

Proiect PR- Sistem smart de iluminare a curții v2

~Drucea Diana Ioana 345C1~

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 îți dai seama că ai uitat lumina aprinsă afară și trebuie să cobori, jumătate adormit, să stingi luminile din toate curtea.

De aceea, după multe rugăminte primite de la tata, am decis să creez un sistem de iluminare inteligent ce se aprinde automat în jurul orei apusului și se dezactivează după 4 ore, astfel neavând grija ca să uita vreo lumina aprinsă. Mai mult, pentru a salva din curentul folosit, senzorii de mișcare conectați la aparat vor detecta când cineva e în curte și, astfel, va aprinde lumina când e nevoie. În funcție de senzorul ce detectează mișcarea, 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).

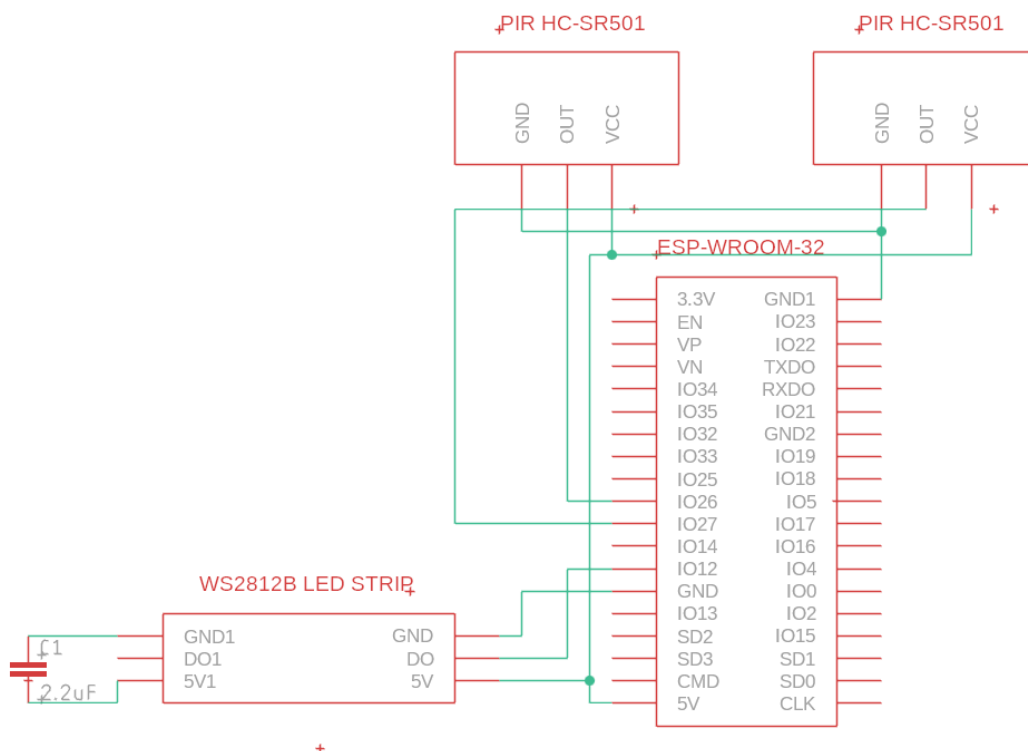
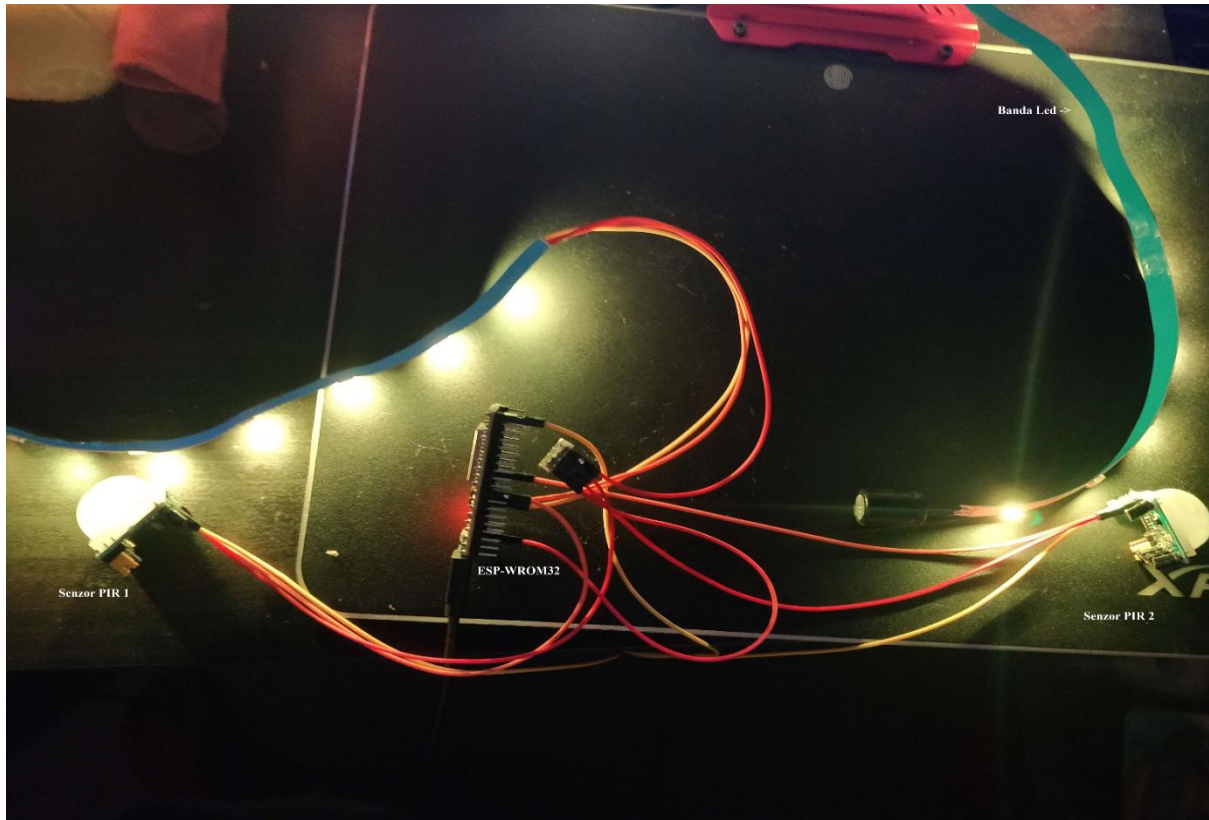
Astfel, scopul proiectului este să îmbunătățesc vechiul sistem de iluminare! Mai exact, să fac ca sistemul să fie configurat printr-un program simplu, în python, ce îi va trimite configurațiile plăcuței, totodată având și un mini sistem de supraveghere ce înregistrează de fiecare dată când e detectat 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 ști 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 mosquito).

