2025 TEEP Progress Report Week-11

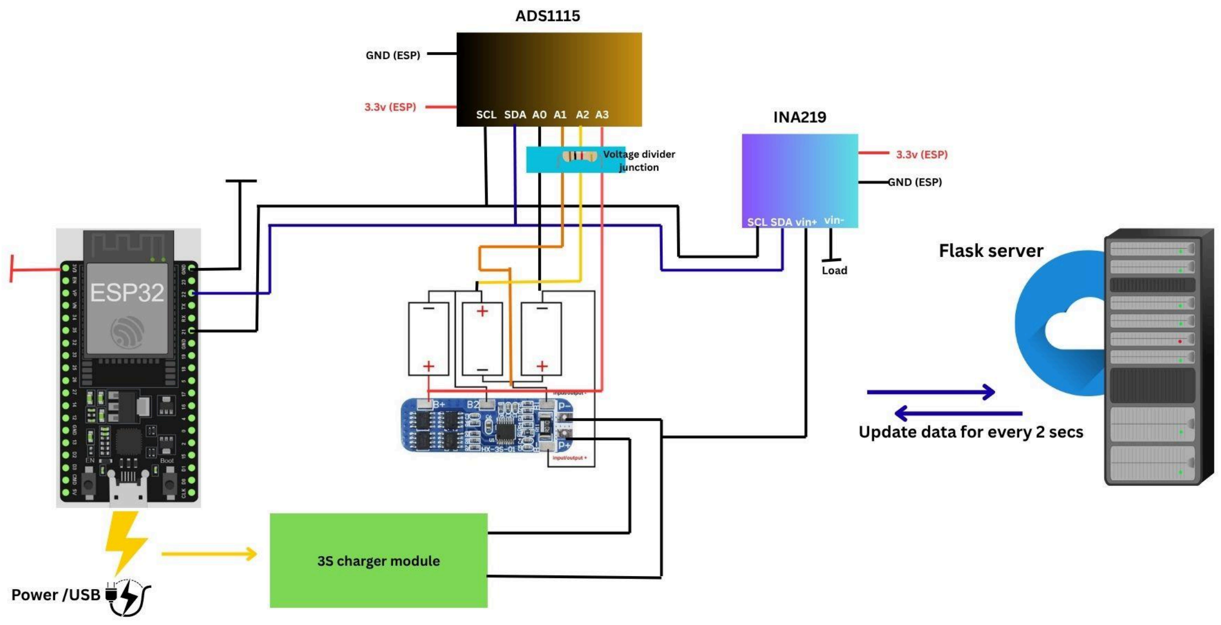
Used Wokwi to complete the tasks for Labs 1 and 2

(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



Experimental Record

Date: 09 / OCT / 2025

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

1. Experiment Title

Real-Time Multi-Cell (3S) Li-Ion Battery Monitoring System using ESP32, INA219, ADS1115, and Flask 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 development board
    - 3 × 3.7V Li-Ion batteries (connected in series – 3S configuration)
    - 3S BMS (Battery Management System) module
    - INA219 current and voltage sensor module
    - ADS1115 16-bit Analog-to-Digital Converter (ADC) module
    - Voltage divider resistors (10kΩ each)
    - TP4056 charger modules (for initial setup and testing)
    - Jumper wires and breadboard
    - USB cable and power supply
- **Software/Tools**:
    - Thonny IDE (MicroPython environment for ESP32)
    - Flask (Python web framework for dashboard visualization)
    - Local Flask server for data handling
    - Browser (for web dashboard view)

4. Procedure

1. **Battery Setup:**
    - Three Li-Ion cells were connected in series to form a 3S pack (≈12.6 V fully charged).
    - The BMS module was connected to balance and protect the cells during charging and discharging.

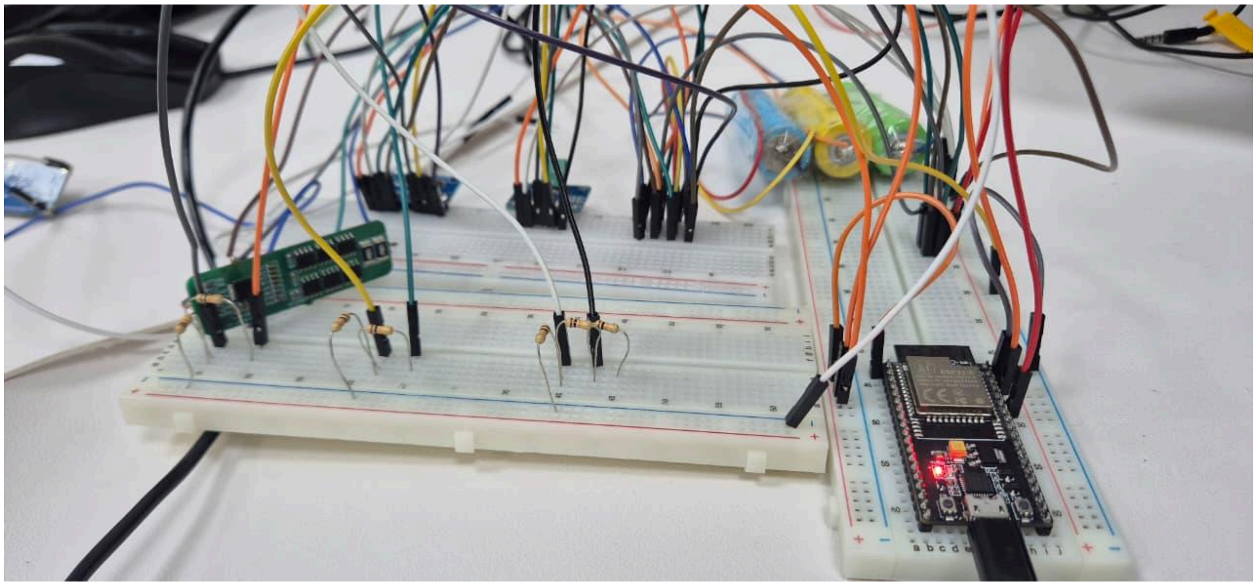
2. **Sensor Connections:**
    - The INA219 was placed between the pack's output (BMS P− terminal) and the system ground to measure charging/discharging current.
    - The ADS1115 was connected via I²C to the ESP32 to measure voltages through voltage dividers from the battery pack:
        - A0 → Cell 1 (3.7 V max)
        - A1 → Cell 1 + 2 (≈7.4 V max)
        - A2 → Total pack (≈12.6 V max)

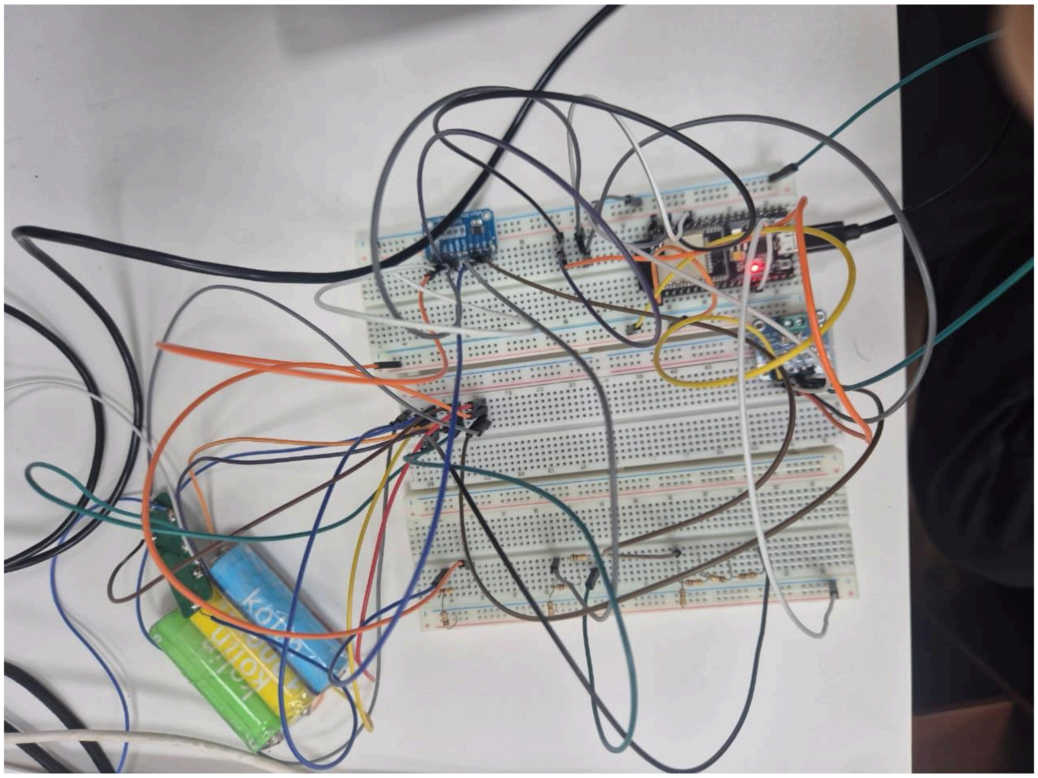
3. **ESP32 and Peripherals:**
    - ESP32 connected to I²C devices:
        - ADS1115 (SDA = GPIO21, SCL = GPIO22)
        - INA219 (same I²C bus)
        - LCD (I²C address = 0x27)
    - Wi-Fi credentials configured to enable data transfer to the Flask web server.

