Proiect PR IoT
Sistem de casă inteligent pentru controlul intensității luminii dintr-o încăpere

Arpășanu Emilia-Oana
341C1

2 ianuarie 2025

1 Introducere

Proiectul are ca scop crearea unui rețele locale formată din dispozitive IoT capabilă să analizeze intensitatea luminii din mediul ambiant cu ajutorul unui fotorezistor. Aceasta prelucrează datele și le afișează într-o manieră intuitivă. În plus, în funcție de gradul intensității luminoase, se interacționează cu un actuator (bec LED inteligent), ce aduce nivelul intensității la cel dorit de către utilizator.

Informațiile din mediul ambiant sunt preluate de un microcontroller care le transmite mai departe serverului ce rulează pe laptop-ul personal, pentru simplitate și eficiență. Rezultatele ce îi sunt prezentate utilizatorului sunt actualizate în timp real, interfața dispunând de capacitatea controlului actuatorului în ceea ce privește intensitatea luminoasă și oprirea sau pornirea lui.

2 Arhitectură

2.1 Structura rețelei

Datele de la senzorul de lumină sunt preluate și prelucrate cu ajutorul unui microcontroller de tip ESP32, programat pentru simplitate în MicroPython. Acestea sunt trimise pe cale wireless către serverul central (care rulează pe laptop-ul personal) ce expune o interfață web utilizatorului pentru a vizualiza datele despre intensitatea luminii și a lua decizii asupra actuatorului (pornire/oprire, setare luminozitate, determinarea parametrilor acestuia, aplicarea modului twinkle). Actuatorul este și el controlat la rândul lui prin intermediul Wi-Fi. Toate aceste componente sunt conectate la aceeași rețea pentru a asigura buna funcționare, fiind necesare informațiile despre aceasta (SSID și parola pentru conectare).

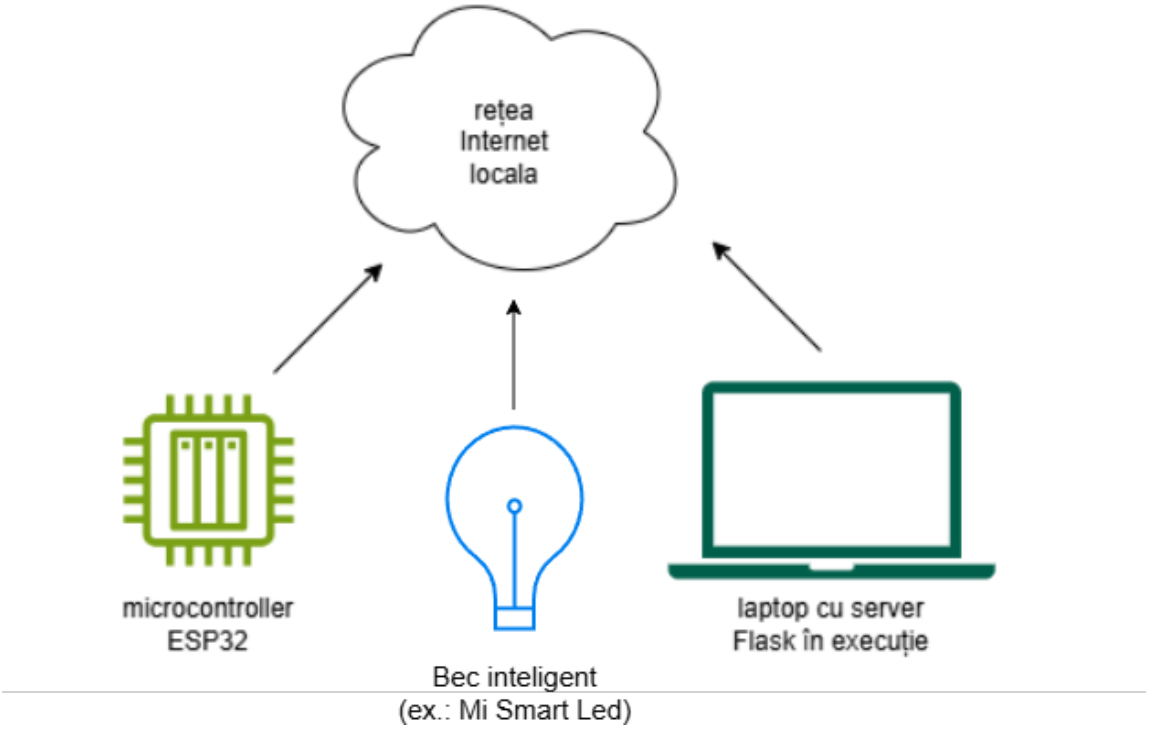

Figura 1: Diagrama rețelei

3 Implementare software și hardware

3.1 Componentele necesare

Pentru implementarea circuitului sunt necesare următoarele elemente:

- microcontroller de tip ESP32;
- rezistență 10kΩ;
- fotorezistență;
- breadboard (830 tie-points);
- 4 fire de legătură (male-to-male);
- cablu USB pentru alimentare.

3.2 Programarea microcontroller-ului

Pentru a putea citi datele de la senzor și a le trimite către server se execută un program scris în MicroPython pe microcontroller-ul de tip ESP32, ce îndeplinește următoarele roluri:

- se realizează conexiunea microcontroller-ului la rețeaua Wi-Fi locală;
- se setează pinul GPIO la care este conectat fotorezistorul (în acest caz pinul logic corespunde celui fizic denumit VN);
- se setează atenuarea valorilor citite cu ajutorul ADC-ului corespunzătoare tensiunii circuitului (în acest caz 6 dB);
- într-o buclă infinită se citește valoarea curentă de la senzor (care este normalizată pentru a fi în intervalul 0-100, precum luminozitatea becului), se construiește payload-ul ce o conține (obiect JSON), se trimite către server și se așteaptă 1 minut între citiri.