Arhitectură

Pe 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.

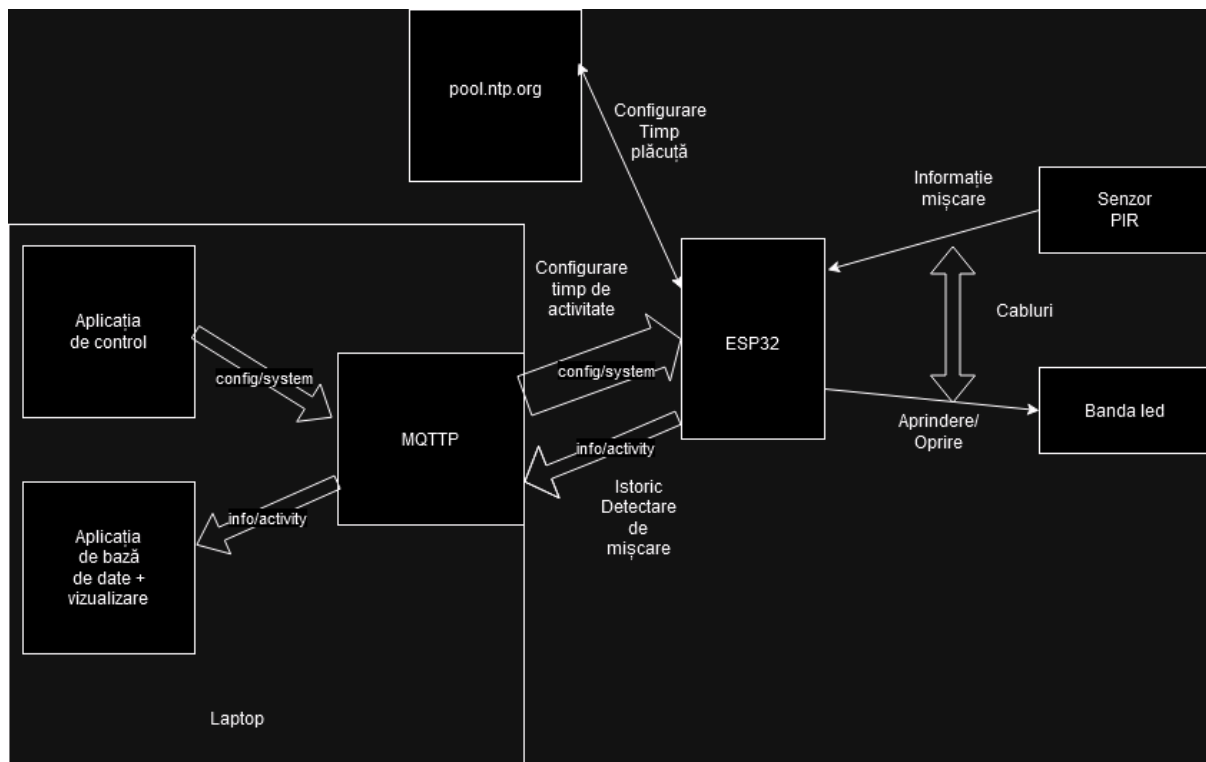


Pe partea software, vom avea aplicația de control făcută în Python ce va comunica cu plăcuța și va oferi informațiile despre configurarea sistemului. Mai exact, va permite utilizatorului de a alege între opțiunea default de a folosi orele răsăritului și al apusului pentru orele de activitate (plus un 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 adresă 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:

```
PS E:\facultate\anu 4\PR\proiect\ProiectPR\src> python Config_Program.py
Config types:
1. Sunset/Sunrise
2. Specific time
3. Exit
Enter the number of the configuration you want to set: Broker granted the following QoS: 0
```

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.

```
Enter the number of the configuration you want to set: Broker granted the following QoS: 0
1
How many minutes after sunset should the lights turn on? 10
How many minutes before sunrise should the lights turn off? 10
Enter the latitude: 44.22
Enter the longitude: 22.10
Enter timezone format:(e.g. Europe/Bucharest): Europe/Bucharest
Sunset/Sunrise calculation
Deactivation hour: 2025-01-14 08:12:23+02:00
Activation hour: 2025-01-14 17:29:02+02:00
```

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> + 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 parse 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.

```
Enter the number of the configuration you want to set: 2
Enter the activation time(format %Y-%m-%dT%H:%M:%S%z): 2025-01-14T12:20:15+0200
Enter the deactivation time(format %Y-%m-%dT%H:%M:%S%z): 2025-01-14T15:20:15+0200
config/system {"Activation": "2025-01-14T12:20:15+0200", "Deactivation": "2025-01-14T15:20:15+0200"}
```

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

```
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 mosquito). Codul pentru aceasta este scris în C++, mai exact în ce se folosește 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 ști 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 plăcuța 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.