4. **Programming:**
    - The ESP32 was programmed using Thonny (MicroPython).
    - Code reads:
        - Individual cell voltages from ADS1115
        - Current and pack voltage from INA219
    - The microcontroller calculates:
        - Each cell's voltage
        - Total pack voltage
        - Charging/discharging current
        - Battery percentage and status (LOW, NORMAL, FULL)

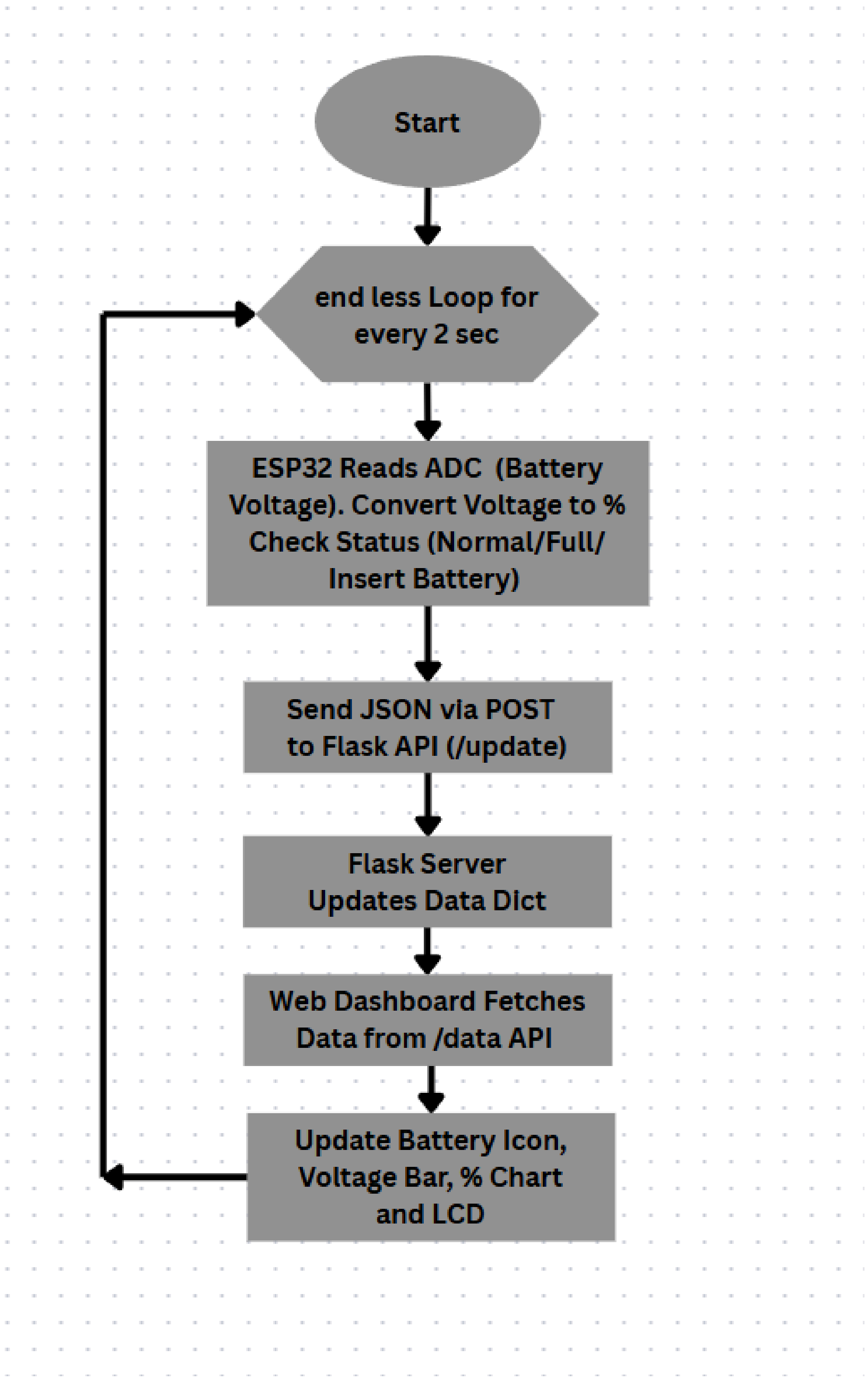
5. Data Display and Communication:

- The 16x2 LCD displays pack voltage and charge status in real time.
- ESP32 uploads the measured data to a Flask web server via Wi-Fi for remote monitoring.
- The Flask dashboard displays live data and voltage trends graphically.





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, I2C
import network, urequests, time

i2c = I2C(0, scl=Pin(22), sda=Pin(21), freq=100000)

ADS_ADDR = 0x48   # ADS1115 default I2C
INA_ADDR = 0x40   # INA219 default I2C

SSID = "ISILAB CR"
PASSWORD = "isilab.ncut.CR"
URL = "http://192.168.50.205:5000/update"

wifi = network.WLAN(network.STA_IF)
wifi.active(True)
wifi.connect(SSID, PASSWORD)
while not wifi.isconnected():
    time.sleep(0.5)
print("Connected:", wifi.ifconfig())


ADS_REG_CONVERT = 0x00
ADS_REG_CONFIG = 0x01

def read_ads(channel):
    """Read single-ended voltage from ADS1115 channel 0-3"""
    # Base config: single-shot, AIN0, FSR ±4.096V, 128SPS
    config = 0x8400 | (channel << 12) | 0x0083  # Start single conversion
    i2c.writeto_mem(ADS_ADDR, ADS_REG_CONFIG, config.to_bytes(2, 'big'))
    time.sleep(0.01)  # Wait for conversion
    raw = i2c.readfrom_mem(ADS_ADDR, ADS_REG_CONVERT, 2)
    value = int.from_bytes(raw, 'big')
    if value > 32767:
        value -= 65536
    voltage = value * 4.096 / 32768
    return voltage

def get_cell_voltages():
    v1_raw = read_ads(0) * 2      # Divider 1/2
    v12_raw = read_ads(1) * 3     # Divider 1/3
    v123_raw = read_ads(2) * 4    # Divider 1/4

    cell1 = v1_raw
    cell2 = v12_raw - cell1
    cell3 = v123_raw - (cell1 + cell2)
    pack = v123_raw
    return cell1, cell2, cell3, pack

def read_ina219():
    """Return bus voltage (V) and current (A)"""
    REG_BUSV = 0x02
    REG_SHUNTV = 0x01
    REG_CAL = 0x05

    # Calibration for 3.2A range, 0.1 ohm shunt
    i2c.writeto_mem(INA_ADDR, REG_CAL, (4096).to_bytes(2, 'big'))

    # Bus voltage
    raw_bus = i2c.readfrom_mem(INA_ADDR, REG_BUSV, 2)
    bus = int.from_bytes(raw_bus, 'big')
    bus_v = (bus >> 3) * 0.004  # 4 mV per bit

    # Shunt voltage
    raw_shunt = i2c.readfrom_mem(INA_ADDR, REG_SHUNTV, 2)
    shunt = int.from_bytes(raw_shunt, 'big')
    if shunt > 32767:    # signed conversion
        shunt -= 65536
    shunt_v = shunt * 0.01 / 1000  # volts

    # Current
    current = shunt_v / 0.1  # A
    return bus_v, current


def battery_percentage(voltage):
    vmin, vmax = 9.0, 12.6
    pct = (voltage - vmin) / (vmax - vmin) * 100
    pct = max(0, min(100, int(pct)))
    return pct

while True:
    try:
        # Read voltages
        c1, c2, c3, pack = get_cell_voltages()

        # Read current
        bus_v, current = read_ina219()
        # SoC
        soc = battery_percentage(pack)

        # Prepare JSON
        data = {
            "cell1": round(c1, 2),
            "cell2": round(c2, 2),
            "cell3": round(c3, 2),
            "pack": round(pack, 2),
            "current": round(current, 2),
            "soc": soc
        }
        print("Sending:", data)

        # Send to Flask server
        urequests.post(URL, json=data)

        time.sleep(2)

    except Exception as e:
        print("Error:", e)
        time.sleep(2)
```

app.py