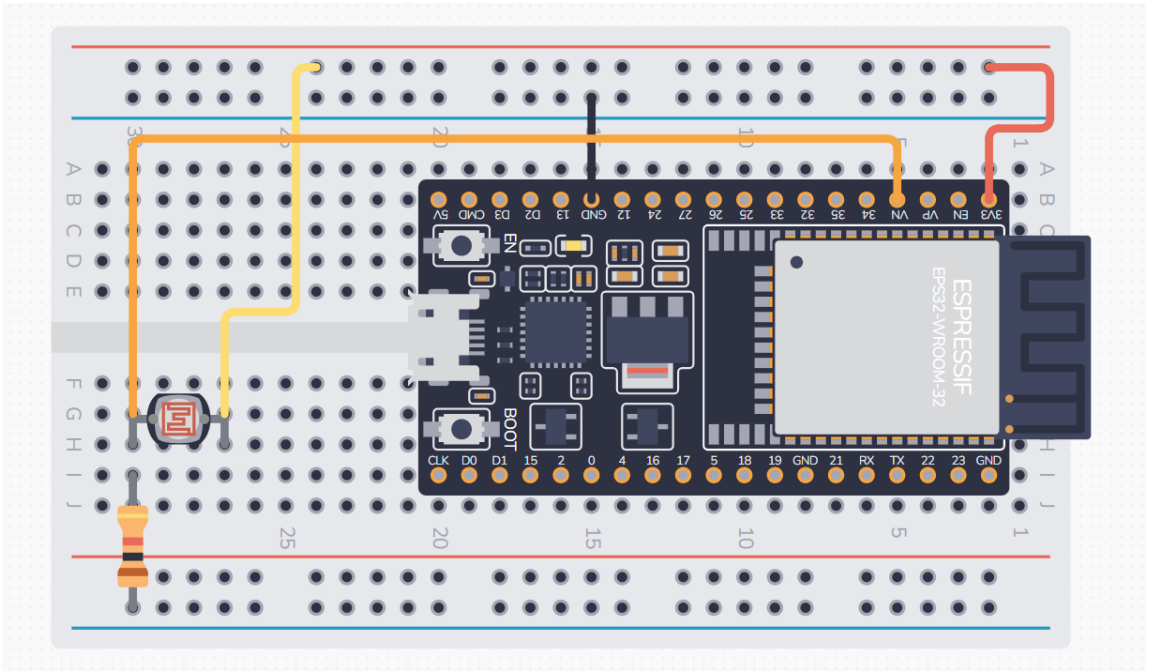

Figura 2: Schema circuitului

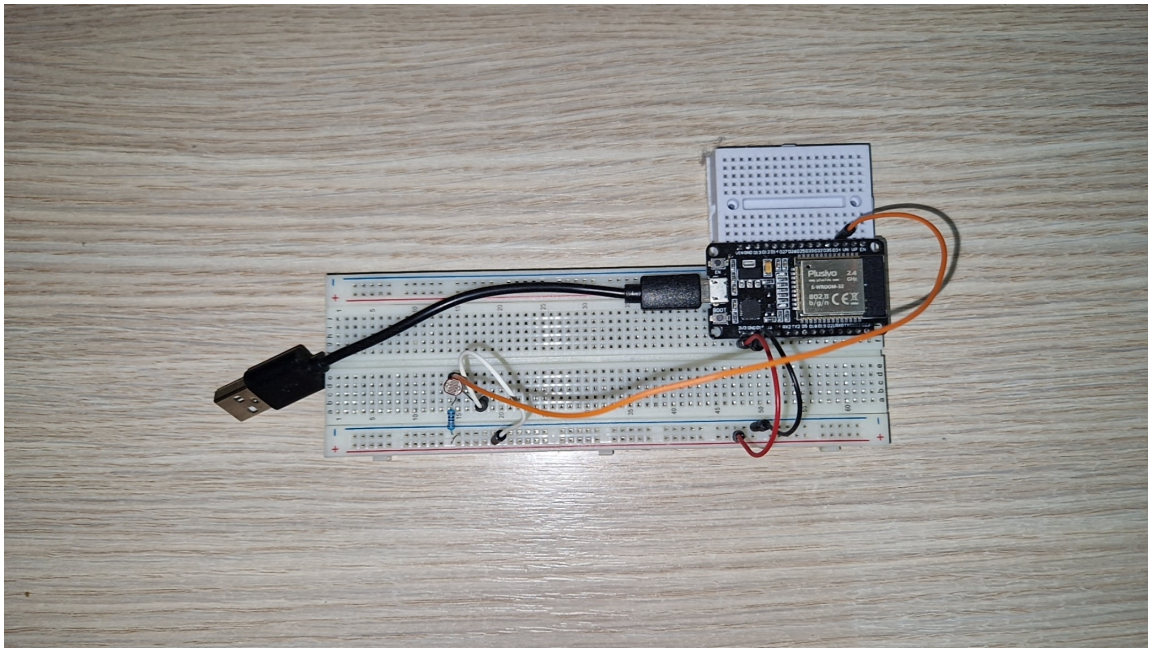

Figura 3: Circuitul fizic rezultat

3.3 Implementarea interfeței expuse utilizatorului

Modalitatea aleasă de realizare a serverului este cea reprezentată de utilizarea framework-ului Flask, întrucât se integrează facil baza de date SQLite3 și funcționalitatea se redactează în mod intuitiv, cu ajutorul rutelor. Fiecare rută definește câte o operație expusă de interfață web construită cu ajutorul template-urilor HTML:

- / : pagina principală a aplicației, care expune butoane pentru alegerea acțiunii de către utilizator;
- /login: introducerea IP-ului și token-ului aferente becului (nu se poate trece la pagina de pornire fără acestea); opțional, pentru a putea emite statisticile calculate pe telefon, se pot introduce token-ul asociat bot-ului custom Telegram configurat în prealabil și ID-ul chat-ului pe care se vor primi mesajele;
- /turn-on: se trimite comanda de pornire a becului;
- /turn-off: se trimite comanda de stingere a becului;
- /set-brightness: se trimite comanda de actualizare a luminozității becului conform valorii introduse de utilizator în pagină;
- /adjust-brightness: se trimite comanda de actualizare a luminozității becului conform ultimei valori citite de la senzor;
- /current-status: se trimite comanda de determinare a stării curente a becului (dacă este pornit/oprit și procentul de luminozitate);
- /visualizeData: se construiește graficul intensității luminoase din încăpere, al statusului becului și a gradului de luminozitate a becului în funcție de timp, care se actualizează în timp real pe măsură ce senzorul citește datele și sunt introduse în baza de date;
- /getStatistics: se determină statistici relevante pe baza datelor aflate la un moment dat în baza de date (media valorilor inten-

sităților luminoase preluate de la senzor și cele unice înregistrate); de asemenea, pentru simplitate, se estimează următoarea valoare a intensității cu ajutorul metodei ARIMA; aceste informații sunt trimise utilizatorului pe telefon prin intermediul aplicației Telegram, pentru a ține evidența gradului de lumină din încăpere mai ușor de la distanță;

- /twinkle: se trimite comanda de creare a unui efect luminos pentru bec: clipire de 3 ori.

