCD_MOVE_RIGHT = 0x04

    LCD_FUNCTION = 0x20
    LCD_FUNCTION_8BIT = 0x10
    LCD_FUNCTION_2LINES = 0x08
    LCD_FUNCTION_10DOTS = 0x04
    LCD_FUNCTION_RESET = 0x30
    LCD_CGRAM = 0x40
    LCD_DDRAM = 0x80

    LCD_RS_CMD = 0
    LCD_RS_DATA = 1

    LCD_RW_WRITE = 0
    LCD_RW_READ = 1

    def __init__(self, num_lines, num_columns):
        self.num_lines = min(num_lines, 4)
        self.num_columns = min(num_columns, 40)
        self.cursor_x = 0
        self.cursor_y = 0

    def clear(self):
        self.write_cmd(self.LCD_CLR)
        self.cursor_x = 0
        self.cursor_y = 0

    def home(self):
        self.write_cmd(self.LCD_HOME)
        self.cursor_x = 0
        self.cursor_y = 0

    def show_cursor(self):
        self.write_cmd(self.LCD_ON_CTRL | self.LCD_ON_DISPLAY |
self.LCD_ON_CURSOR)

    def hide_cursor(self):
        self.write_cmd(self.LCD_ON_CTRL | self.LCD_ON_DISPLAY)

    def blink_cursor_on(self):
        self.write_cmd(self.LCD_ON_CTRL | self.LCD_ON_DISPLAY |
                       self.LCD_ON_CURSOR | self.LCD_ON_BLINK)

    def blink_cursor_off(self):
        self.write_cmd(self.LCD_ON_CTRL | self.LCD_ON_DISPLAY |
self.LCD_ON_CURSOR)

    def display_on(self):
        self.write_cmd(self.LCD_ON_CTRL | self.LCD_ON_DISPLAY)

    def display_off(self):
        self.write_cmd(self.LCD_ON_CTRL)

    def move_to(self, cursor_x, cursor_y):
        self.cursor_x = cursor_x
        self.cursor_y = cursor_y
        addr = cursor_x & 0x3f
        if cursor_y & 1:
            addr += 0x40
        if cursor_y & 2:
            addr += 0x14
        self.write_cmd(self.LCD_DDRAM | addr)

    def putchar(self, char):
        if char != '\n':
            self.write_data(ord(char))
            self.cursor_x += 1
            if self.cursor_x >= self.num_columns:
                self.cursor_x = 0
                self.cursor_y += 1
                if self.cursor_y >= self.num_lines:
                    self.cursor_y = 0
                self.move_to(self.cursor_x, self.cursor_y)

    def putstr(self, string):
        for char in string:
            self.putchar(char)

    def custom_char(self, location, charmap):
        location &= 0x7
        self.write_cmd(self.LCD_CGRAM | (location << 3))
        for i in range(8):
            self.write_data(charmap[i])
```

I2c_lcd.py (save this esp32)

```
from lcd_api import LcdApi
from machine import I2C
from time import sleep_ms

MASK_RS = 0x01
MASK_RW = 0x02
MASK_E = 0x04
SHIFT_BACKLIGHT = 3
SHIFT_DATA = 4
```

```
class I2cLcd(LcdApi):
    def __init__(self, i2c, i2c_addr, num_lines, num_columns):
        self.i2c = i2c
        self.i2c_addr = i2c_addr
        self.backlight = 1
        self.mcp_output(0)
        sleep_ms(20)
        self.hal_write_init_nibble(self.LCD_FUNCTION_RESET)
        sleep_ms(5)
        self.hal_write_init_nibble(self.LCD_FUNCTION_RESET)
        sleep_ms(1)
        self.hal_write_init_nibble(self.LCD_FUNCTION_RESET)
        sleep_ms(1)
        self.hal_write_init_nibble(self.LCD_FUNCTION)
        sleep_ms(1)
        LcdApi.__init__(self, num_lines, num_columns)
        cmd = self.LCD_FUNCTION
        if num_lines > 1:
            cmd |= self.LCD_FUNCTION_2LINES
        self.write_cmd(cmd)
        self.display_on()
        self.clear()
        self.write_cmd(self.LCD_ENTRY_MODE | self.LCD_ENTRY_INC)

    def mcp_output(self, data):
        self.i2c.writeto(self.i2c_addr, bytes([data]))

    def hal_write_init_nibble(self, nibble):
        data = ((nibble >> 4) & 0x0f) << SHIFT_DATA
        self.pulse_enable(data)

    def pulse_enable(self, data):
        self.mcp_output(data | MASK_E | (self.backlight << SHIFT_BACKLIGHT))
        sleep_ms(1)
        self.mcp_output(data & ~MASK_E | (self.backlight << SHIFT_BACKLIGHT))
        sleep_ms(1)

    def hal_backlight_on(self):
        self.backlight = 1
        self.mcp_output(self.backlight << SHIFT_BACKLIGHT)

    def hal_backlight_off(self):
        self.backlight = 0
        self.mcp_output(0)
```

