Proiect PR- Sistem smart de iluminare a curții v2

# Introducere

În viața de zi cu zi se întâmplă foarte des să uiți lumina aprinsă undeva. Cel mai enervant este când deja ești pregătit să dormi și iți dai seama că ai uitat lumina aprinsă afară și trebuie să cobori, jumătate adormit, să stingi luminile din toate curtea.

De aceea, dupa multe rugaminte primite de la tata, am decis sa creez un sistem de iluminare inteligent ce se aprinde automat in jurul orei apusului si se dezactiveaza dupa 4 ore, astfel neavand grija ca va uita vreo lumina aprinsa. Mai mult, pentru a salva din curentul folosit, senzorii de miscare conectati la aparat vor detecta cand cineva e in curte si, astfel, va aprinde lumina cand e nevoie. In functie de senzorul ce detecteaza miscarea, se vor activa doar luminile din zona acelui senzor. Ei bine, asta versiunea 1.0! Acest sistem avea totul făcut direct pe ESP32, mai exact calculele prin care își seta orele de funcționare, fiind totodată fixat să funcționeze doar într-o anumită locație (având geolocația setată pe București).

Astfel, scopul proiectului este să îmbunătățesc vechiul sistem de iluminare! Mai exact, să fac ca sistemul să fie configurat printr-un program simplu, în python, ce îi va trimite configurațiile plăcuței, totodată având și un mini sistem de supraveghere ce înregistrează de fiecare data când e detectat cineva de către senzori în afara orelor de funcționare (mai exact, pe timpul nopții, dar lăsând să fie la mâna utilizatorului). Totodată, voi elimina nevoia plăcuței de a știi exact unde se află, lăsând la latitudinea aplicației să calculeze ora de apus și răsărit, trimițându-le la plăcuță.

Senzorii de mișcare vor detecta mișcarea, înregistra timpul la care se întâmplă, iar plăcuța va trimite aceste informații la baza de date.

Desigur, acest proiect reprezintă un prototip, acesta fiind făcut la o scară mult mai mică decât proiectul în sine, având doar o bandă de 30 de led-uri și doi senzori PIR, putând, desigur, să fie extins pentru mult mai mulți senzori și benzi de led (codul putând fi portat pe mai multe plăcuțe ce se vor conecta la același broker de mosquitto).

# Arhitectură

A computer with wires and lights

Description automatically generated with medium confidencePe partea de hardware, piesele folosite vor fi un ESP32, 2 senzori PIR, o Banda Led RGB WS2812B, cabluri și alimentarea pentru ESP32. Mai jos avem piesele propriu zise și schema electrică a proiectului.

A diagram of a circuit board

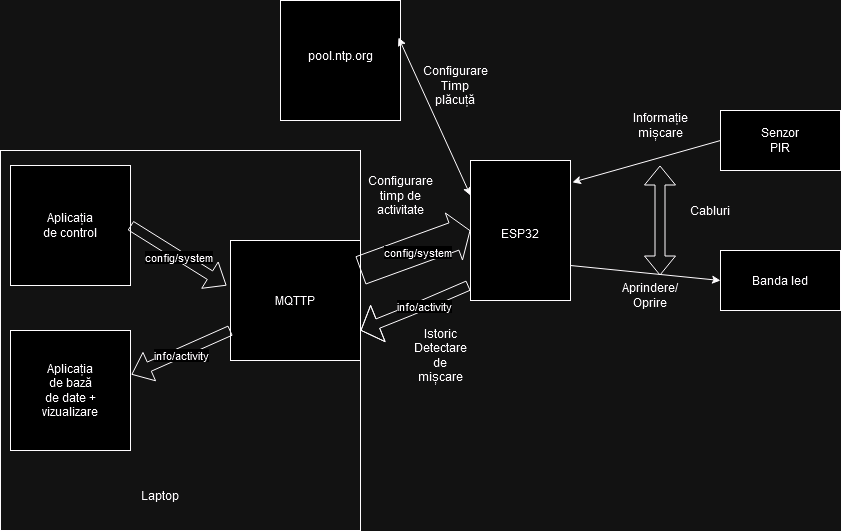
Description automatically generated

Pe partea software, vom avea aplicația de control făcută în Python ce va comunica cu plăcuța și va oferi informațiile despre configurarea sistemului. Mai exact, va permite utilizatorului de a alege între opțiunea default de a folosi orele răsăritului și al apusului pentru orele de activitate (plus un delay de câteva minute dacă se vrea) și opțiunea de a seta orele pe care le vrea utilizatorul pentru lumină. Astfel, prin aplicația de configurare, utilizatorul are libertatea de a folosi luminile la orice oră, atât timp cât a făcut setup-ul de câteva comenzi.