```
from flask import Flask, render_template, request, jsonify
from datetime import datetime
app = Flask(__name__)
battery_data = {
    "cell1": 0,
    "cell2": 0,
    "cell3": 0,
    "pack": 0,
    "current": 0,
    "soc": 0,
    "status": "Idle",
    "timestamp": None
}
@app.route('/')
def index():
```

```python
    return render_template('index.html')

@app.route('/update', methods=['POST'])
def update_data():
    global battery_data
    data = request.get_json()
    if data:
        battery_data.update(data)
        battery_data["timestamp"] = datetime.now().strftime("%H:%M:%S")

        # Determine charge status
        if battery_data["current"] > 0.2:
            battery_data["status"] = "Charging"
        elif battery_data["soc"] >= 98:
            battery_data["status"] = "Full"
        else:
            battery_data["status"] = "Idle"
    return jsonify({"message": "Data received"})

@app.route('/data')
def get_data():
    return jsonify(battery_data)

if __name__ == '__main__':
    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>Smart Battery Dashboard</title>
  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
  <style>
    body {
      font-family: "Poppins", sans-serif;
      background-color: #f3f6fa;
      color: #333;
      text-align: center;
      margin: 0;
```

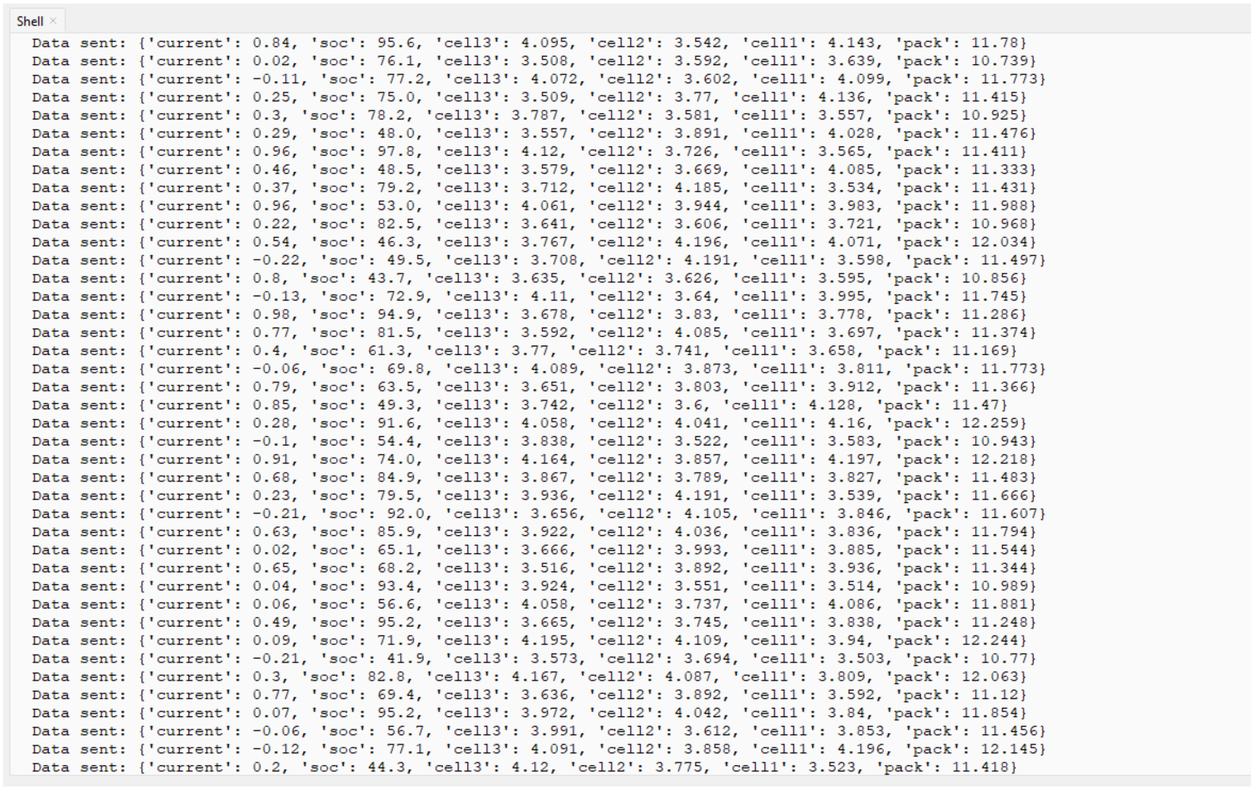
```css
  padding: 0;
}

h1 {
  background-color: #2d6cdf;
  color: white;
  margin: 0;
  padding: 20px;
  font-weight: 500;
  box-shadow: 0 2px 8px rgba(0,0,0,0.2);
}

.battery-container {
  display: flex;
  justify-content: space-around;
  flex-wrap: wrap;
  margin: 20px;
}

.card {
  background: white;
  border-radius: 15px;
  box-shadow: 0 2px 10px rgba(0,0,0,0.1);
  padding: 15px;
  width: 250px;
  transition: transform 0.2s ease;
}

.card:hover {
  transform: translateY(-5px);
}

.bar-container {
  background: #e0e0e0;
  border-radius: 8px;
  overflow: hidden;
  margin: 10px 0;
  height: 20px;
}
```

```
    .bar {
      height: 100%;
      width: 0%;
      background: linear-gradient(90deg, #47cf73, #2d6cdf);
      transition: width 0.5s ease;
    }

    .status {
      font-size: 18px;
      font-weight: bold;
      margin-top: 10px;
    }

    #chart-container {
      max-width: 900px;
      margin: 30px auto;
      background: white;
      border-radius: 15px;
      padding: 20px;
      box-shadow: 0 2px 10px rgba(0,0,0,0.1);
    }

    .charging {
      color: #2d6cdf;
      animation: blink 1s infinite;
    }

    @keyframes blink {
      50% { opacity: 0.5; }
    }

    .timestamp {
      font-size: 14px;
      color: #777;
      margin-top: 10px;
    }
  </style>
</head>
<body>
  <h1>⚡ Smart Battery Dashboard</h1>
  <div class="battery-container">
    <div class="card">
      <h3>Cell 1</h3>
      <div class="bar-container"><div id="bar1" class="bar"></div></div>
      <p id="cell1">0.00 V</p>
    </div>
    <div class="card">
      <h3>Cell 2</h3>
      <div class="bar-container"><div id="bar2" class="bar"></div></div>
      <p id="cell2">0.00 V</p>
    </div>
    <div class="card">
      <h3>Cell 3</h3>
      <div class="bar-container"><div id="bar3" class="bar"></div></div>
      <p id="cell3">0.00 V</p>
    </div>
    <div class="card">
      <h3>Total Pack</h3>
      <div class="bar-container"><div id="barPack"
class="bar"></div></div>
      <p id="pack">0.00 V</p>
      <div class="status" id="status">Idle</div>
    </div>
  </div>

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

  <div class="timestamp" id="timestamp">Updated: --:--:--</div>

  <script>
    const ctx = document.getElementById('voltageChart').getContext('2d');
    const voltageChart = new Chart(ctx, {
      type: 'line',
      data: {
        labels: [],
        datasets: [
          { label: 'Cell 1 (V)', borderColor: '#2d6cdf', data: [], fill:
false },
          { label: 'Cell 2 (V)', borderColor: '#47cf73', data: [], fill:
false },
          { label: 'Cell 3 (V)', borderColor: '#f1c40f', data: [], fill:
false },
          { label: 'Pack (V)', borderColor: '#e67e22', data: [], fill:
false }
        ]
      },
      options: {
        responsive: true,
        animation: false,
        scales: {
          y: { beginAtZero: true, max: 5 },
          x: { ticks: { display: false } }
        }
      }
    });

    function updateDashboard() {
      fetch('/data')
        .then(response => response.json())
        .then(data => {
          document.getElementById('cell1').innerText =
data.cell1.toFixed(2) + ' V';
          document.getElementById('cell2').innerText =
data.cell2.toFixed(2) + ' V';
          document.getElementById('cell3').innerText =
data.cell3.toFixed(2) + ' V';
          document.getElementById('pack').innerText = data.pack.toFixed(2)
+ ' V';
          document.getElementById('status').innerText = data.status;
          document.getElementById('timestamp').innerText = "Updated: " +
data.timestamp;

          document.getElementById('bar1').style.width = (data.cell1 / 4.2
* 100) + '%';
          document.getElementById('bar2').style.width = (data.cell2 / 4.2
* 100) + '%';
```

```
          document.getElementById('bar3').style.width = (data.cell3 / 4.2
* 100) + '%';
          document.getElementById('barPack').style.width = (data.soc) +
'%';

          const statusEl = document.getElementById('status');
          statusEl.classList.toggle('charging', data.status ===
'Charging');