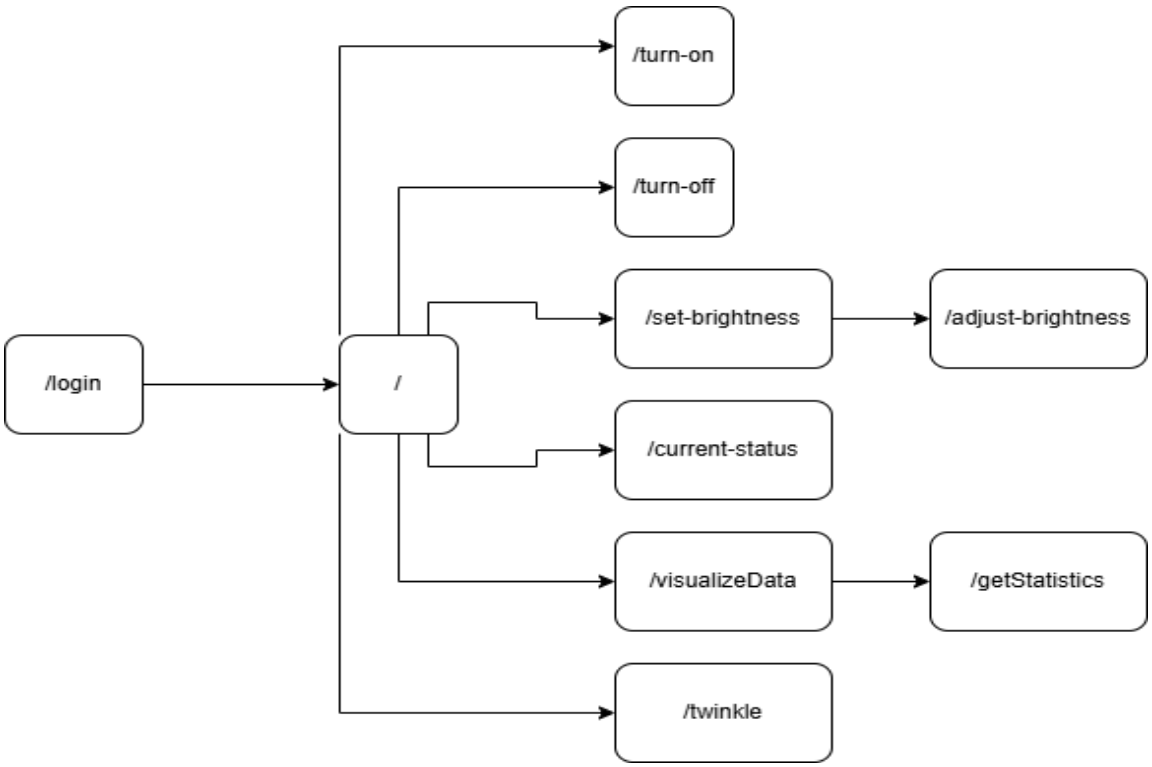

Figura 4: Navigarea prin paginile web ale serverului

Pentru fiecare rută definită endpoint-ul execută operațiile necesare asupra actuatorului și, dacă este cazul, asupra bazei de date (pentru introducere de date și interogare). Baza de date este de tip SQLite3 și are următoarea structură:

- id: identificator unic pentru fiecare înregistrare nouă introdusă;
- record_time: data completă la care a fost citită valoarea de la senzor;
- state: starea curentă a actuatorului (aprins/stins);
- light_val: gradul de luminozitate setat pentru actuator (ultima valoare asignată);
- light_intensity: intensitatea luminoasă citită de la senzor;
- color_temp: temperatura culorii becului (ultima valoare setată).

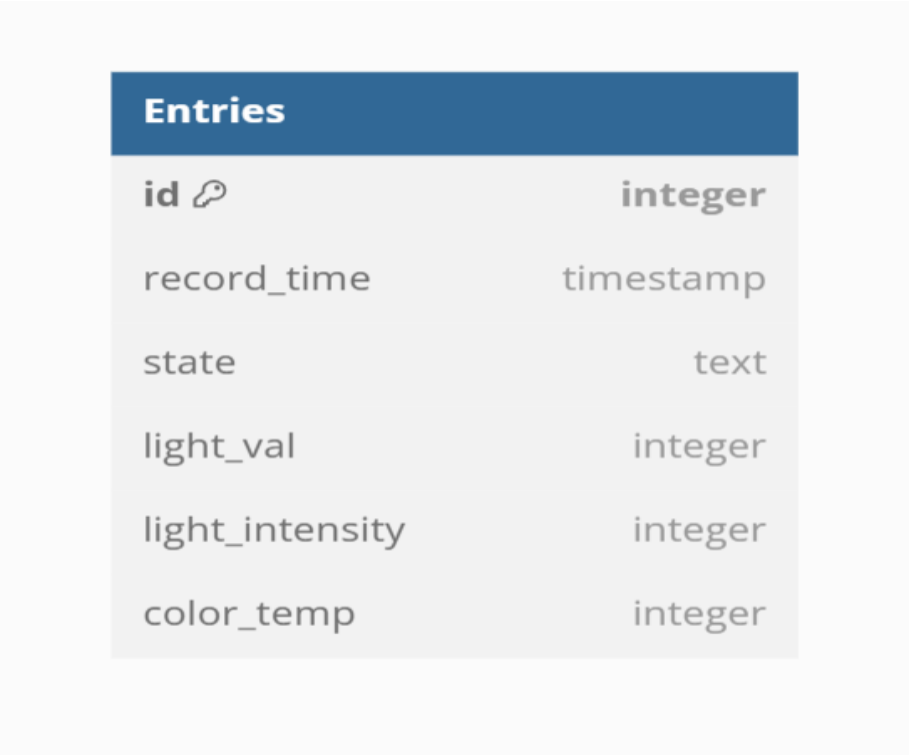

Figura 5: Vizualizarea structurii tabelei

Mai jos sunt prezentate paginile web la care utilizatorul are acces:

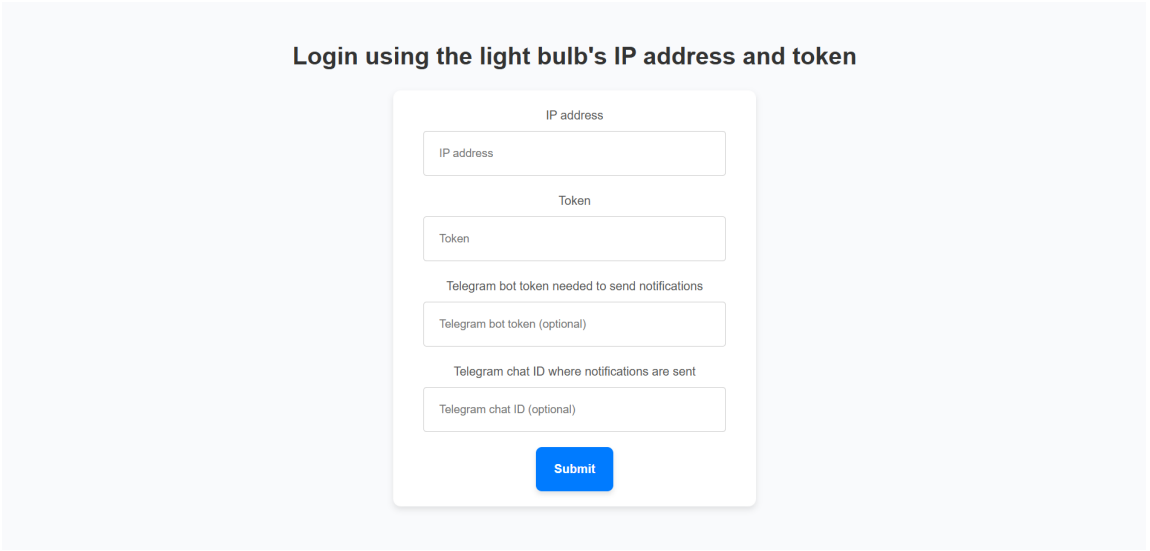

Figura 6: Pagina de login

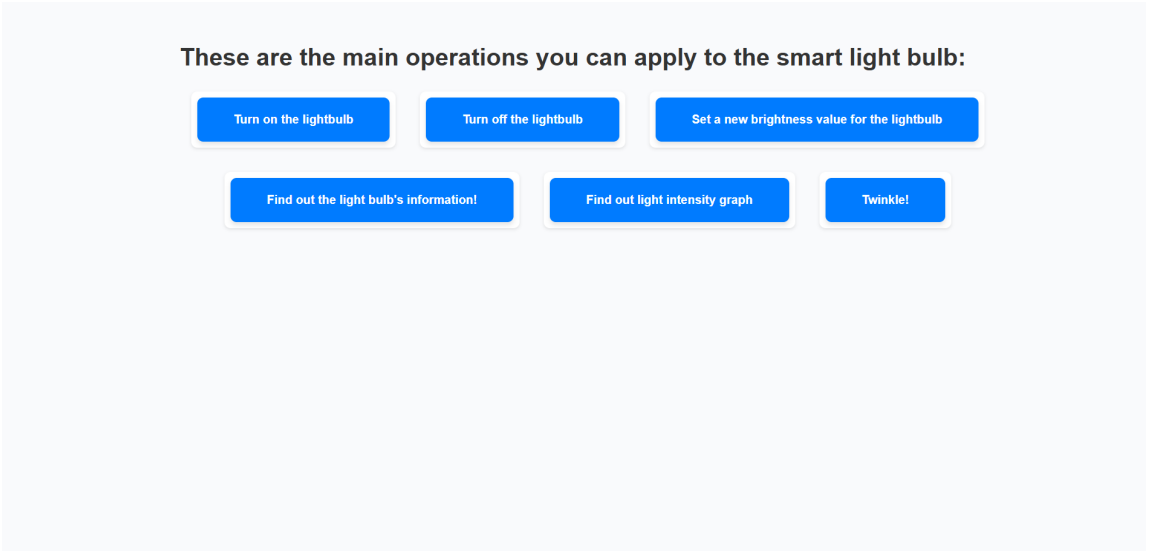

Figura 7: Pagina de pornire (principală)

Figura 8: Pagina cu rezultate /turn-on (cazul de succes)

Figura 9: Pagina cu rezultate /turn-off (cazul de succes)