Mai întâi de toate, a trebuit să mă asigur că plăcuța se conectează la internet, folosind librăria <Wifi.h>. Mai jos e codul folosit pentru conectarea la o rețea wi-fi, în codul plăcuței fiind hardcodată cu ip-ul pe care îl am la hotspot (ca să îmi fie mult mai ușor la testare.)

```
void connectToWiFi() {
    WiFi.begin(ssid, password);
    Serial.print("Connecting to WiFi");
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("\nConnected to WiFi");
}
```

Astfel, pentru partea de MQTT, 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. Variabila „config” mă asigură că plăcuța are deja o configurație activă pe ea. Este mai mult ca să nu încerce să verifice variabile ce nu au fost declarate deja (de exemplu, start_time și end_time).


```

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 ști 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 procesarea 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 "Sensor" not in json_payload:
        print("Invalid message")
        return

    time_str = json_payload["Time"]
    sensor = json_payload["Sensor"]

    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:

```
Available commands:
1. Show
2. Show data from a specific day
3. Show data from a specific sensor
4. Show data from a specific sensor from a specific day
5. Exit
Enter the number of a command: Broker granted the following QoS: 0
```

(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 senzor 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 opțiunile alese:

Enter the number of a command: Broker granted the following QoS: 0
1

	Day	Time	Sensor
0	2025-01-13	09:16:15	Poarta
1	2025-01-13	09:30:15	Poarta
2	2025-01-13	09:30:15	Curte
3	2025-01-14	13:09:44	Curte
4	2025-01-14	13:10:11	Curte
5	2025-01-14	13:10:11	Poarta
6	2025-01-14	13:10:20	Curte
7	2025-01-14	13:10:20	Poarta
8	2025-01-14	13:10:23	Poarta

Opțiunea simplă de „Show”.

Enter the number of a command: 2
Enter the day(YYYY-MM-DD): 2025-01-13

	Day	Time	Sensor
0	2025-01-13	09:16:15	Poarta
1	2025-01-13	09:30:15	Poarta
2	2025-01-13	09:30:15	Curte

Opțiunea de „Show data from specific day”.