Pe lângă aceasta vom avea o aplicație ce înregistrează datele trimise de plăcuță, mai exact date despre când aceasta detectează mișcare, pe care după utilizatorul le poate accesa.

Posibile adăugări este ca plăcuța să poată trimite notificări pe timpul nopții printr-un bot de Telegram, astfel anunțând direct utilizatorul când s-a întâmplat ceva, numele utilizatorilor fiind configurabil, desigur, prin aplicația de control.

Pentru toate această voi folosi MQTT pentru comunicarea între plăcuță și aplicația de baza de date și cea de control, folosind topic-uri diferite pentru cele două. Partea de MQTT va fi rulată pe laptop, folosindu-ma de hotspot-ul mobil (pentru a asigura aceeași adresa IP tot timpul) și de portul 8883.



# Implementare

Proiectul constă în 3 mici programe, unul de configurare, programul de pe plăcuță și cel de salvare și afișare a datelor. În următorul capitol mă voi axa pe programul de configurare și programul de pe plăcuță, urmând ca în următorul capitol să vorbesc de cel legat de afișarea + salvarea datelor.

## Programul de configurare

Pentru a configura sistemul pentru a fi activ în intervalul orar vrut (fie default între apus și răsărit, fie într-un interval dat de utilizator), se folosește o aplicație în Python denumită **Config\_Program.py**. Aceasta afișează la pornire un meniu cu mai multe opțiuni, așteptând input-ul de la utilizator:

A black screen with white text

Description automatically generated

În același timp, când aceasta rulează, se va conecta prin librăria paho.mqtt la brokerul de mqtt instalat și activ pe laptop (mosquitto) (aici arătat prin „Broker granted the following QoS: 0”).

Dacă configurația selectată este cea apus + răsărit, se va cere un interval de timp de la utilizator. Mai exact, la câte minute după apus să se activeze automat și la câte minute înainte de apus să se dezactiveze sistemul de iluminare.

A screenshot of a computer screen

Description automatically generated

O dată primite datele, acesta va cere datele despre locație (latitudine, longitudine și fusul orar de tip continent + capitală) iar apoi va trimite o cerere https la api-ul prezent la link-ul [**https://api.sunrise-sunset.org/json**](https://api.sunrise-sunset.org/json)+ parametrii pentru latitudine, longitudine și fus orar.

def fetch\_data(latitude, longitude, tzid):

    url = "https://api.sunrise-sunset.org/json"

    params = {

        "lat": latitude,

        "lng": longitude,

        "formatted": 0,

        "date": "today",

        "tzid": tzid

    }

După ce are datele necesare, acesta le va parsa și împacheta într-un json, adăugând diferența de minute acolo unde este cazul. Apoi le trimite pe topicul config/system.

try:

        # Efectuarea cererii HTTP GET

        response = requests.get(url, params=params)

        # Verificarea statusului răspunsului

        if response.status\_code == 200:

            # Parsarea răspunsului JSON

            data = response.json()

            # Extrage datele utile

            sunrise = data['results']['sunrise']

            sunset = data['results']['sunset']

            # Afișează informațiile

            sunrise\_time = datetime.datetime.fromisoformat(sunrise)

            sunset\_time = datetime.datetime.fromisoformat(sunset)

            return sunrise\_time, sunset\_time

        else:

            print("Eroare la obținerea datelor")

    except Exception as e:

        print(f"An error occurred: {e}")

Obținerea datelor din răspunsul primit de la cererea HTTPS.