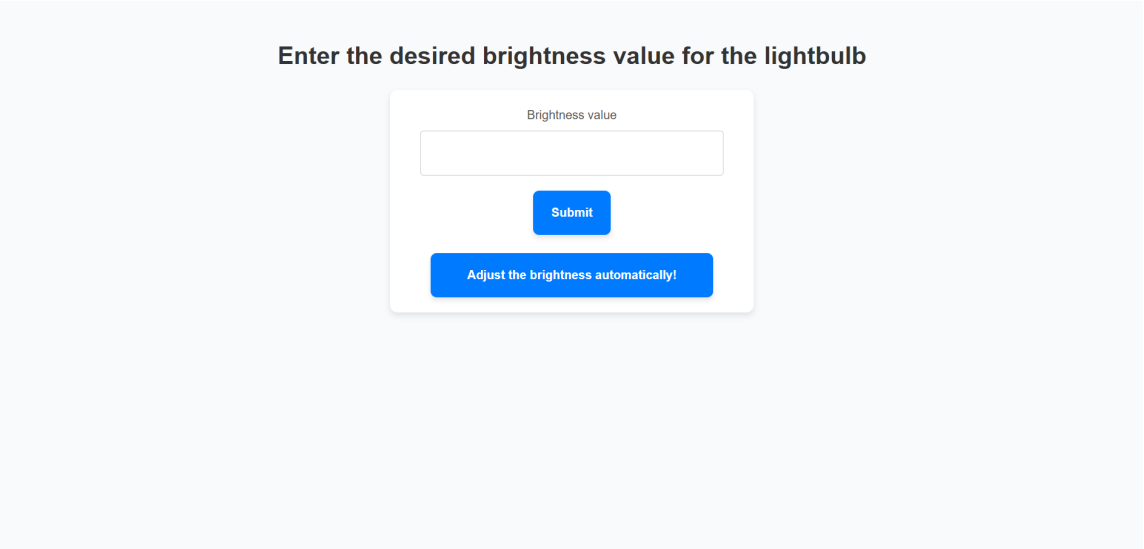

Figura 10: Pagina de setat o nouă valoare a luminozității becului

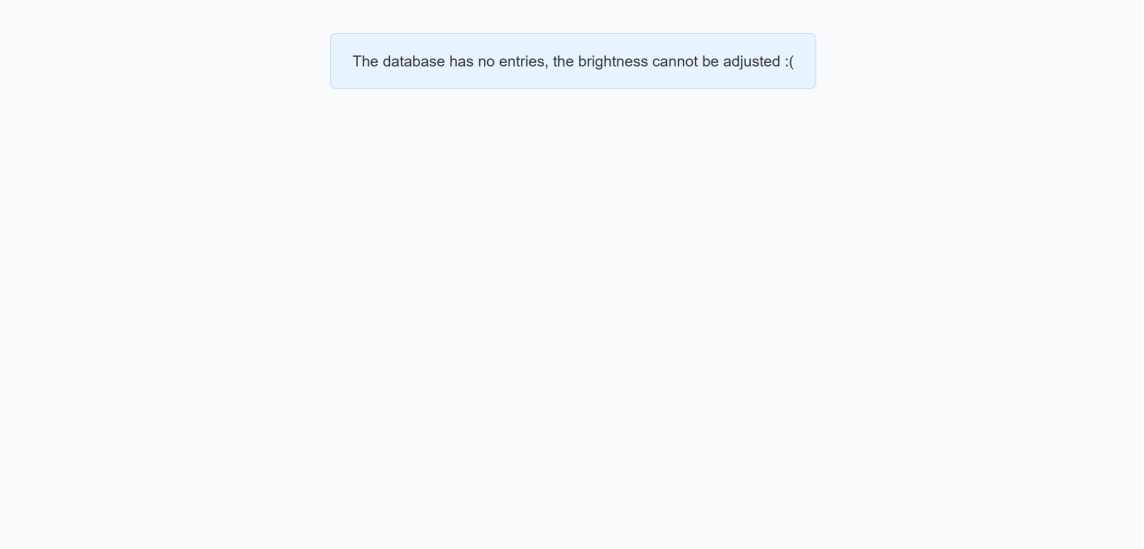

Figura 11: Pagina cu unul dintre cazurile de eșec pentru /adjust-brightness în mod automat (nu există suficiente date introduse în baza de date)

Figura 12: Pagina cu unul dintre cazurile de eșec pentru /set-brightness (becul nu este aprins)

Figura 13: Pagina cu rezultate /set-brightness-result (cazul de succes)