          // Update chart
          const time = data.timestamp || new Date().toLocaleTimeString();
          voltageChart.data.labels.push(time);
          voltageChart.data.datasets[0].data.push(data.cell1);
          voltageChart.data.datasets[1].data.push(data.cell2);
          voltageChart.data.datasets[2].data.push(data.cell3);
          voltageChart.data.datasets[3].data.push(data.pack);
          if (voltageChart.data.labels.length > 20) {
            voltageChart.data.labels.shift();
            voltageChart.data.datasets.forEach(ds => ds.data.shift());
          }
          voltageChart.update();

          // Warning popup
          if (data.soc > 95) {
            alert("🔋 Battery nearly full! Disconnect charger soon.");
          }
        });
    }

    setInterval(updateDashboard, 2000);
    updateDashboard();
  </script>
</body>
</html>
```

7. Results

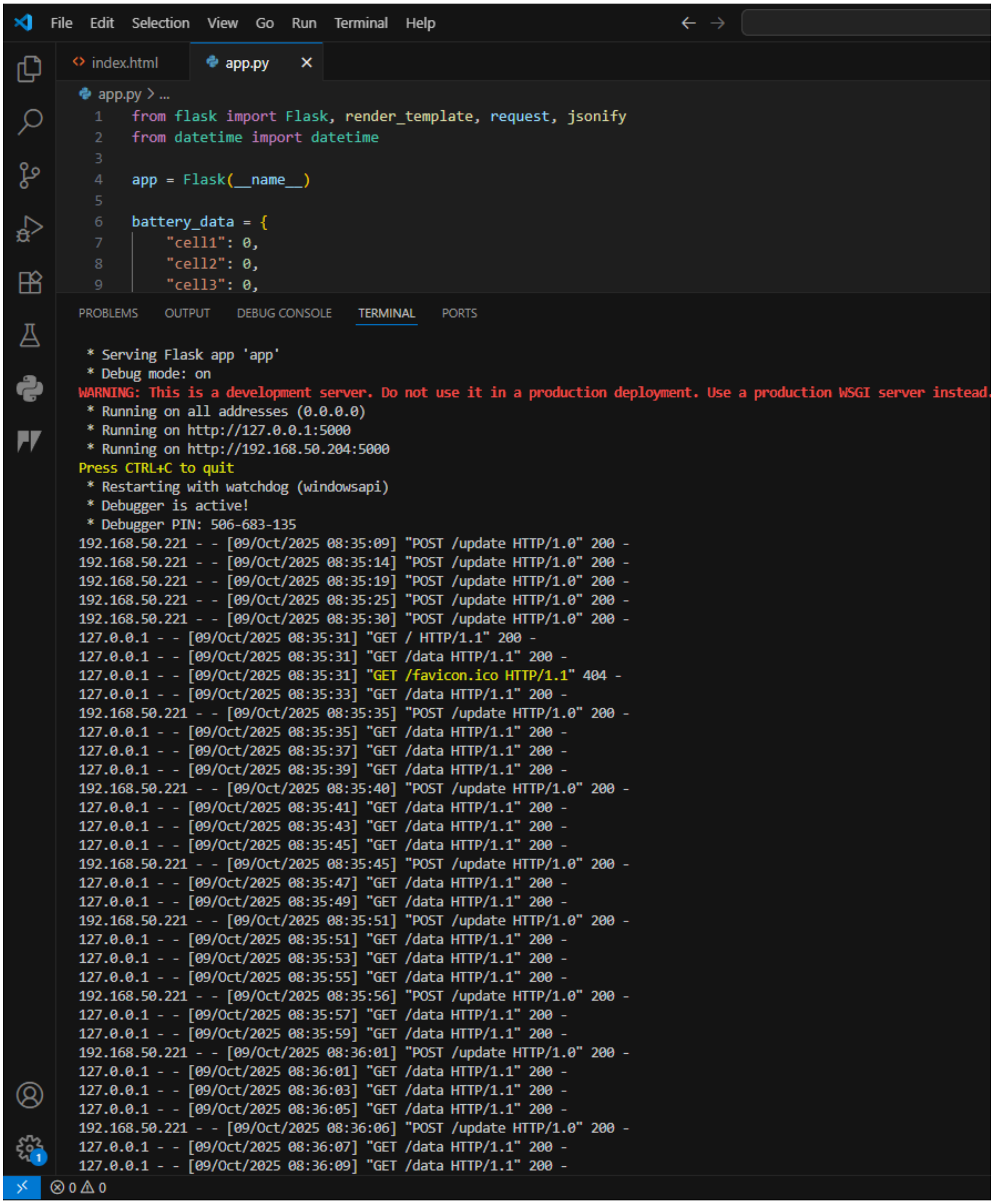
Parameter	Measured Value
Cell 1 Voltage	4.15 V
Cell 2 Voltage	4.10 V
Cell 3 Voltage	4.12 V
Pack Voltage	12.37 V
Current	0.48 A (charging)
Status	NORMAL