def sunset\_sunrise\_calc(delta\_sunset, delta\_sunrise):

    latitude = input("Enter the latitude: ")

    longitude = input("Enter the longitude: ")

    if latitude == "" or longitude == "":

        print("Invalid input")

        return

    tzid = input("Enter timezone format:(e.g. Europe/Bucharest): ")

    print("Sunset/Sunrise calculation")

    sunrise\_hour, sunset\_hour = fetch\_data(latitude, longitude, tzid)

    sunrise\_hour = sunrise\_hour + datetime.timedelta(minutes=int(delta\_sunrise))

    sunset\_hour = sunset\_hour + datetime.timedelta(minutes=int(delta\_sunset))

    print(f"Deactivation hour: {sunrise\_hour}")

    print(f"Activation hour: {sunset\_hour}")

    json\_data = {

        "Activation": sunset\_hour.strftime("%Y-%m-%dT%H:%M:%S"),

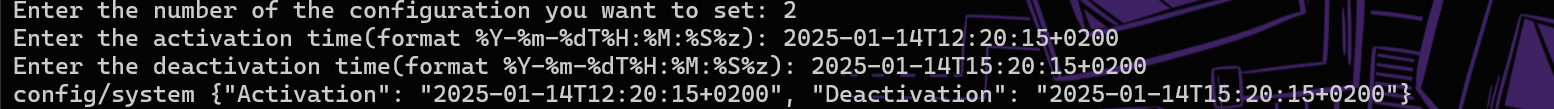
        "Deactivation": sunrise\_hour.strftime("%Y-%m-%dT%H:%M:%S")

    }

    return json\_data

Calcularea datelor ce vor fi trimise prin mqtt în format json

Dacă e selectat un interval orar, se introduc orele de activitate, ce sunt după împachetate într-un json și trimise mai departe pe topicul config/system.

Cum arată în meniul aplicației selectarea opțiunii de introducere a unui interval orar ales de utilizartor

elif message == "2":

        activation\_time = input("Enter the activation time(format %Y-%m-%dT%H:%M:%S%z): ")

        deactivation\_time = input("Enter the deactivation time(format %Y-%m-%dT%H:%M:%S%z): ")

        config\_message = {

            "Activation": activation\_time,

            "Deactivation": deactivation\_time

        }

        msg\_info = mqttc.publish("config/system", json.dumps(config\_message), qos=1)

        msg\_info.wait\_for\_publish()

        print("/n")

Partea de cod din spatele acestei acțiuni.

## Programul de pe plăcuță

Codul este împărțit în mai multe bucăți: partea ce se ocupă cu elementele hardware (întreruperi cauzate de senzori și activarea benzii de led) și partea ce se ocupă de comunicarea cu celelalte două programe (partea de conectare la mosquitto). Codul pentru aceasta este scris în C++, mai exact în ce se foloseste pentru Arduino.

### Senzorii PIR

Pentru a configura senzorii PIR, m-am folosit de întreruperi. Astfel, de fiecare dată când aceștia detectează o mișcare, vor trimite un mesaj pe Serial în funcție de dacă e în program de funcționare sau nu. Partea această e mai mult pentru debug. Dar, pe lângă aceasta, o dată ce detectează o mișcare, vor trece o variabilă pe 1 pentru a ști să anunțe plăcuța să trimită mesaj la baza de date că a detectat activitate.

    pinMode(PIR\_SENSOR\_OUTPUT\_PIN\_1, INPUT);

    attachInterrupt(digitalPinToInterrupt(PIR\_SENSOR\_OUTPUT\_PIN\_1), pir\_1, FALLING);

    pinMode(PIR\_SENSOR\_OUTPUT\_PIN\_2, INPUT);

    attachInterrupt(digitalPinToInterrupt(PIR\_SENSOR\_OUTPUT\_PIN\_2), pir\_2, FALLING);

Configurarea pinurilor.

void IRAM\_ATTR pir\_1() {

  if (connected == 1) {

     motion\_1 = 1;

     Serial.println("Object Detected\_1 things connected");

  } else {

    Serial.println("Object Detected\_1 nothing connected");

  }

  senzor\_1\_message = 1;

}

Exemplu de funcție pentru întreruperi.

Notă: **„Connected”** se referă la faptul dacă plăcuța e în intervalul de activitate sau nu (da = 1, nu = 0).