Enter the number of a command: 3
Enter the sensor: Poarta

	Day	Time	Sensor
0	2025-01-13	09:16:15	Poarta
1	2025-01-13	09:30:15	Poarta
5	2025-01-14	13:10:11	Poarta
7	2025-01-14	13:10:20	Poarta
8	2025-01-14	13:10:23	Poarta

Opțiunea de „Show data from specific sensor”

```
Enter the number of a command: 4
Enter the sensor: Poarta
Enter the day(YYYY-MM-DD): 2025-01-14
```

	Day	Time	Sensor
5	2025-01-14	13:10:11	Poarta
7	2025-01-14	13:10:20	Poarta
8	2025-01-14	13:10:23	Poarta

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

Astfel, toată partea de meniu arată așa în cod:

```
while True:
    print("Available commands:")
    print("1. Show")
    print("2. Show data from a specific day")
    print("3. Show data from a specific sensor")
    print("4. Show data from a specific sensor from a specific day")
    print("5. Exit")
    command = input("Enter the number of a command: ")
    if command == "5":
        database.to_csv("database.csv", index=False)
        print("Database was saved in database.csv")
        break
    elif command == "1":
        print('\n')
        print(database)
        print("\n")
    elif command == "2":
        day = input("Enter the day(YYYY-MM-DD): ")
        print('\n')
        print(database[database["Day"] == day])
        print("\n")
    elif command == "3":
        sensor = input("Enter the sensor: ")
        print('\n')
```

```

        print(database[database["Sensor"] == sensor])
        print("\n")
    elif command == "4":
        sensor = input("Enter the sensor: ")
        day = input("Enter the day(YYYY-MM-DD): ")
        print('\n')
        print(database[(database["Sensor"] == sensor) & (database["Day"] ==
day)]))
        print("\n")
    else:
        print("Invalid command")
        print("\n")

```

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

Pentru 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. Avem, desigur, și fișierul „passwd”, folosit pentru a reține utilizatorii

 ca.crt	13.01.2025 18:41
 ca.key	13.01.2025 18:41
 client.crt	13.01.2025 18:41
 client.csr	13.01.2025 18:41
 client.key	13.01.2025 18:41
 passwd	17.01.2025 11:29
 server.crt	13.01.2025 18:41
 server.csr	13.01.2025 18:41
 server.key	13.01.2025 18:41

Certificatele folosite de broker.

Am verificat să mă asigur ca broker-ul folosește certificatele, folosind comenzi openssl pentru interogarea acestuia.

```
→ ~ openssl s_client -connect 192.168.43.212:8883 -showcerts
CONNECTED(00000003)
Can't use SSL_get_servername
depth=0 C = RO, ST = Romania, L = Bucuresti, O = cat, CN = 192.168.43.212, emailAddress = drugeadiana@gmail.com
verify error:num=18:self-signed certificate
verify return:1
depth=0 C = RO, ST = Romania, L = Bucuresti, O = cat, CN = 192.168.43.212, emailAddress = drugeadiana@gmail.com
verify return:1
---
Certificate chain
 0 s:C = RO, ST = Romania, L = Bucuresti, O = cat, CN = 192.168.43.212, emailAddress = drugeadiana@gmail.com
  i:C = RO, ST = Romania, L = Bucuresti, O = cat, CN = 192.168.43.212, emailAddress = drugeadiana@gmail.com
  a:PKEY: rsaEncryption, 2048 (bit); sigalg: RSA-SHA256
  v:NotBefore: Jan 13 15:27:58 2025 GMT; NotAfter: Jan 13 15:27:58 2026 GMT
-----BEGIN CERTIFICATE-----
```

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.

Provocări și Soluții

Prima și cea mai mare provocare a fost partea de securitate, întrucât, orice i-aș fi făcut, plăcuța ESP32 nu voia să se conecteze la mosquito. Tot aveam problema că dădea „Connection denied”, ca după să îmi dau seama că nu făcusem nimic pe partea de conturi. Problema era că plăcuța încerca să se conecteze la un cont ce *nu exista* în baza de date a broker-ului. Așa că am făcut fișierul de parole și conturi și l-am atașat la mosquito.conf.

A doua provocare a fost, desigur, rescrierea vechiului cod pentru a funcționa pe ce aveam nevoie. Din fericire, nu a trebuit să fac foarte multe modificări, logica fiind oarecum aceeași, doar a trebuit să elimin părțile în care plăcuța își lua singură informațiile despre ora răsăritului și apusului pentru București. Am făcut modificarea aceasta tocmai ca să fie mai ușor de configurat programul, fără să trebuiască să se schimbe codul și să se pună iar pe plăcuță.