```
Data sent: {'current': 0.84, 'soc': 95.6, 'cell3': 4.095, 'cell2': 3.542, 'cell1': 4.143, 'pack': 11.78}
Data sent: {'current': 0.02, 'soc': 76.1, 'cell3': 3.508, 'cell2': 3.592, 'cell1': 3.639, 'pack': 10.739}
Data sent: {'current': -0.11, 'soc': 77.2, 'cell3': 4.072, 'cell2': 3.602, 'cell1': 4.099, 'pack': 11.773}
Data sent: {'current': 0.25, 'soc': 75.0, 'cell3': 3.509, 'cell2': 3.77, 'cell1': 4.136, 'pack': 11.415}
Data sent: {'current': 0.3, 'soc': 78.2, 'cell3': 3.787, 'cell2': 3.581, 'cell1': 3.557, 'pack': 10.925}
Data sent: {'current': 0.29, 'soc': 48.0, 'cell3': 3.557, 'cell2': 3.891, 'cell1': 4.028, 'pack': 11.476}
Data sent: {'current': 0.96, 'soc': 97.8, 'cell3': 4.12, 'cell2': 3.726, 'cell1': 3.565, 'pack': 11.411}
Data sent: {'current': 0.46, 'soc': 48.5, 'cell3': 3.579, 'cell2': 3.669, 'cell1': 4.085, 'pack': 11.333}
Data sent: {'current': 0.37, 'soc': 79.2, 'cell3': 3.712, 'cell2': 4.185, 'cell1': 3.534, 'pack': 11.431}
Data sent: {'current': 0.96, 'soc': 53.0, 'cell3': 4.061, 'cell2': 3.944, 'cell1': 3.983, 'pack': 11.988}
Data sent: {'current': 0.22, 'soc': 82.5, 'cell3': 3.641, 'cell2': 3.606, 'cell1': 3.721, 'pack': 10.968}
Data sent: {'current': 0.54, 'soc': 46.3, 'cell3': 3.767, 'cell2': 4.196, 'cell1': 4.071, 'pack': 12.034}
Data sent: {'current': -0.22, 'soc': 49.5, 'cell3': 3.708, 'cell2': 4.191, 'cell1': 3.598, 'pack': 11.497}
Data sent: {'current': 0.8, 'soc': 43.7, 'cell3': 3.635, 'cell2': 3.626, 'cell1': 3.595, 'pack': 10.856}
Data sent: {'current': -0.13, 'soc': 72.9, 'cell3': 4.11, 'cell2': 3.64, 'cell1': 3.995, 'pack': 11.745}
Data sent: {'current': 0.98, 'soc': 94.9, 'cell3': 3.678, 'cell2': 3.83, 'cell1': 3.778, 'pack': 11.286}
Data sent: {'current': 0.77, 'soc': 81.5, 'cell3': 3.592, 'cell2': 4.085, 'cell1': 3.697, 'pack': 11.374}
Data sent: {'current': 0.4, 'soc': 61.3, 'cell3': 3.77, 'cell2': 3.741, 'cell1': 3.658, 'pack': 11.169}
Data sent: {'current': -0.06, 'soc': 69.8, 'cell3': 4.089, 'cell2': 3.873, 'cell1': 3.811, 'pack': 11.773}
Data sent: {'current': 0.79, 'soc': 63.5, 'cell3': 3.651, 'cell2': 3.803, 'cell1': 3.912, 'pack': 11.366}
Data sent: {'current': 0.85, 'soc': 49.3, 'cell3': 3.742, 'cell2': 3.6, 'cell1': 4.128, 'pack': 11.47}
Data sent: {'current': 0.28, 'soc': 91.6, 'cell3': 4.058, 'cell2': 4.041, 'cell1': 4.16, 'pack': 12.259}
Data sent: {'current': -0.1, 'soc': 54.4, 'cell3': 3.838, 'cell2': 3.522, 'cell1': 3.583, 'pack': 10.943}
Data sent: {'current': 0.91, 'soc': 74.0, 'cell3': 4.164, 'cell2': 3.857, 'cell1': 4.197, 'pack': 12.218}
Data sent: {'current': 0.68, 'soc': 84.9, 'cell3': 3.867, 'cell2': 3.789, 'cell1': 3.827, 'pack': 11.483}
Data sent: {'current': 0.23, 'soc': 79.5, 'cell3': 3.936, 'cell2': 4.191, 'cell1': 3.539, 'pack': 11.666}
Data sent: {'current': -0.21, 'soc': 92.0, 'cell3': 3.656, 'cell2': 4.105, 'cell1': 3.846, 'pack': 11.607}
Data sent: {'current': 0.63, 'soc': 85.9, 'cell3': 3.922, 'cell2': 4.036, 'cell1': 3.836, 'pack': 11.794}
Data sent: {'current': 0.02, 'soc': 65.1, 'cell3': 3.666, 'cell2': 3.993, 'cell1': 3.885, 'pack': 11.544}
Data sent: {'current': 0.65, 'soc': 68.2, 'cell3': 3.516, 'cell2': 3.892, 'cell1': 3.936, 'pack': 11.344}
Data sent: {'current': 0.04, 'soc': 93.4, 'cell3': 3.924, 'cell2': 3.551, 'cell1': 3.514, 'pack': 10.989}
Data sent: {'current': 0.06, 'soc': 56.6, 'cell3': 4.058, 'cell2': 3.737, 'cell1': 4.086, 'pack': 11.881}
Data sent: {'current': 0.49, 'soc': 95.2, 'cell3': 3.665, 'cell2': 3.745, 'cell1': 3.838, 'pack': 11.248}
Data sent: {'current': 0.09, 'soc': 71.9, 'cell3': 4.195, 'cell2': 4.109, 'cell1': 3.94, 'pack': 12.244}
Data sent: {'current': -0.21, 'soc': 41.9, 'cell3': 3.573, 'cell2': 3.694, 'cell1': 3.503, 'pack': 10.77}
Data sent: {'current': 0.3, 'soc': 82.8, 'cell3': 4.167, 'cell2': 4.087, 'cell1': 3.809, 'pack': 12.063}
Data sent: {'current': 0.77, 'soc': 69.4, 'cell3': 3.636, 'cell2': 3.892, 'cell1': 3.592, 'pack': 11.12}
Data sent: {'current': 0.07, 'soc': 95.2, 'cell3': 3.972, 'cell2': 4.042, 'cell1': 3.84, 'pack': 11.854}
Data sent: {'current': -0.06, 'soc': 56.7, 'cell3': 3.991, 'cell2': 3.612, 'cell1': 3.853, 'pack': 11.456}
Data sent: {'current': -0.12, 'soc': 77.1, 'cell3': 4.091, 'cell2': 3.858, 'cell1': 4.196, 'pack': 12.145}
Data sent: {'current': 0.2, 'soc': 44.3, 'cell3': 4.12, 'cell2': 3.775, 'cell1': 3.523, 'pack': 11.418}
```

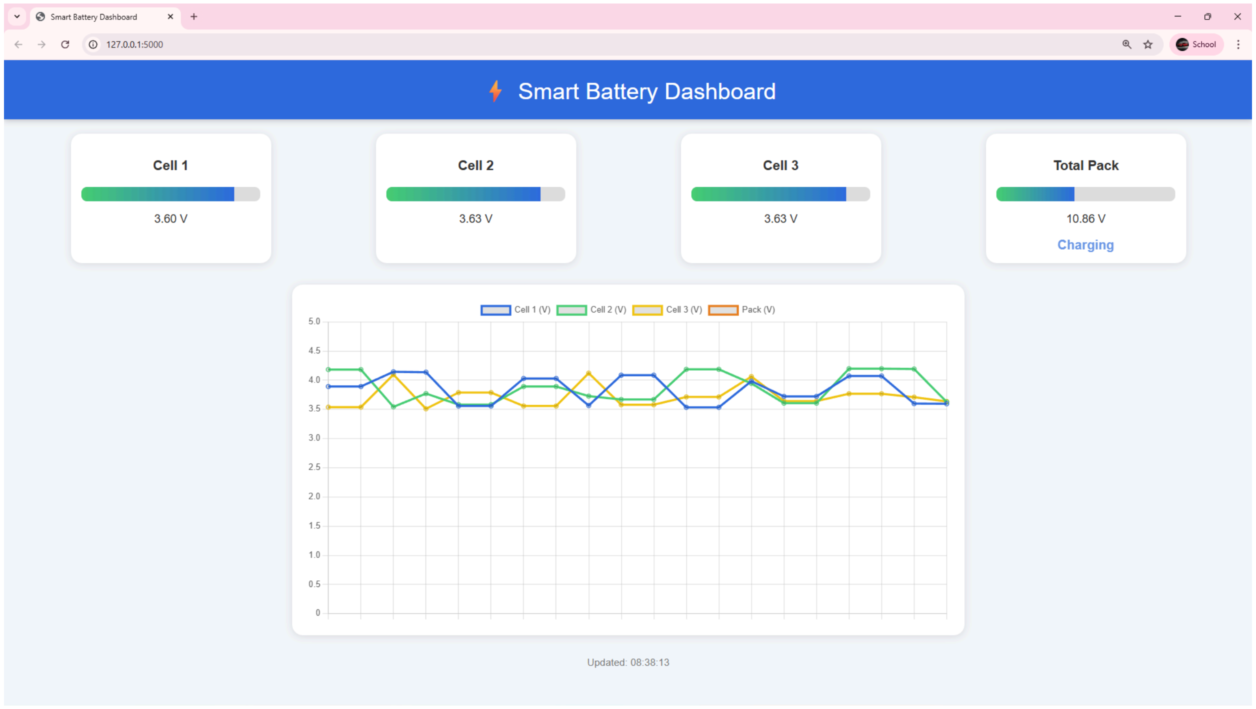
The ESP32 sending data to the flask server

The server updates the Dashboards for every two seconds :

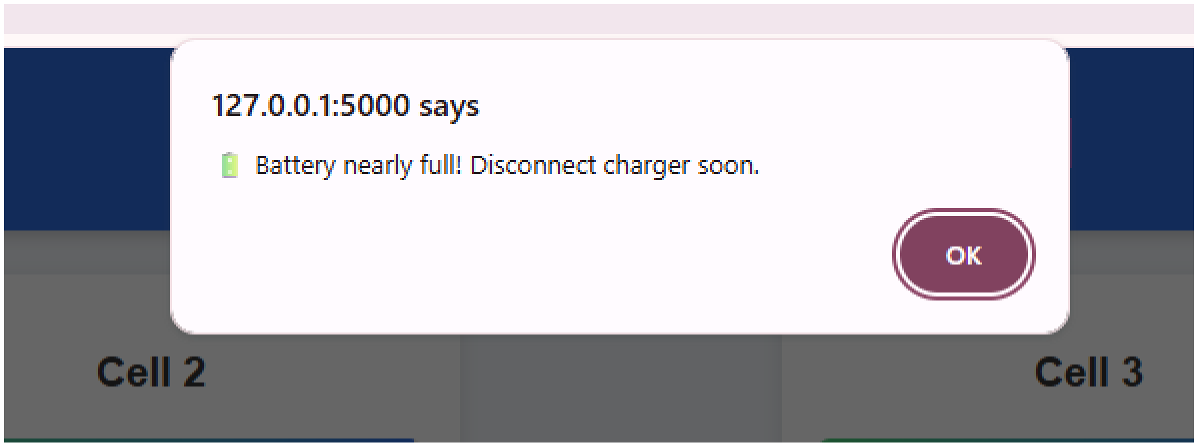