### Banda de led

Pentru banda de led m-am folosit de librăria „**Adafruit\_NeoPixel.h”**. Configurarea acesteia se face în câteva linii iar, pentru a activa banda când e nevoie, trebuie să trec prin fiecare led și să-l activez / dezactivez.

    strip.begin();     // Initialize the LED strip

    strip.show();

La fiecare loop al aplicației, verific dacă mă aflu în intervalul de activitate (la aprox. fiecare 10 minute), iar apoi verific dacă:  
- s-a detectat mișcare (variabila motion\_x este setată pe 1)  
- plăcuța e în domeniul de activitate (connected = 1)  
- dacă banda de led este deja activă (startTimer\_x este fals)

if (connected == 1 && motion\_1 == 1 && startTimer\_1 == false) {

          for (int i = 0; i < LED\_COUNT/2; i++) {

              strip.setPixelColor(i, strip.Color(0, 0, 255));

          }

          strip.show();

          Serial.println("Object Detected\_1");

          startTimer\_1 = true;

          lastTrigger\_1 = millis();

      }

Dacă toate acestea sunt adevărate, activez banda de led pentru un minut. După ce activez, verific dacă a trecut acel minut pentru a știi dacă sting lumina sau nu (îmi salvez într-o variabilă, lastTrigger\_x, când mai exact i-am dat drumul).

if( startTimer\_1 && (now - lastTrigger\_1 > ( timeMinutes\*1)))

      {

        Serial.println("Turning OFF the LED 1" );

        for (int i = 0; i < LED\_COUNT/2; i++) {

          strip.setPixelColor(i, strip.Color(0, 0, 0));

        }

        strip.show();

        startTimer\_1 = false;

        motion\_1 = 0;

      }

### Comunicarea prin MQTT

Comunicarea cu broker-ul de MQTT se face prin folosirea librăriei **PubSubClient.h.** Pentru aceasta am salvat în variabile adresa folosită de broker-ul de mosquitto, topic-urile pe care le folosește plăcuța și certificatul CA folosit pentru conectarea cu TLS la MQTT și contu pe care se loghează placuța în broker. Pentru partea de securitate (TLS), o să mă leg la capitolul aferent. Partea de cod legată de conectare le-am luat dintr-un exemplu pentru interacțiunea ESP32 – MQTT.

Astfel, pentru toate acestea mi-am făcut două funcții, una prin care verific dacă plăcuța e conectată la broker (și dacă nu, încerc reconectarea) și una ce e apelată de fiecare dată când este primit un mesaj pe topic-ul „config/system”.

void connectToMQTT() {

    while (!mqtt\_client.connected()) {

        String client\_id = "esp32-client-" + String(WiFi.macAddress());

        Serial.printf("Connecting to MQTT Broker as %s...\n", client\_id.c\_str());

        if (mqtt\_client.connect(client\_id.c\_str(), mqtt\_username, mqtt\_password)) {

            Serial.println("Connected to MQTT broker");

            mqtt\_client.subscribe(mqtt\_topic);

        } else {

            Serial.print("Failed to connect to MQTT broker, rc=");

            Serial.print(mqtt\_client.state());

            Serial.println(" Retrying in 5 seconds.");

            delay(5000);

        }

    }

}

Funcția de conectare

Pentru funcția apelată la fiecare mesaj primit, am adăugat și partea de configurare a orei de activare și dezactivare, întrucât plăcuța primește mesaje doar dacă se schimbă configurarea. Partea de text trimis în serial e mai mult pentru debug și să pot confirma 100% că totul se trimite și configurează cum trebuie.

void mqttCallback(char \*topic, byte \*payload, unsigned int length) {

    Serial.print("Message received on topic: ");

    Serial.println(topic);

    Serial.print("Message: ");

    char messageBuffer[300];

    memcpy(messageBuffer, payload, length);   //copy the payload into the array

    messageBuffer[length] = '\0';

    Serial.printf("%s\n", messageBuffer);

    StaticJsonDocument<300> doc;

    DeserializationError error = deserializeJson(doc, messageBuffer);

    const char\* activation = doc["Activation"];

    const char\* deactivation = doc["Deactivation"];