```
def write_cmd(self, cmd):
    self.hal_write(cmd, 0)

def write_data(self, data):
    self.hal_write(data, MASK_RS)

def hal_write(self, data, mode):
    high = (data & 0xf0) | mode
    low = ((data << 4) & 0xf0) | mode
    self.pulse_enable(high)
    self.pulse_enable(low)
```

app.py

```
from flask import Flask, request, render_template, jsonify

app = Flask(__name__)

# Store latest battery data
battery_data = {"voltage": 0, "percentage": 0, "status": "Unknown"}

@app.route('/')
def index():
    # Render HTML dashboard from templates/index.html
    return render_template("index.html")

@app.route('/update', methods=['POST'])
def update():
    global battery_data
    data = request.get_json(force=True)  # ensure JSON is parsed

    battery_data['voltage'] = round(float(data.get('voltage', 0)), 2)
    battery_data['percentage'] = int(data.get('percentage', 0))
    battery_data['status'] = data.get('status', "Unknown")

    print("Received:", battery_data)  # debug log in terminal
    return jsonify({"status": "ok"})

@app.route('/data')
def data():
```

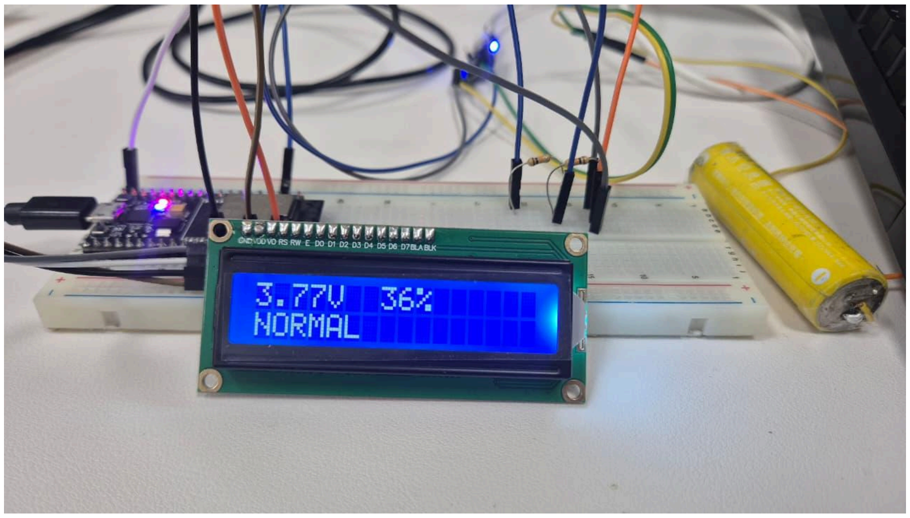
```python
    return jsonify(battery_data)