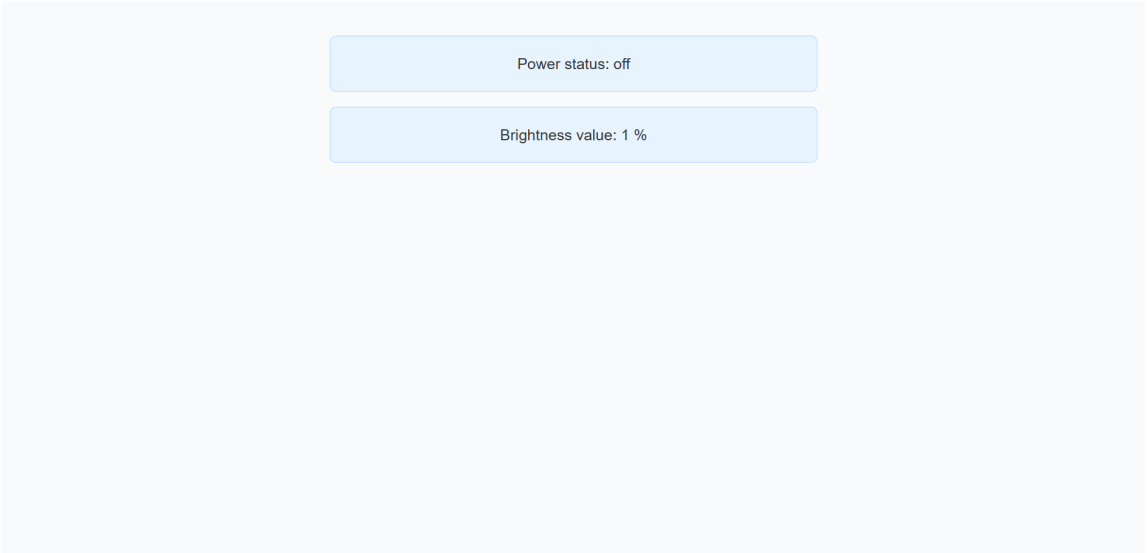

Figura 14: Pagina ce expune informațiile curente despre bec

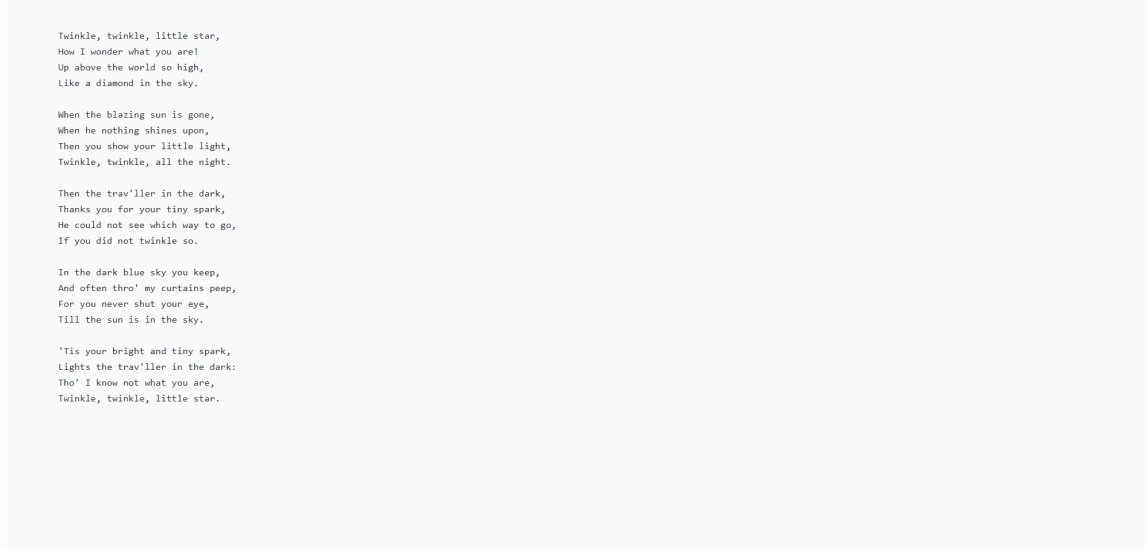

Figura 15: Pagina ce se afișează în urma operației de twinkle efectuate asupra becului

Figura 16: Pagina de eroare standard pentru operațiile care nu se pot efectua (datele despre bec sunt incorecte/becul nu este conectat)

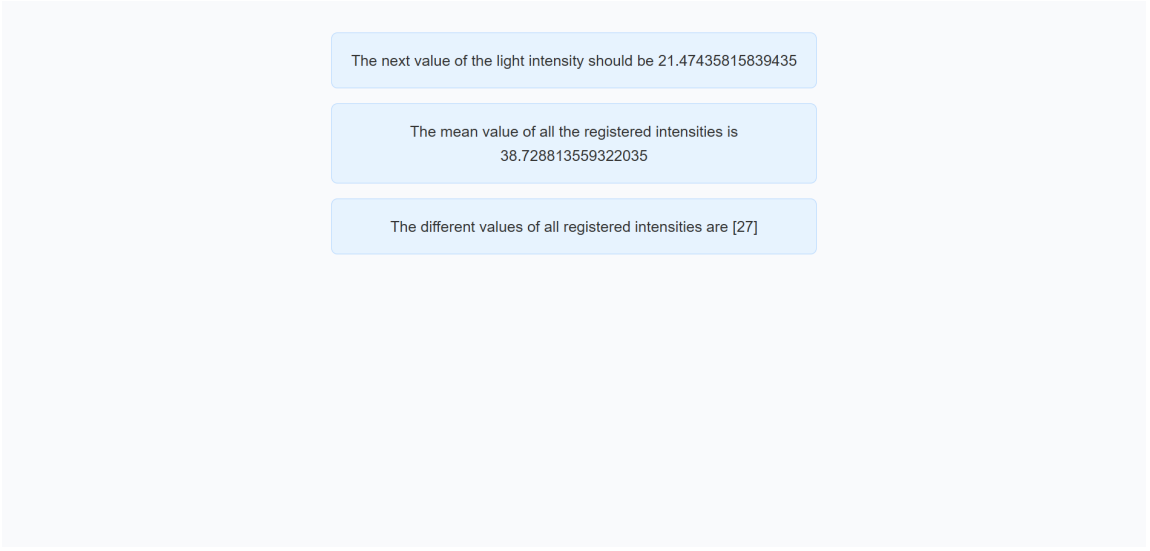

Figura 17: Exemplu de statistici calulate pe baza datelor din grafic (corespundente imaginilor de mai jos)

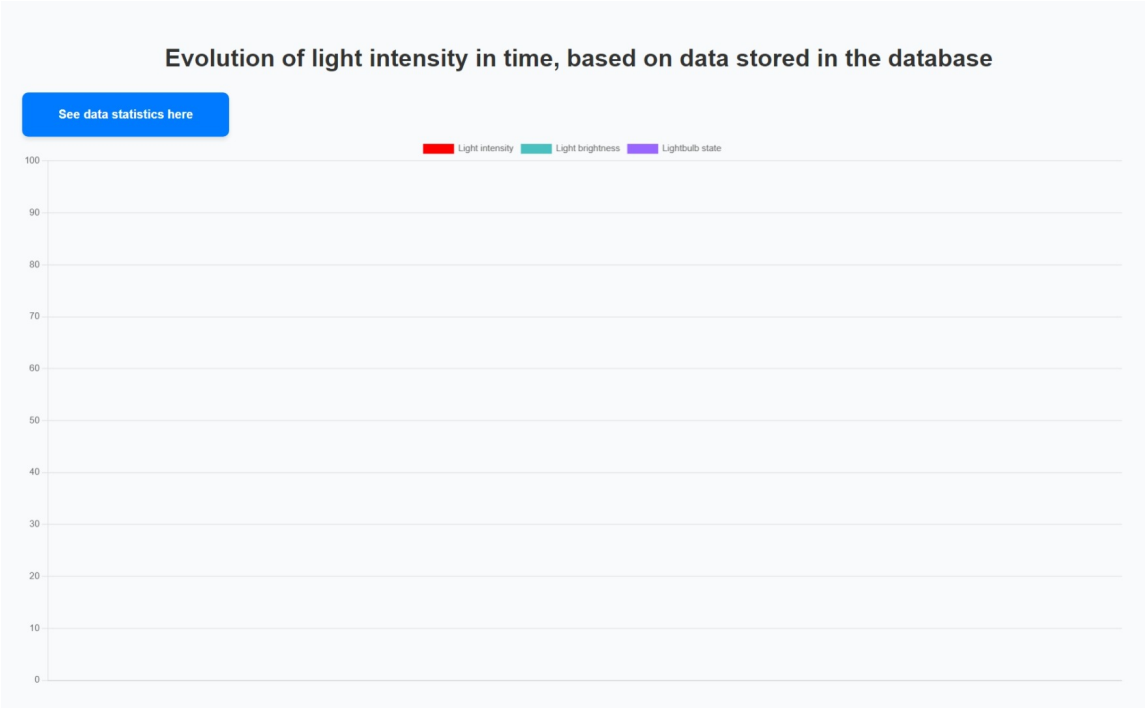

Figura 18: Pagina unde se completează graficul intensităților luminoase înregistrate primite de la senzor