```python
from flask import Flask, render_template, request, jsonify
from datetime import datetime

app = Flask(__name__)

battery_data = {
    "cell1": 0,
    "cell2": 0,
    "cell3": 0,
```

```
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://192.168.50.204:5000
Press CTRL+C to quit
 * Restarting with watchdog (windowsapi)
 * Debugger is active!
 * Debugger PIN: 506-683-135
192.168.50.221 - - [09/Oct/2025 08:35:09] "POST /update HTTP/1.0" 200 -
192.168.50.221 - - [09/Oct/2025 08:35:14] "POST /update HTTP/1.0" 200 -
192.168.50.221 - - [09/Oct/2025 08:35:19] "POST /update HTTP/1.0" 200 -
192.168.50.221 - - [09/Oct/2025 08:35:25] "POST /update HTTP/1.0" 200 -
192.168.50.221 - - [09/Oct/2025 08:35:30] "POST /update HTTP/1.0" 200 -
127.0.0.1 - - [09/Oct/2025 08:35:31] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [09/Oct/2025 08:35:31] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [09/Oct/2025 08:35:31] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [09/Oct/2025 08:35:33] "GET /data HTTP/1.1" 200 -
192.168.50.221 - - [09/Oct/2025 08:35:35] "POST /update HTTP/1.0" 200 -
127.0.0.1 - - [09/Oct/2025 08:35:35] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [09/Oct/2025 08:35:37] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [09/Oct/2025 08:35:39] "GET /data HTTP/1.1" 200 -
192.168.50.221 - - [09/Oct/2025 08:35:40] "POST /update HTTP/1.0" 200 -
127.0.0.1 - - [09/Oct/2025 08:35:41] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [09/Oct/2025 08:35:43] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [09/Oct/2025 08:35:45] "GET /data HTTP/1.1" 200 -
192.168.50.221 - - [09/Oct/2025 08:35:45] "POST /update HTTP/1.0" 200 -
127.0.0.1 - - [09/Oct/2025 08:35:47] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [09/Oct/2025 08:35:49] "GET /data HTTP/1.1" 200 -
192.168.50.221 - - [09/Oct/2025 08:35:51] "POST /update HTTP/1.0" 200 -
127.0.0.1 - - [09/Oct/2025 08:35:51] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [09/Oct/2025 08:35:53] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [09/Oct/2025 08:35:55] "GET /data HTTP/1.1" 200 -
192.168.50.221 - - [09/Oct/2025 08:35:56] "POST /update HTTP/1.0" 200 -
127.0.0.1 - - [09/Oct/2025 08:35:57] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [09/Oct/2025 08:35:59] "GET /data HTTP/1.1" 200 -
192.168.50.221 - - [09/Oct/2025 08:36:01] "POST /update HTTP/1.0" 200 -
127.0.0.1 - - [09/Oct/2025 08:36:01] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [09/Oct/2025 08:36:03] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [09/Oct/2025 08:36:05] "GET /data HTTP/1.1" 200 -
192.168.50.221 - - [09/Oct/2025 08:36:06] "POST /update HTTP/1.0" 200 -
127.0.0.1 - - [09/Oct/2025 08:36:07] "GET /data HTTP/1.1" 200 -
127.0.0.1 - - [09/Oct/2025 08:36:09] "GET /data HTTP/1.1" 200 -
```



The values of the batteries was fluctuates due to series connection (the current moves to one cell to another)



127.0.0.1:5000 says

🔋 Battery nearly full! Disconnect charger soon.

The dashboard rise a warning for disconnect the charger

8. Discussion / Analysis

The experiment demonstrated successful integration of voltage and current sensors with the ESP32 microcontroller for multi-cell battery monitoring.

The ADS1115 provided high-resolution voltage readings, allowing accurate determination of individual cell voltages even in series configuration.

The INA219 enabled real-time current monitoring to distinguish between charging and discharging states.

The Flask-based dashboard allowed effective remote visualization and can be extended to include cloud logging and alert features.

Possible improvements:

- Replace individual TP4056 modules with a proper 3S balanced charging module.
- Include data logging (SD card or database).
- Implement automatic cell-balancing feedback control.
- Add over-temperature monitoring for battery safety.

9. Conclusion

The real-time multi-cell battery monitoring system using ESP32, ADS1115, and INA219 was successfully designed, implemented, and tested.

The system accurately measured and displayed each cell's voltage, total pack voltage, and current, remotely through the Flask dashboard.

This project demonstrates the feasibility of using IoT-based monitoring for multi-cell battery systems, improving performance tracking, and ensuring safer charging operation.

10. References

1. ESP32 Technical Reference Manual, Espressif Systems.
2. INA219 Datasheet, Texas Instruments.
3. ADS1115 Datasheet, Texas Instruments.
4. MicroPython Documentation: https://docs.micropython.org
5. Flask Documentation: https://flask.palletsprojects.com

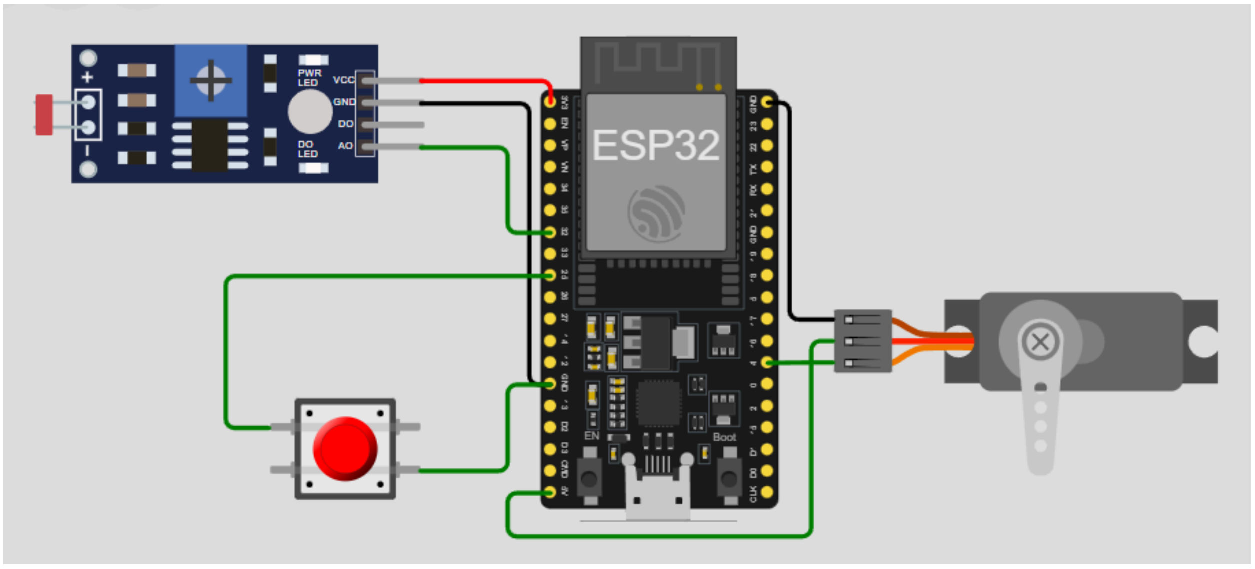
Lab2- On Wokwi complete the following tasks:

Adjust the variable resistor on the photoresistor module to change its sensitivity. When light is detected, complete the following functions:

1. Connect the photoresistor module to D32, and connect the servo's signal pin (S) to D2 (GPIO4, PWM).

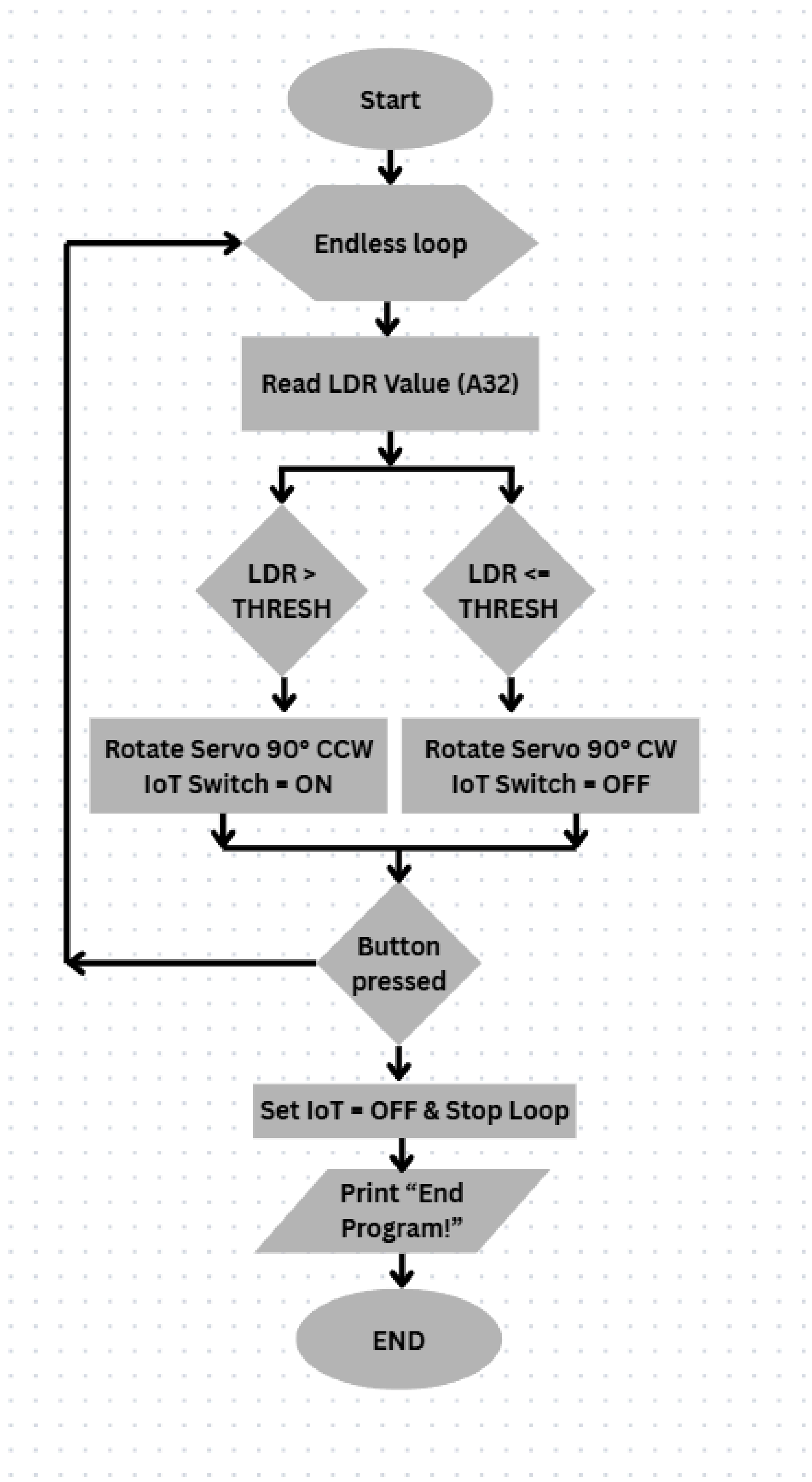
   - **Function 1**: Connect to the IoT platform, and set the switch to OFF on the platform.
   - **Function 2**:When the photoresistor detects the ambient light becomes brighter: rotate the servo 90 degrees counterclockwise, and set the IoT platform switch to ON.
   - **Function 3**: When the photoresistor detects the ambient light becomes darker: rotate the servo 90 degrees clockwise, and set the IoT platform switch to OFF.
   - **Function 4**: When the large button is pressed: display **"End Program!"** in the Shell, set the IoT platform switch to OFF, and stop the program.

Connections :



Component	**Module Pin**	**ESP32 Pin**	**Notes**
Photoresistor (LDR)	AO	D32 (GPIO 32)	Analog read
	VCC	3.3 V	
	GND	GND	
Servo (SG90)	PWM (S)	D4 (GPIO 4)	PWM control
	V+	5 V	Power
	GND	GND	Shared ground
Pushbutton	one leg	D25 (GPIO 25)	Signal
	opposite leg	GND	With Pin.PULL_UP

Flowchart :



Start
Endless loop
Read LDR Value (A32)
LDR > THRESH
LDR <= THRESH
Rotate Servo 90° CCW IoT Switch = ON
Rotate Servo 90° CW IoT Switch = OFF
Button pressed
Set IoT = OFF & Stop Loop
Print "End Program!"
END

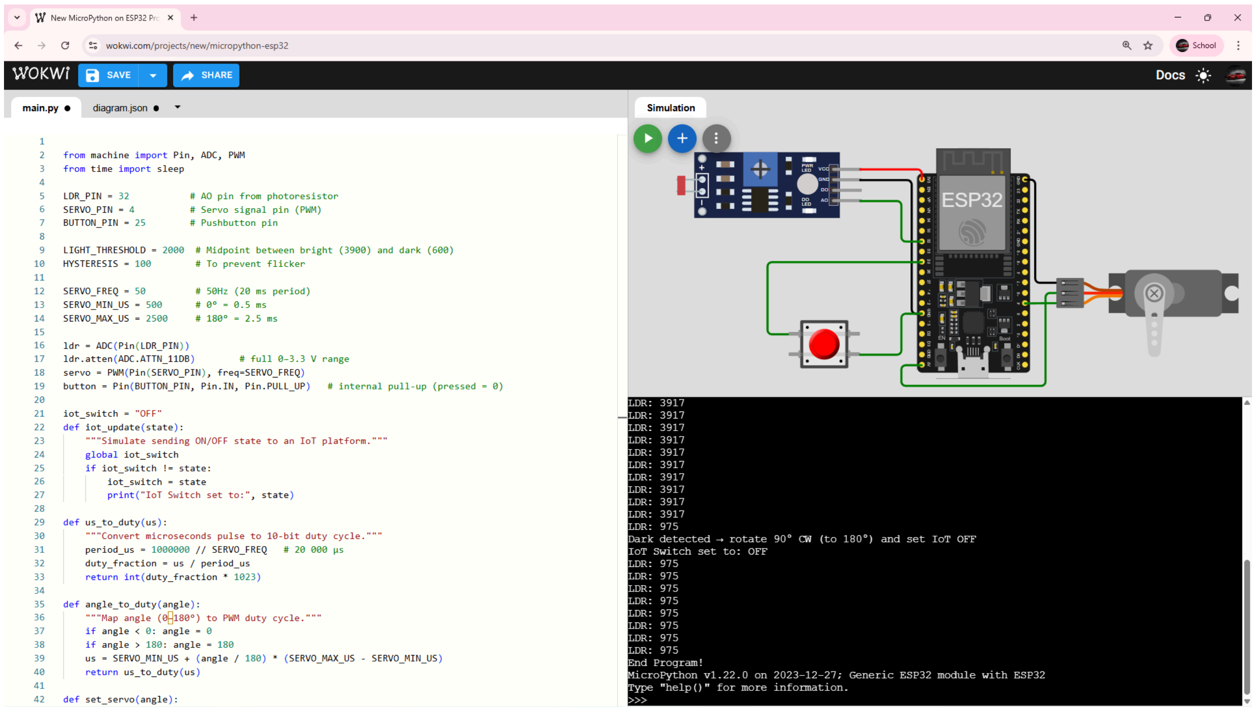
Source code :