if __name__ == "__main__":
    # Accessible to ESP32 on local WiFi
    app.run(host="0.0.0.0", port=5000, debug=True)
```

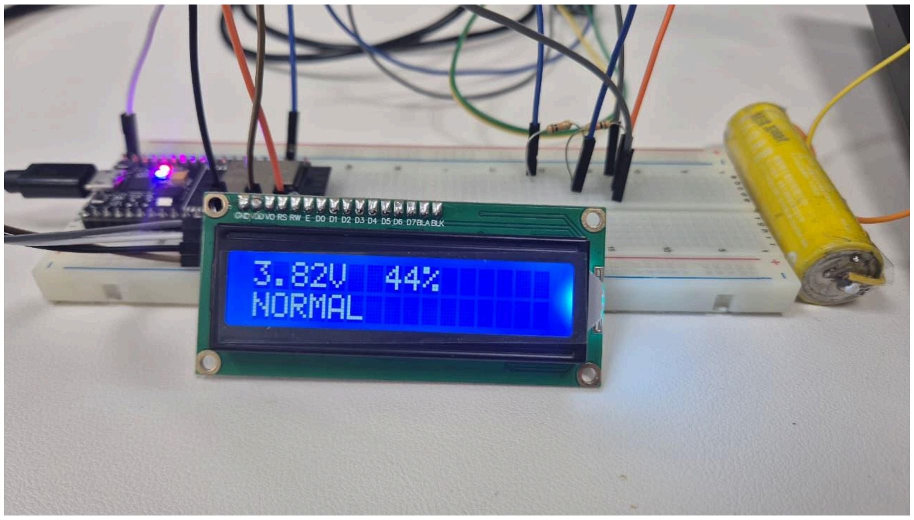
Index.html

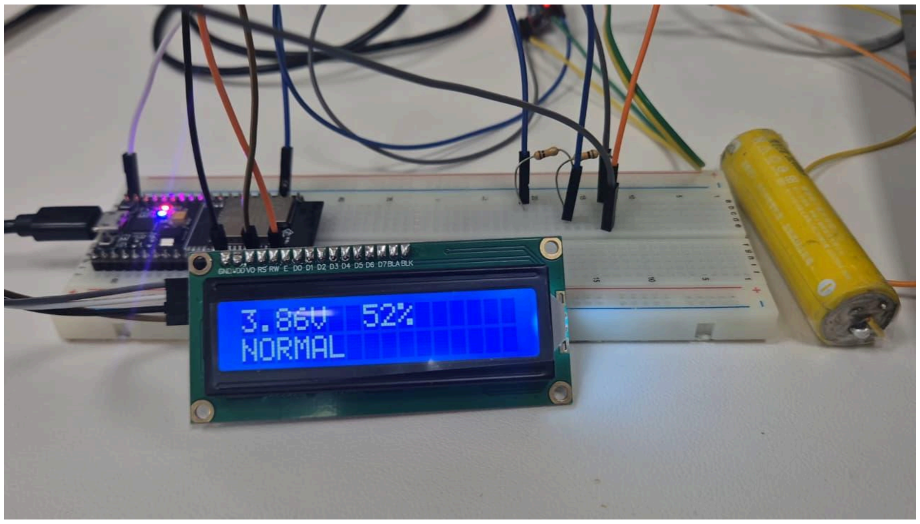
```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Battery Monitor</title>
<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
<style>
body { font-family: Arial; text-align:center; background: #f0f2f5;
padding:20px; }
h1 { color:#333; }
.card { display:inline-block; background:#fff; border-radius:15px;
padding:20px; margin:10px; width:250px; box-shadow:0 4px 10px
rgba(0,0,0,0.1);}
#batteryIcon { width:80px; height:150px; border:4px solid #555;
border-radius:10px; position:relative; margin:20px auto; background:#ddd;}
#batteryLevel { position:absolute; bottom:0; width:100%; height:50%;
background:green; border-radius:6px 6px 0 0; transition:height 0.5s,
background 0.5s;}
.chart-container { width:80%; max-width:700px; margin:auto;
margin-top:30px; }
.status-text { font-size:1.2em; margin-top:10px; font-weight:bold; }
</style>
</head>
<body>

<h1>🔋 Battery Monitor Dashboard</h1>

<div class="card">
  <h2>Battery Status</h2>
  <div id="batteryIcon"><div id="batteryLevel"></div></div>
  <div class="status-text" id="status">Unknown</div>
  <div class="status-text" id="percentageText">0%</div>
```

```
</div>

<div class="chart-container"><canvas id="percentageChart"></canvas></div>
<div class="chart-container"><canvas id="voltageChart"></canvas></div>

<script>
let percentageData = [], labels = [], maxPoints=20;

// Line chart for battery %
const ctxPercentage =
document.getElementById('percentageChart').getContext('2d');
const percentageChart = new Chart(ctxPercentage,{
    type:'line',
    data:{
        labels:labels,
        datasets:[{
            label:'Battery %',
            data:percentageData,
            borderColor:'rgba(0,200,83,1)',
            backgroundColor:'rgba(0,200,83,0.2)',
            fill:true,
            tension:0.4,
            pointRadius:4,
            pointBackgroundColor:'rgba(0,200,83,1)'
        }]
    },
    options:{
        responsive:true,
        scales:{y:{beginAtZero:true,max:100}}
    }
});

// Voltage bar chart
const ctxVoltage =
document.getElementById('voltageChart').getContext('2d');
const voltageChart = new Chart(ctxVoltage,{
    type:'bar',
    data:{
        labels:['Voltage'],
        datasets:[{label:'Voltage (V)',data:[0],backgroundColor:'green'}]
    },
    options:{
        responsive:true,
        scales:{y:{beginAtZero:true,max:5}}
    }
});

function updateDashboard(){
    fetch('/data').then(r=>r.json()).then(data=>{
        let level = Math.max(0,Math.min(100,data.percentage));

        // battery icon
        document.getElementById('batteryLevel').style.height = level+'%';
        document.getElementById('batteryLevel').style.background =
            level>80?'green':level>50?'yellow':level>20?'orange':'red';
        document.getElementById('status').innerText = data.status;
        document.getElementById('percentageText').innerText = level+'%';
        // line chart
        const now = new Date().toLocaleTimeString();
        if(labels.length>=maxPoints){ labels.shift();
percentageData.shift(); }
        labels.push(now); percentageData.push(data.percentage);
percentageChart.update();