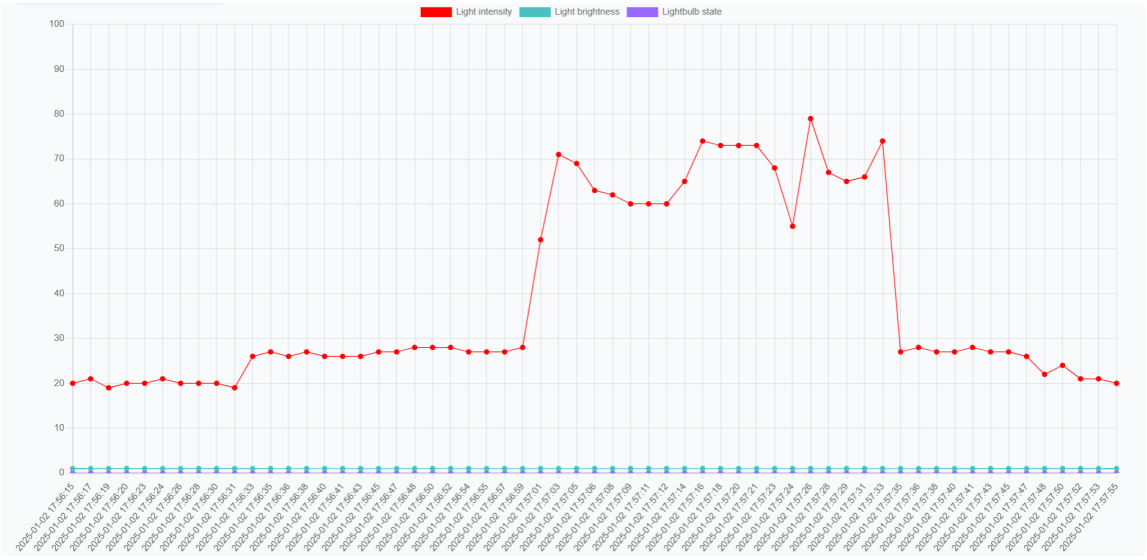

Figura 19: Exemplu de grafic cu date

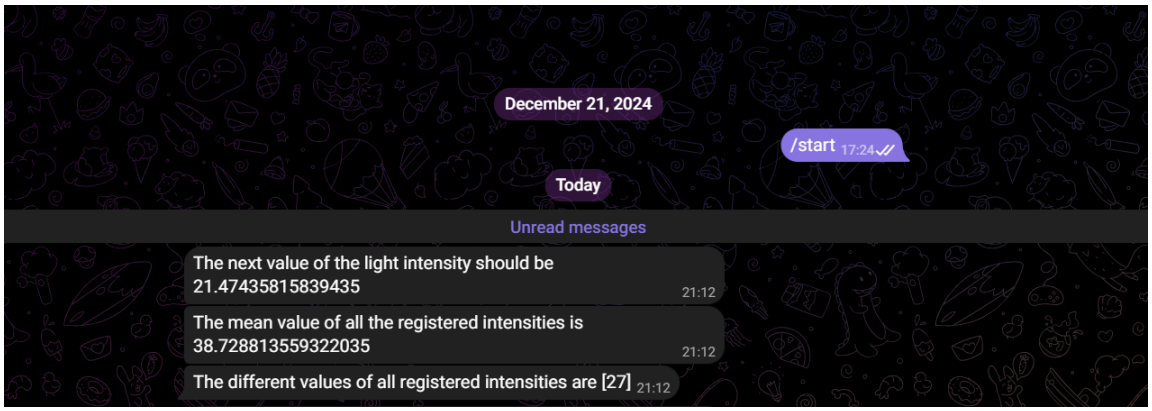

Figura 20: Mesajele transmise pe canalul bot-ului de Telegram ce conțin informațiile afișate la /getStatistics

3.4 Implementarea comenzilor date de către utilizator

Controlul actuatorului se realizează cu ajutorul pachetului python-miio, comenzile principale trimise către acesta fiind următoarele:

```
# starea curenta a becului
res = dev.send("get_prop", ["power", "bright", "ct", "rgb", "led",
    "mode"])
# res este o lista ce contine in ordine valorile cerute in ordine
# daca dispozitivul nu nu este caracterizat de o propritate din
    lista, valoarea din lista de rezultate va fi ''

# aprinderea becului
res = dev.send("set_power", ["on"])
# res reprezinta starea operatiei (succes/la eroare se arunca o
    exceptie)

# stingerea becului
res = dev.send("set_power", ["off"])
# res reprezinta starea operatiei (succes/la eroare se arunca o
    exceptie)

# setarea temperaturii culorii
res = dev.send("set_ct_abx", [2700, "smooth", 500])
```

```
# res reprezinta starea operatiei (succes/la eroare se arunca o
    exceptie)

# modificarea gradului de iluminare a becului (in acest caz 10;
    valoarea este intre 1 si 100)
res = dev.send("set_bright", [10])
# res reprezinta starea operatiei (succes/la eroare se arunca o
    exceptie)
```

Microcontroller-ul trimite cereri de tip POST serverului la ruta /submit ce conțin valorile intensității luminoase; payload-ul este extras și introdus în baza de date alături de data la care au ajuns cererile, starea becului, luminozitatea lui curentă și temperatura culorii. Rezultatul în urma operațiilor de actualizare a valorii luminozității actuatorului este afișat la ruta /set-brightness-result/<bright>/<err>/<res>, în urma redirectării de la /set-brightness care transmite prin query params valoarea nouă a luminozității, dacă operația a fost efectuată cu succes și mesajul care se afișează în pagină.