    Serial.printf("Deactivation: %s\n", activation);

    Serial.printf("Activation: %s\n", deactivation);

    parse\_time(activation, deactivation);

    Serial.print("Ora activare: "); Serial.println(start\_time.tm\_hour);

    Serial.print("Minute activare: "); Serial.println(start\_time.tm\_min);

    Serial.println("\n-----------------------");

    config = 1;

}

Funcția de callback

void parse\_time(const char \*activation, const char \*deactivation) {

    sscanf(activation, "%4d-%2d-%2dT%2d:%2d:%2d",

         &start\_time.tm\_year,

         &start\_time.tm\_mon,

         &start\_time.tm\_mday,

         &start\_time.tm\_hour,

         &start\_time.tm\_min,

         &start\_time.tm\_sec);

    start\_time.tm\_year -= 1900;

    start\_time.tm\_mon -= 1;

    start\_time.tm\_isdst = -1;

    sscanf(deactivation, "%4d-%2d-%2dT%2d:%2d:%2d",

         &end\_time.tm\_year,

         &end\_time.tm\_mon,

         &end\_time.tm\_mday,

         &end\_time.tm\_hour,

         &end\_time.tm\_min,

         &end\_time.tm\_sec);

    end\_time.tm\_year -= 1900;

    end\_time.tm\_mon -= 1;

    end\_time.tm\_isdst = -1;

}

Funcția parse\_time

Parse\_time este funcția prin care se ia string-ul din json-ul primit în mesaj și se pune în structura tm aferentă.

Odată ce un senzor detectează o mișcare, acesta va pune variabila **senzor\_x\_message** aferentă senzorului pe 1. Astfel, plăcuța va știi că trebuie să creeze un mesaj pe care să-l trimită pe topicul „info/activity”. Funcția constă în luarea timpului curent și formatarea acestuia pentru a putea fi citit după de aplicația ce ține de baza de date. Mai jos este atașată funcția folosită de ambii senzori (diferența se face prin variabila „senzor”):

void send\_message(int senzor) {

    struct tm timeinfo;

    if(!getLocalTime(&timeinfo)){

      Serial.println("Failed to obtain time");

      return;

    }

    char formattedTime[20];

    strftime(formattedTime, sizeof(formattedTime), "%Y-%m-%dT%H:%M:%S", &timeinfo);

    Serial.print("Timpul formatat: ");

    Serial.println(formattedTime);

    char message[100];

    if (senzor == 1) {

      const char\* senzor\_name = "\"Curte\"";

      sprintf(message, "{\"Time\": \"%s\", \"Senzor\": %s}", formattedTime, senzor\_name);

    }

    if (senzor == 2) {

      const char\* senzor\_name = "\"Poarta\"";

      sprintf(message, "{\"Time\": \"%s\", \"Senzor\": %s}", formattedTime, senzor\_name);

    }

    Serial.println(message);

    mqtt\_client.publish(mqtt\_send\_topic, message);

}

# Vizualizare și Procesare de date

Pentru procesoarea și afișarea datelor, am făcut un mic script în Python ce salvează intrările într-un fișier .csv, numit „Database.py”.  
În spate, programul își dă subscribe pe topicul info/activity și, de fiecare dată când primește mesaj, acesta îl va transforma json-ul într-un dicționar și îl va adăuga în dataframe-ul folosit pentru stocarea datelor:

def on\_message(client, userdata, msg):

    payload = msg.payload.decode('utf-8')

    json\_payload = json.loads(payload)

    if "Time" not in json\_payload or "Senzor" not in json\_payload:

        print("Invalid message")

        return

    time\_str = json\_payload["Time"]

    sensor = json\_payload["Senzor"]

    time\_obj = datetime.strptime(time\_str, "%Y-%m-%dT%H:%M:%S")

    day = time\_obj.strftime("%Y-%m-%d")

    time = time\_obj.strftime("%H:%M:%S")

    dictionary = {

        "Day": day,

        "Time": time,

        "Sensor": sensor

    }

    df\_dict = pd.DataFrame([dictionary])

    global database

    database = pd.concat([database, df\_dict], ignore\_index=True)

Programul întâmpină utilizator cu un mic meniu, oferind mai multe opțiuni pentru afișarea datelor:  
A screen shot of a computer

Description automatically generated

(Mesajul de „Broker granted the following QoS: 0” arată conectarea la MQTT când programul este proaspăt deschis).