        // voltage chart
        voltageChart.data.datasets[0].data[0] = data.voltage;
        voltageChart.data.datasets[0].backgroundColor =
            data.voltage>4?'green':data.voltage>3.5?'orange':'red';
        voltageChart.update();
    }).catch(err=>{
        console.log("Fetch error:", err);
    });
}

// update every 2 sec
setInterval(updateDashboard,2000);
</script>

</body>
</html>
```

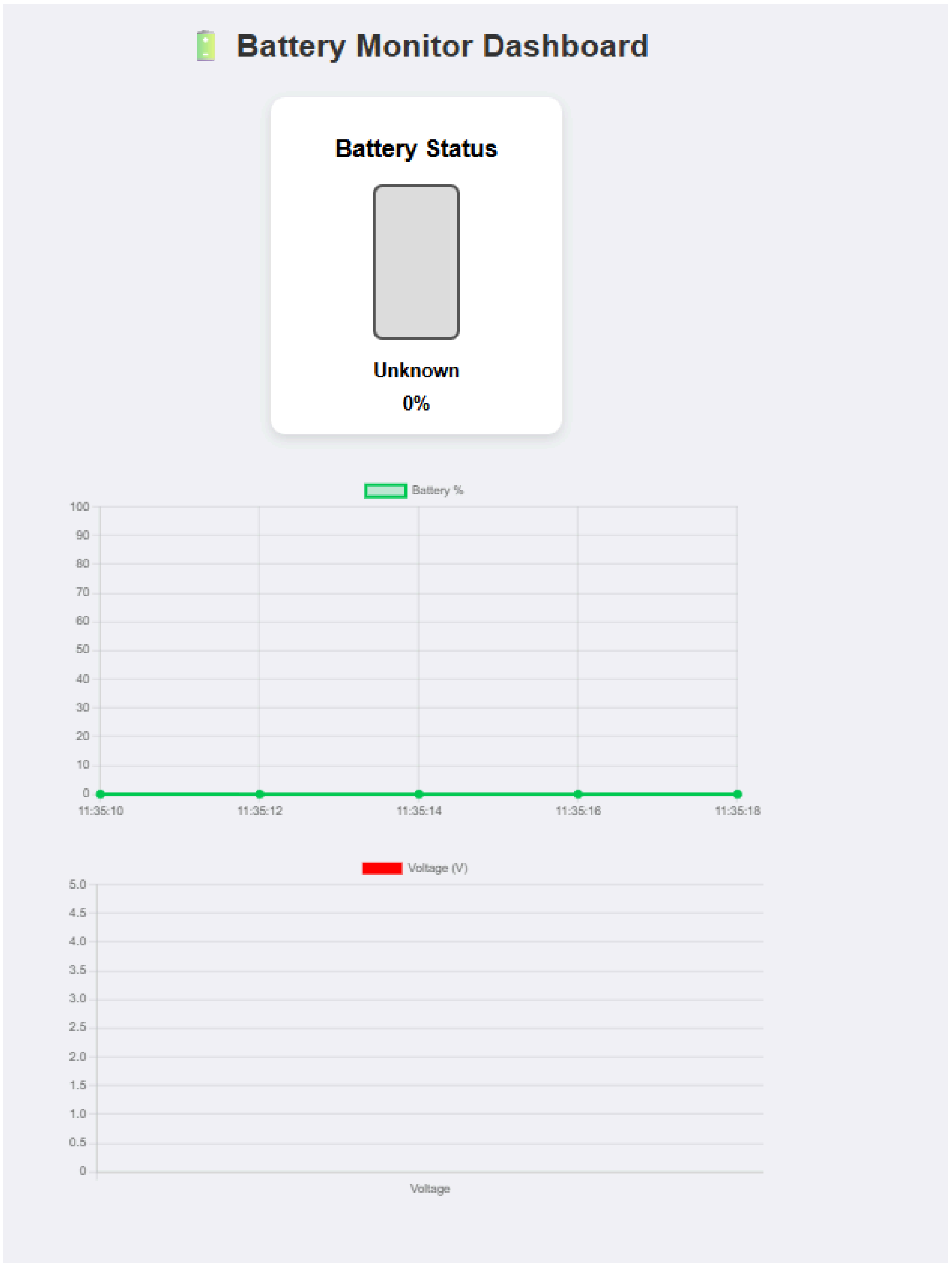
7. Results