3.5 Sistem de alertare și notificare

Utilizatorul primește constant feedback în timpul navigării printre paginile web puse la dispoziție de server, prin afișarea de mesaje de informare în caz de succes sau de eroare (pentru a ști cum se pot remedia acestea). În cazul în care acesta dorește să fie notificat cu privire la statisticile determinate la un moment dat se pot introduce la logare datele necesare unui bot de Telegram pentru a primi mesajele și a putea ține ușor evidența valorilor intensității luminoase din încăpere.

4 Vizualizarea și procesarea datelor

Pentru afișarea graficului s-a folosit framework-ul Chart.js, întrucât este ușor de încorporat în template-ul HTML. Valorile introduse în grafic sunt preluate direct din baza de date cu ajutorul rutei /getDBData (ce aplică un query de SELECT asupra tabelei). Graficul

rezultat se actualizează în timp real, o dată la 5 secunde, pentru a putea observa facil modificările gradului de lumină din încăpere și efectul adaptării automate a luminozității actuatorului.

Ruta pentru preluarea datelor este cea de mai jos, care întoarce un obiect JSON cu acestea pentru a fi parsate în secțiunea <script>din pagina HTML:

```
def getDB():
    db = getattr(g, '_database', None)

    if db is None:
        db = g._database = sqlite3.connect(DATABASE)

    return db

@app.route("/getDBData")
def showEntries():
    with app.app_context():
        db = getDB()

        cur = db.execute('SELECT id, record_time, state, light_val,
           light_intensity, color_temp FROM entries')

        entries = cur.fetchall()
        return jsonify(entries)
```

La deschiderea paginii se execută funcția care instanțiază un obiect de tip Chart în cadrul unui element de tip canvas; acestuia i definesc componentele ce vor fi vizualizate (intensitatea luminoasă, valoarea brightness-ului actuatorului și starea acestuia), alături de legenda fiecăruia și caracteristicile desenării efective; de asemenea se setează limitele valorilor de pe axe (în cazul de față intensitatea este între 0 și 100). Datele sunt preluate și mapate fiecărei componente din grafic în funcția updateChart():

```
$(document).ready(function() {
    var ctx = document.getElementById('myChart').getContext('2d');
    var myChart = new Chart(ctx, {
        type: 'line',
        data: {
            labels: [],
            datasets: [
                {
                    label: 'Light intensity',
                    data: [],
                    borderColor: 'rgba(255, 0, 0, 1)',
                    backgroundColor: 'rgba(255, 0, 0, 1)',
                    borderWidth: 1
                },
                {
                    label: 'Light brightness',
                    data: [],
                    borderColor: 'rgba(75, 192, 192, 1)',
                    backgroundColor: 'rgba(75, 192, 192, 1)',
                    borderWidth: 1
                },
                {
                    label: 'Lightbulb state',
                    data: [],
                    borderColor: 'rgba(153, 102, 255, 1)',
                    backgroundColor: 'rgba(153, 102, 255, 1)',
                    borderWidth: 1
                }
            ]
        },
        options: {
            scales: {
                y: {
                    min: 0,
                    max: 100
                }
            }
        }
    });

    function updateChart() {
```

```
        $.getJSON('/getDBData', function(data) {
            var labels = data.map(item => item[1]);
            var lightValues = data.map(item => item[3]);
            var lightIntensities = data.map(item => item[4]);
            var states = data.map(item => item[2] == 'off' ? 0 : 1);

            myChart.data.labels = labels;
            myChart.data.datasets[1].data = lightValues;
            myChart.data.datasets[0].data = lightIntensities;
            myChart.data.datasets[2].data = states;
            myChart.update();
        });
    }

    // Initial chart update
    updateChart();

    // Update the chart every 5 seconds
    setInterval(updateChart, 5000);
});
```

5 Securitate

Transmiterea datelor de la senzor către server se realizează prin protocolul HTTP securizat, paginele web asociate server-ului având certificate generate care atestă siguranța conexiunii. Microcontroller-ul și server-ul comunică prin intermediul socketilor, care dispun de criptare SSL. În cadrul programului uploadat pe plăcuță se deschide câte un socket pentru fiecare pachet de date care se trimite, cu ajutorul funcțiilor auxiliare următoare:

```
# intoarce socket-ul nou deschis pentru o noua conexiune
def connect_to_socket():
    addr = socket.getaddrinfo(SERVER_IP, SERVER_PORT)[0][-1]

    sock = socket.socket()
    sock.connect(addr)
    sock = ssl.wrap_socket(sock)
```

```
    return sock

# SERVER_IP si SERVER_PORT sunt constante definite anterior
def send_to_socket(sock, data):
    json_data = ujson.dumps(data)
    packet = (
        f"POST /submit HTTP/1.1\r\n"
        f"Host: {SERVER_IP}:{SERVER_PORT}\r\n"
        f"Content-Type: application/json\r\n"
        f"Content-Length: {len(json_data)}\r\n"
        f"\r\n"
        f"{json_data}"
    )

    sock.write(packet.encode('utf-8'))
```

Pachetul de date construit trebuie să respecte standardul HTTP pentru ca cererea POST să întoarcă un cod de succes (201), astfel că înaintea payload-ului se introduc headerele specifice necesare, care descriu tipul operației, ruta la care ajung datele, tipul și lungimea lor.

Biblioteca dedicată SSL pentru ultima versiune de Micropython nu permite setarea manuală a contextului de verificare a certificatului necesar criptării datelor serverului, fapt pentru care nu este necesară urcarea lui și a cheii în memoria microcontroller-ului.

6 Provocări întâmpinate în cadrul implementării

Dificultățile majore ale implementării au constat în identificarea unei biblioteci de Python care să poată trimite comenzi către actuatorul de tipul celui utilizat și în asigurarea comunicării securizate dintre server și microcontroller: generarea certificatelor, tranformarea lor în formatul acceptat de către aplicația Flask (.pem), efectuarea cererilor POST de pe socketi în mod corect.