În spate, opțiunile constau în interogări făcute pe baza de date, prin ajutorul librăriei pandas. La fiecare se va afișa un mic tabel, ce conține trei tabele: ora și data la care a fost activat senzorul și ce sensor a fost activat. Utilizatorul poate alege să vadă datele pentru un anumit senzor sau pentru o anumită zi (sau ambele filtre în același timp).

Mai jos sunt exemple de cum se afișează datele în funcțiile de opțiuniile alese:

A black screen with white text

Description automatically generatedA screenshot of a computer

Description automatically generatedOpțiunea simplă de „Show”.

A screen shot of a computer screen

Description automatically generatedOpțiunea de „Show data from specific day”.

Opțiunea de „Show data from specific sensor”

A black screen with white text

Description automatically generated

Opțiunea de „Show data from specific sensor from a specific day”

Pe partea de cod, totul constă în alegerea coloanei (fie de zi, fie de senzor sau ambele) ce este egală cu ce a introdus utilizatorul de la tastatură:

print(database[(database["Sensor"] == sensor) & (database["Day"] == day)])

Exemplu de astfel de alegere

La final, înainte să se închidă aplicația, datele sunt salvate într-un fișier tip.csv, astfel datele fiind păstrate și în cazul în care device-ul pe care se rulează aplicația se închide, mai jos fiind o poză cu cum arată fișierul .csv:

Day,Time,Sensor

2025-01-13,09:16:15,Poarta

2025-01-13,09:30:15,Poarta

2025-01-13,09:30:15,Curte

2025-01-14,13:09:44,Curte

2025-01-14,13:10:11,Curte

2025-01-14,13:10:11,Poarta

2025-01-14,13:10:20,Curte

2025-01-14,13:10:20,Poarta

2025-01-14,13:10:23,Poarta

2025-01-14,16:39:57,Poarta

2025-01-14,16:39:58,Curte

2025-01-14,16:40:38,Curte

2025-01-14,16:58:19,Curte

2025-01-14,16:59:13,Curte

2025-01-14,16:59:18,Poarta

# Securitate

A black background with white numbers

Description automatically generatedPentru securitate am folosit tls peste mqtt. Mai exact, m-am folosit de certificate ca, generate prin openssl și semnate. Broker-ul are și el la randul lui fișiere tip .key și .crt pe lângă certificat.

Certificatele folosite de broker.

Am verificat să mă asigur ca broker-ul foloseste certificatele, folosind comenzi openssl pentru interogarea acestuia. A computer screen shot of a computer code

Description automatically generated

Certificatele acestea sunt salvate pe laptopul pe care rulează mosquitto. Pe partea de cod, aplicațiile iau certificatul ca (cele de python) sau îl au deja salvat în codul propriu-zis (esp32) și îl folosesc pentru logarea pe mosquitto. Mai mult, fiecare are deja setat username-ul și parola pe care le vor folosi la conectare.

mqttc = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2, "Config")

mqttc.tls\_set(ca\_certs="ca.crt", tls\_version=ssl.PROTOCOL\_TLSv1\_2, ciphers=None)

mqttc.username\_pw\_set("proiect", "1234")

Conectarea în python.

const char \*mqtt\_broker = "192.168.43.212";

const char \*mqtt\_topic = "config/system";

const char \*mqtt\_send\_topic = "info/activity";

const char \*mqtt\_username = "proiect";

const char \*mqtt\_password = "1234";

const int mqtt\_port = 8883;

const char \*ca\_cert = R"EOF(

-----BEGIN CERTIFICATE-----

...........................

-----END CERTIFICATE-----

)EOF";

    // Set Root CA certificate

    esp\_client.setCACert(ca\_cert);

Conectarea pe ESP32

Astfel, nu se pot conecta programe și device-uri ce nu dețin certificatul sau contul de utilizator. E un simplu mod de a menține securitatea device-urilor, dar s-a dovedit că funcționează foarte bine în urma orelor pierdute în timpul testărilor.