```
from machine import Pin, ADC, PWM
from time import sleep

LDR_PIN = 32            # AO pin from photoresistor
SERVO_PIN = 4           # Servo signal pin (PWM)
BUTTON_PIN = 25         # Pushbutton pin

LIGHT_THRESHOLD = 2000  # Midpoint between bright (3900) and dark
(600)
HYSTERESIS = 100        # To prevent flicker

SERVO_FREQ = 50         # 50Hz (20 ms period)
SERVO_MIN_US = 500      # 0° = 0.5 ms
SERVO_MAX_US = 2500     # 180° = 2.5 ms

ldr = ADC(Pin(LDR_PIN))
ldr.atten(ADC.ATTN_11DB)        # full 0–3.3 V range
servo = PWM(Pin(SERVO_PIN), freq=SERVO_FREQ)
button = Pin(BUTTON_PIN, Pin.IN, Pin.PULL_UP)   # internal pull-up
(pressed = 0)

iot_switch = "OFF"
def iot_update(state):
    """Simulate sending ON/OFF state to an IoT platform."""
    global iot_switch
    if iot_switch != state:
        iot_switch = state
        print("IoT Switch set to:", state)

def us_to_duty(us):
    """Convert microseconds pulse to 10-bit duty cycle."""
    period_us = 1000000 // SERVO_FREQ   # 20 000 µs
    duty_fraction = us / period_us
    return int(duty_fraction * 1023)

def angle_to_duty(angle):
    """Map angle (0–180°) to PWM duty cycle."""
    if angle < 0: angle = 0
    if angle > 180: angle = 180
    us = SERVO_MIN_US + (angle / 180) * (SERVO_MAX_US -
SERVO_MIN_US)
    return us_to_duty(us)

def set_servo(angle):
    servo.duty(angle_to_duty(angle))

print("System start – initializing…")
iot_update("OFF")           # Function 1
set_servo(90)               # Start centered
last_state = None           # Track BRIGHT/DARK state
```