Dash Board when the battery is not inserted

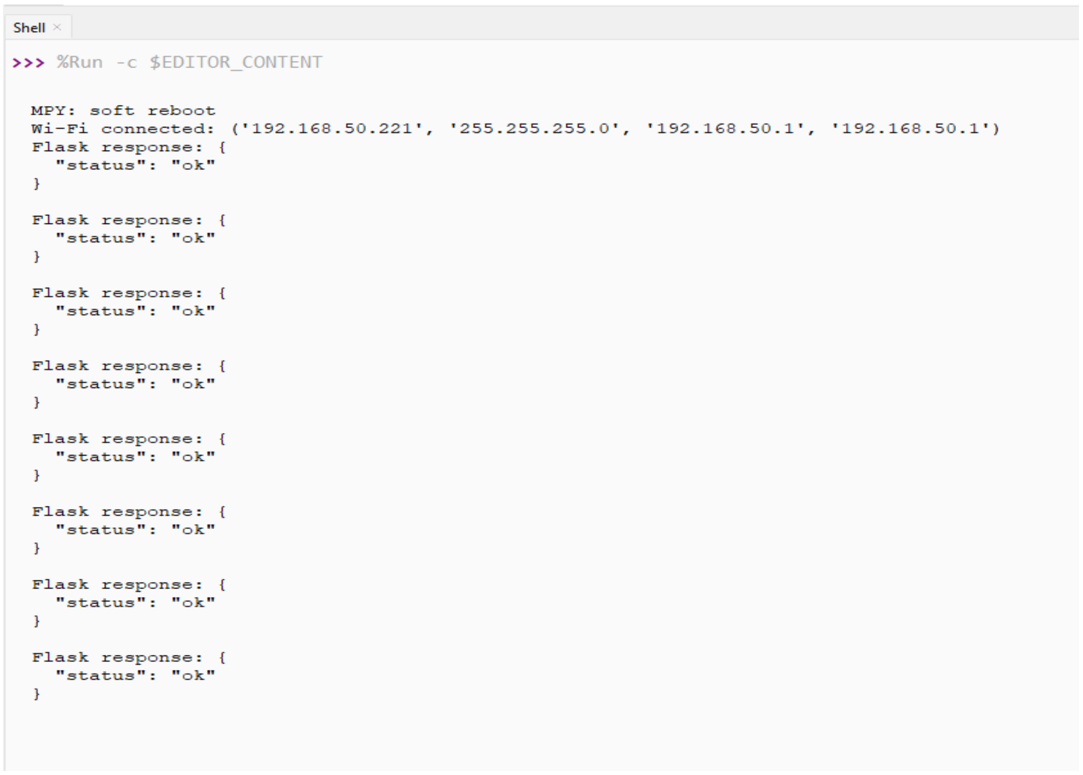

Battery Monitor Dashboard

Battery Status

Unknown

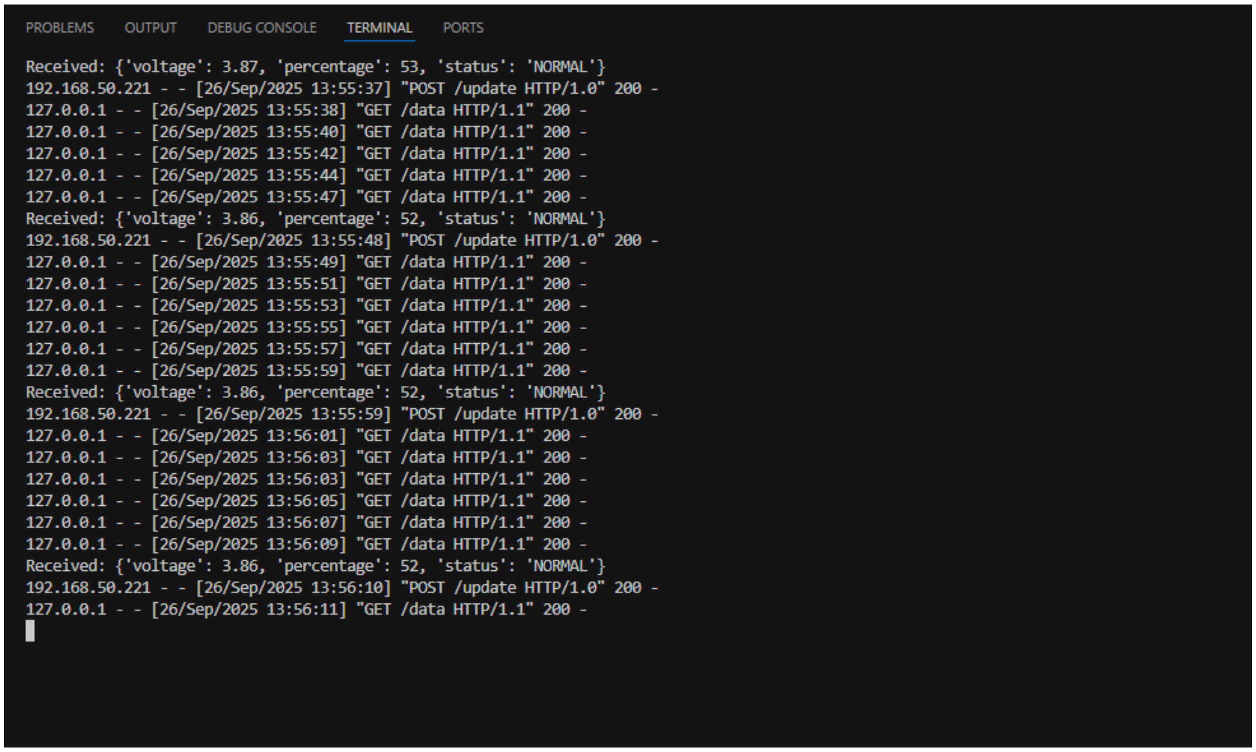
0%

The data was updating using ESP32

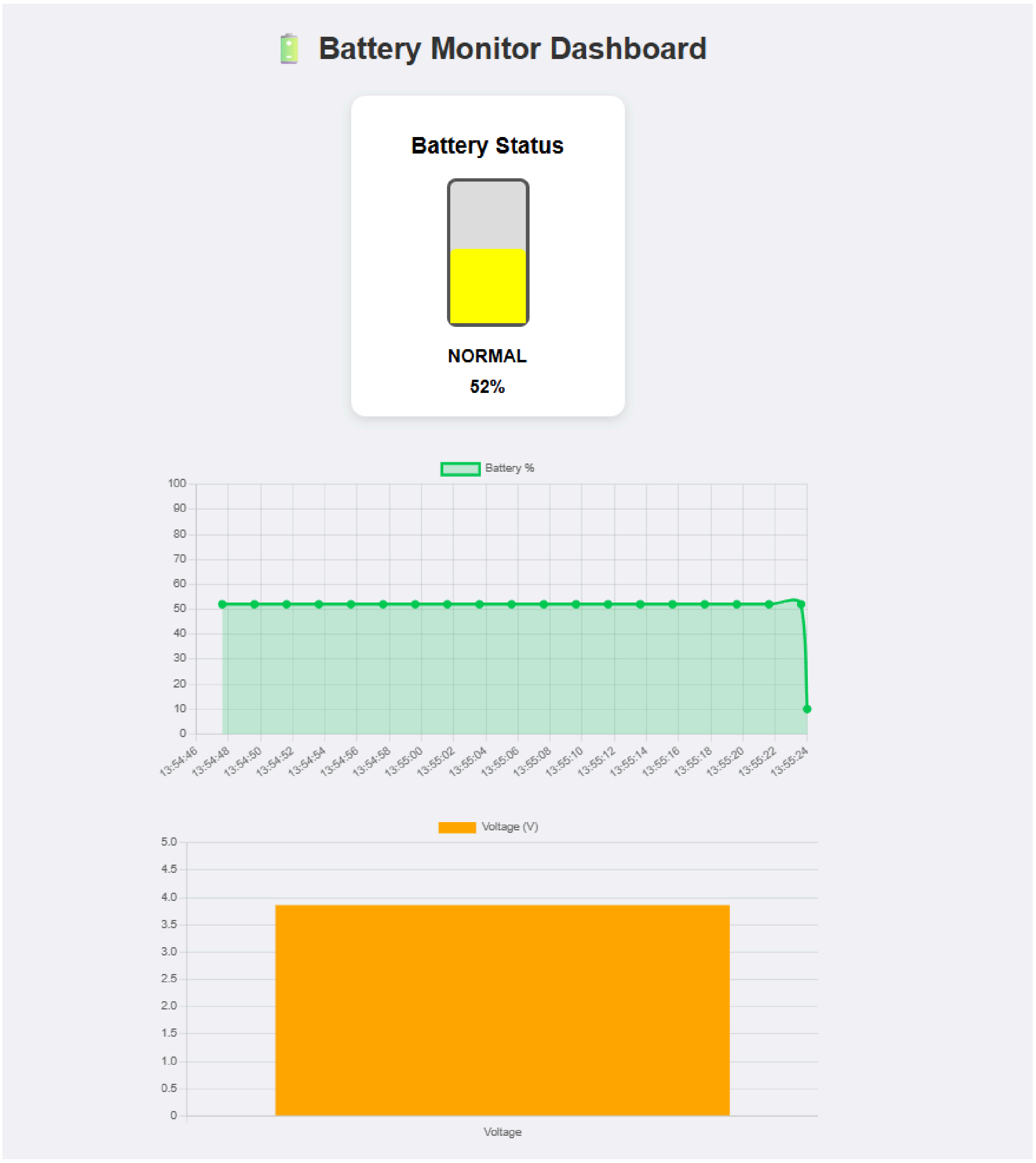


```
>>> %Run -c $EDITOR_CONTENT

  MPY: soft reboot
  Wi-Fi connected: ('192.168.50.221', '255.255.255.0', '192.168.50.1', '192.168.50.1')
  Flask response: {
    "status": "ok"
  }

  Flask response: {
    "status": "ok"
  }

  Flask response: {
    "status": "ok"
  }

  Flask response: {
    "status": "ok"
  }

  Flask response: {
    "status": "ok"
  }

  Flask response: {
    "status": "ok"
  }

  Flask response: {
    "status": "ok"
  }

  Flask response: {
    "status": "ok"
  }
```

The flask server sending data to the Web



```
Received: {'voltage': 3.87, 'percentage': 53, 'status': 'NORMAL'}
192.168.50.221 - - [26/Sep/2025 13:55:37] "POST /update HTTP/1.0" 200 -
127.0.0.1 - - [26/Sep/2025 13:55:38] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [26/Sep/2025 13:55:40] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [26/Sep/2025 13:55:42] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [26/Sep/2025 13:55:44] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [26/Sep/2025 13:55:47] "GET /data HTTP/1.1" 200 -
Received: {'voltage': 3.86, 'percentage': 52, 'status': 'NORMAL'}
192.168.50.221 - - [26/Sep/2025 13:55:48] "POST /update HTTP/1.0" 200 -
127.0.0.1 - - [26/Sep/2025 13:55:49] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [26/Sep/2025 13:55:51] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [26/Sep/2025 13:55:53] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [26/Sep/2025 13:55:55] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [26/Sep/2025 13:55:57] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [26/Sep/2025 13:55:59] "GET /data HTTP/1.1" 200 -
Received: {'voltage': 3.86, 'percentage': 52, 'status': 'NORMAL'}
192.168.50.221 - - [26/Sep/2025 13:55:59] "POST /update HTTP/1.0" 200 -
127.0.0.1 - - [26/Sep/2025 13:56:01] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [26/Sep/2025 13:56:03] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [26/Sep/2025 13:56:03] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [26/Sep/2025 13:56:05] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [26/Sep/2025 13:56:07] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [26/Sep/2025 13:56:09] "GET /data HTTP/1.1" 200 -
Received: {'voltage': 3.86, 'percentage': 52, 'status': 'NORMAL'}
192.168.50.221 - - [26/Sep/2025 13:56:10] "POST /update HTTP/1.0" 200 -
127.0.0.1 - - [26/Sep/2025 13:56:11] "GET /data HTTP/1.1" 200 -
```



🔋 Battery Monitor Dashboard

Battery Status

NORMAL

52%

8. Discussion/Analysis

- The system worked in real-time without storing data.
- Thonny + MicroPython simplified ESP32 programming.
- Charts updated smoothly with live fetch requests.
- Voltage divider calibration is critical for accurate battery readings.
- Possible improvements:
    - Add filtering/averaging for more stable readings.
    - Extend dashboard with temperature/current monitoring.
    - Deploy Flask on Raspberry Pi for portability.

9. Conclusion

The experiment successfully demonstrated a lightweight IoT-based battery monitoring system using ESP32, Flask, and Chart.js. The system achieved live monitoring of voltage, percentage, and status without requiring data storage.

10. References

- Flask Documentation: https://flask.palletsprojects.com/
- ESP32 MicroPython Docs: https://docs.micropython.org/
- TP4056 Datasheet
- Chart.js Documentation: https://www.chartjs.org/

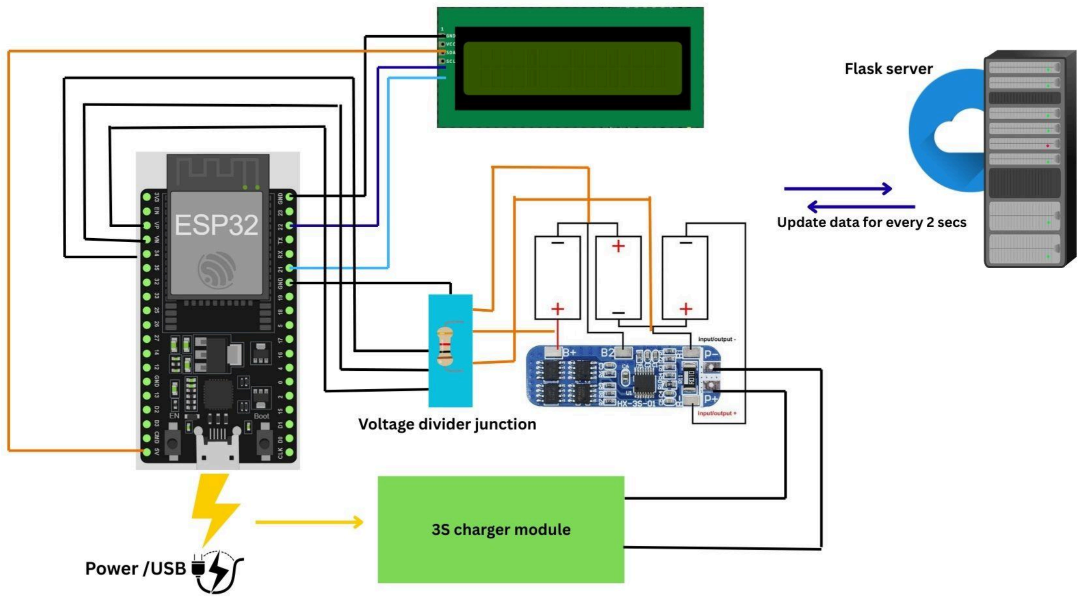
Lab2 – On ESP32, complete the following tasks:

This project focuses on optimizing the 12V battery charging monitoring system, which was completed last week using three 3.7V Li-Ion cells connected in series. The aim is to enhance the overall stability of measurement and display. Each step of the experimental process must be thoroughly documented, compiled into the project report, and also presented in video format to ensure the completeness and traceability of the research process.