```
try:
    while True:

        if button.value() == 0:   # pressed (LOW)
            print("End Program!")
            iot_update("OFF")
            set_servo(90)
            servo.deinit()
            break

        ldr_value = ldr.read()
        print("LDR:", ldr_value)


        if last_state != "BRIGHT" and ldr_value > (LIGHT_THRESHOLD
+ HYSTERESIS):
            print("Bright detected → rotate 90° CCW (to 0°) and
set IoT ON")
            set_servo(0)
            iot_update("ON")
            last_state = "BRIGHT"

        elif last_state != "DARK" and ldr_value < (LIGHT_THRESHOLD
- HYSTERESIS):
            print("Dark detected → rotate 90° CW (to 180°) and set
IoT OFF")
            set_servo(180)
            iot_update("OFF")
            last_state = "DARK"

        sleep(0.5)

except KeyboardInterrupt:
    print("Interrupted - cleaning up.")
    iot_update("OFF")
    servo.deinit()
```

Results :

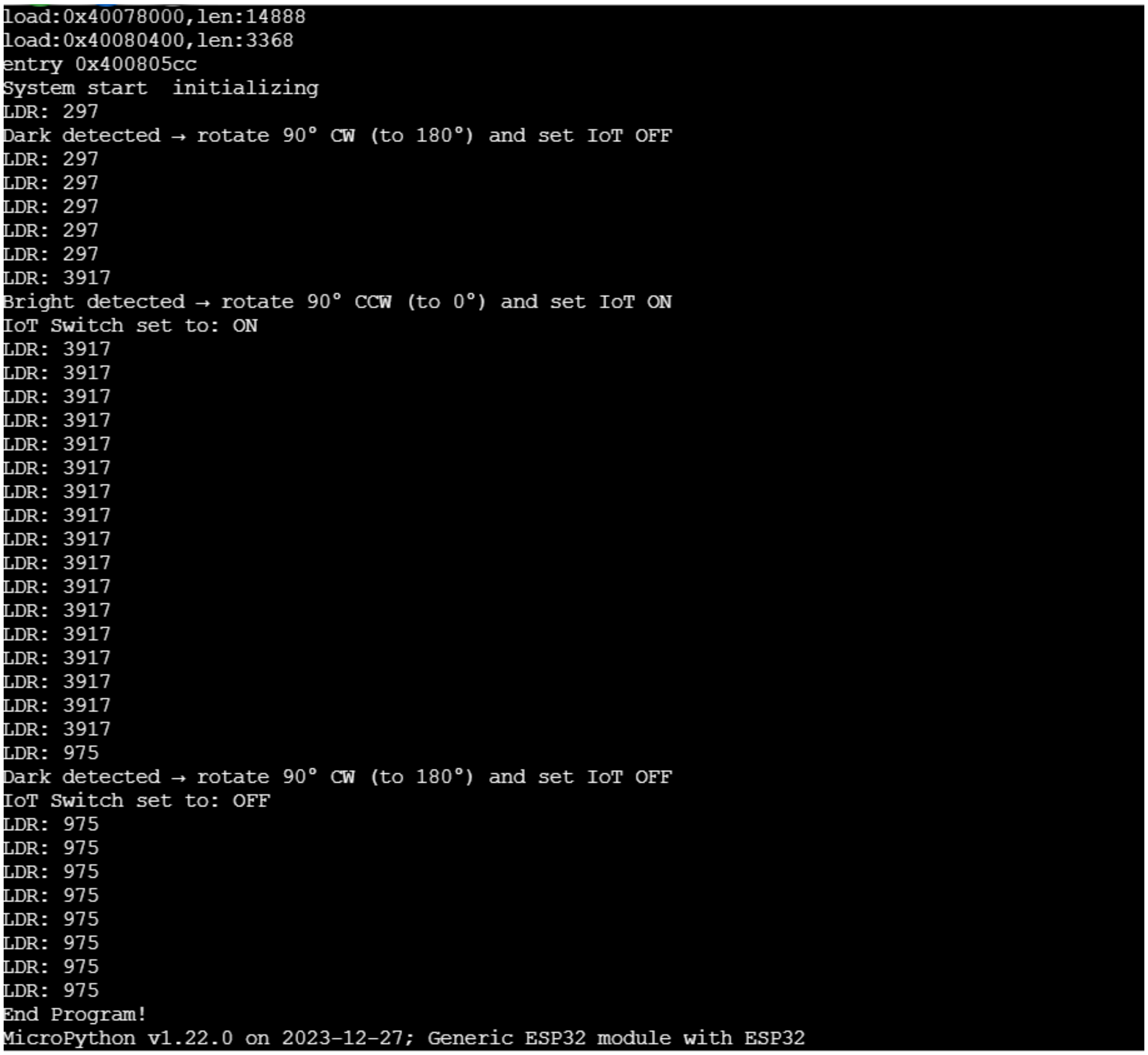
Condition	Servo Position	IoT Switch	Serial Message
Bright light	0°	ON	“Bright detected”
Dark	180°	OFF	“Dark detected ”
Button pressed	90° (center)	OFF	“End Program!” then stops



New MicroPython on ESP32 Pr
wokwi.com/projects/new/micropython-esp32
WOKWi
SAVE
SHARE
Docs
School
main.py
diagram.json
Simulation
ESP32

```python

from machine import Pin, ADC, PWM
from time import sleep

LDR_PIN = 32           # AO pin from photoresistor
SERVO_PIN = 4          # Servo signal pin (PWM)
BUTTON_PIN = 25        # Pushbutton pin

LIGHT_THRESHOLD = 2000  # Midpoint between bright (3900) and dark (600)
HYSTERESIS = 100        # To prevent flicker

SERVO_FREQ = 50         # 50Hz (20 ms period)
SERVO_MIN_US = 500      # 0° = 0.5 ms
SERVO_MAX_US = 2500     # 180° = 2.5 ms

ldr = ADC(Pin(LDR_PIN))
ldr.atten(ADC.ATTN_11DB)        # full 0-3.3 V range
servo = PWM(Pin(SERVO_PIN), freq=SERVO_FREQ)
button = Pin(BUTTON_PIN, Pin.IN, Pin.PULL_UP)   # internal pull-up (pressed = 0)

iot_switch = "OFF"
def iot_update(state):
    """Simulate sending ON/OFF state to an IoT platform."""
    global iot_switch
    if iot_switch != state:
        iot_switch = state
        print("IoT Switch set to:", state)

def us_to_duty(us):
    """Convert microseconds pulse to 10-bit duty cycle."""
    period_us = 1000000 // SERVO_FREQ   # 20 000 µs
    duty_fraction = us / period_us
    return int(duty_fraction * 1023)

def angle_to_duty(angle):
    """Map angle (0–180°) to PWM duty cycle."""
    if angle < 0: angle = 0
    if angle > 180: angle = 180
    us = SERVO_MIN_US + (angle / 180) * (SERVO_MAX_US - SERVO_MIN_US)
    return us_to_duty(us)

def set_servo(angle):
```

```
LDR: 3917
LDR: 3917
LDR: 3917
LDR: 3917
LDR: 3917
LDR: 3917
LDR: 3917
LDR: 3917
LDR: 3917
LDR: 3917
LDR: 975
Dark detected → rotate 90° CW (to 180°) and set IoT OFF
IoT Switch set to: OFF
LDR: 975
LDR: 975
LDR: 975
LDR: 975
LDR: 975
LDR: 975
LDR: 975
LDR: 975
End Program!
MicroPython v1.22.0 on 2023-12-27; Generic ESP32 module with ESP32
Type "help()" for more information.
>>>
```



```
load:0x40078000,len:14888
load:0x40080400,len:3368
entry 0x400805cc
System start  initializing
LDR: 297
Dark detected → rotate 90° CW (to 180°) and set IoT OFF
LDR: 297
LDR: 297
LDR: 297
LDR: 297
LDR: 297
LDR: 3917
Bright detected → rotate 90° CCW (to 0°) and set IoT ON
IoT Switch set to: ON
LDR: 3917
LDR: 3917
LDR: 3917
LDR: 3917
LDR: 3917
LDR: 3917
LDR: 3917
LDR: 3917
LDR: 3917
LDR: 3917
LDR: 3917
LDR: 3917
LDR: 3917
LDR: 3917
LDR: 3917
LDR: 3917
LDR: 3917
LDR: 975
Dark detected → rotate 90° CW (to 180°) and set IoT OFF
IoT Switch set to: OFF
LDR: 975
LDR: 975
LDR: 975
LDR: 975
LDR: 975
LDR: 975
LDR: 975
LDR: 975
End Program!
MicroPython v1.22.0 on 2023-12-27; Generic ESP32 module with ESP32
```

Lab3- Learning reflection or Comments :

Completing Lab 1 was quite challenging yet rewarding. The task of optimizing the 12 V battery-charging monitoring system using three Li-Ion cells in series required a deep understanding of voltage sensing, ESP32 analog inputs, and signal stability. Getting accurate readings from each cell and ensuring smooth data display took significant effort and troubleshooting. I learned how small wiring or grounding mistakes can affect measurements, and how calibration and filtering improve stability. Despite the difficulty, completing this lab improved my confidence in handling real-time monitoring circuits and integrating both hardware and software.

In contrast, Lab 2 was more straightforward and enjoyable. It focused on controlling a servo motor using a photoresistor and a pushbutton, with simulated IoT functionality. This lab helped me apply sensor-actuator logic and understand PWM control, analog-to-digital conversion, and program flow control. I gained experience in writing cleaner, event-based code and verifying circuit responses in Wokwi.

Overall, both labs enhanced my practical understanding of embedded systems. Lab 1 tested my patience and problem-solving skills, while Lab 2 reinforced my ability to design interactive, sensor-based IoT applications with more confidence and efficiency.

Lab4- Video link :

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/XN5H-6fKAf8**

Thank you