Please record the experimental process in the Experimental Record, which should include the following:

Note :

The practical implementation of the charging circuit was **not completed** because a dedicated **3S Li-ion charger module** was not available during the experiment. Only the **monitoring functions (voltage measurement and dashboard visualization)** were carried out. The charging and balancing functions will be tested once the proper 3S charging module is provided.



Experimental Record :

Date: 22 / sep / 2025 – 26 / sep / 2025

Course/Project Title: Development of IOT Situation room for elevator operation monitoring ans health diagnosis.

1. Experiment Title

Battery Monitoring System using ESP32, Flask Server, and Web Dashboard

2. Objective

The purpose of this experiment is to design and implement a smart battery monitoring system for a 3-cell Li-ion battery pack. The system measures individual cell voltages and pack voltage using ESP32 and displays real-time data on a Flask-based web dashboard. This allows tracking of battery status, charge percentage, and performance.

3. Materials and Equipment

- **Hardware**
    - ESP32 development board
    - 3 × 18650 Li-ion cells
    - 3S BMS (Battery Management System) module
    - LCD display
    - LEDs (status indication)
    - Resistors for voltage dividers
    - Breadboard and jumper wires
    - USB cable and power supply
- **Software/Tools**
    - Arduino IDE / MicroPython (for ESP32 programming)
    - Flask (Python framework for web server)
    - HTML/CSS/JavaScript for dashboard visualization
    - Serial Monitor (for debugging)

4. Procedure

1. Connected 3 Li-ion cells in series through a 3S BMS module to form a 12.6V pack.

2. Configured ESP32 ADC pins with resistor dividers to measure individual cell voltages (B–, B1, B2, B+).
3. Programmed ESP32 to calculate pack voltage, individual cell percentages, and status (charging/discharging).
4. Established communication between ESP32 and Flask server to send live voltage data.
5. Developed a Flask web dashboard to display cell voltages, pack voltage, and battery percentage.

6. **Note:** Charging tests were not fully conducted because a dedicated 3S charger module was unavailable. Only monitoring functionality was tested.

6.Discussion/Analysis

- The system can monitor individual cell voltages, pack voltage, and status in real-time.
- Without a proper 3S charger, balancing and safe charging were not tested.
- Possible errors include ADC scaling inaccuracies and voltage divider drift.
- Improvement: Obtain a 3S charger module to complete charging tests and validate full system functionality.

7. References

- ESP32 Technical Reference Manual
- Flask Documentation: https://flask.palletsprojects.com/
- TP4056 datasheet
- Li-ion Battery Management Guidelines

Lab3 learning reflection or comments:

This week's labs provided a valuable hands-on learning experience, allowing me to connect theoretical knowledge with practical implementation. The first project, designing a battery monitoring system using an ESP32 and a Flask server, was both challenging and rewarding.

Initially, setting up the Flask server and integrating it with the ESP32 seemed complex, particularly ensuring that the ESP32 could reliably send battery voltage data to my local server. However, working through the debugging process and writing code in MicroPython enhanced my confidence in using both hardware and software tools. I found it especially satisfying to create a real-time monitoring dashboard, which allowed me to visualize battery levels conveniently without relying on external platforms like ThingSpeak. This approach also gave me more flexibility to integrate additional features in the future, such as notifications or historical logging.

Lab4 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. Present all the processes and outcomes from LAB1 to LAB2.

- **Link: https://youtu.be/BThtYojPPF0**

THANK YOU