

IoT System for Forest Monitoring

**Munteanu Alexandru-Constantin
342 C1**

Cuprins

1. Introducere:

2. Arhitectura:

3. Implementare:

 3.1 Conectarea la internet:

 3.2 Trimiterea mesajelor către Brokerul MQTT:

 3.3 Bot-ul de Telegram:

 3.3 Server-ul Flask:

 3.4 Sistemul de alertare și notificare:

4. Vizualizarea și Procesarea de Date:

5. Securitatea:

6. Provocări și soluții:

 6.1 Achiziționarea componentelor:

 6.2 Asamblarea tuturor componentelor pe breadboard:

 6.3 Interferențele dintre componente:

 6.4 Alegerea unei metode care asigura transmiterea sigură a datelor:

 6.5 Valorile neconstante date de senzori:

 6.6 Modulul GSM:

 6.7 Configurare Grafana:

 6.8 Afisarea paginii HTML:

 6.9 Crearea unui certificat self-signed:

 6.10 Configurare Grafana cu HTTPS:

7. Resurse:

1. Introducere:

Schimbările climatice și activitățile umane au contribuit semnificativ la creșterea frecvenței și intensității incendiilor de vegetație și alunecărilor de teren, fenomene care pun în pericol vieți omenești, ecosisteme și infrastructuri critice. Necesitatea unui sistem eficient de monitorizare și avertizare timpurie devine tot mai importantă pentru prevenirea dezastrelor naturale și pentru minimizarea impactului lor socio-economic.

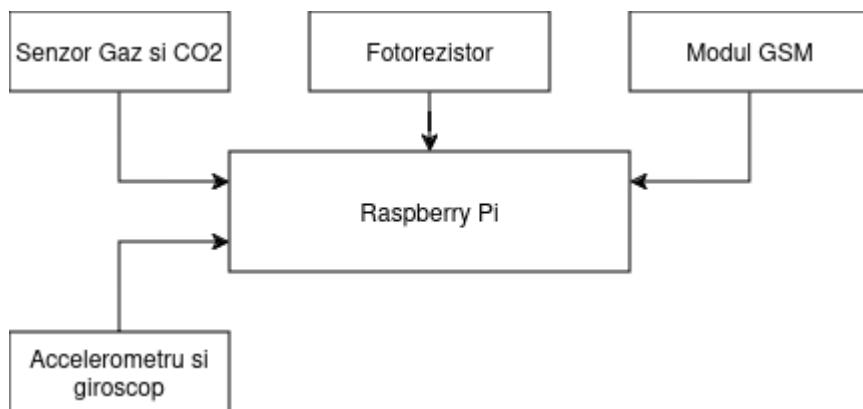
Proiectul își propune să creeze un sistem IoT capabil să detecteze incendiile forestiere prin intermediul unor senzori de lumina și CO₂. De asemenea, ar fi capabil să înregistreze alunecările de teren prin intermediul unui giroscop. Comunicarea ar trebui să se facă folosind protocolul MQTT, peste rețelele 4G/3G pentru a transmite date de la placuta către Broker, și peste internet pentru a comunica cu site-ul web pentru a notifica utilizatorii. Din cauza unor mici (mari) dificultati cu modulul GSM, trimiterea de mesaje se va face folosind MQTT peste Wi-Fi, la fel ca la laboratoare. Brokerul va fi monitorizat folosind Grafana.

Va exista un threshold pentru fotorezistor și un threshold pentru senzorul de CO₂, care atunci când sunt atinse, va fi trimis un mesaj către user. Pentru trimiterea mesajului de avertizare, va trebui fie că: senzorul de CO₂ să depasească threshold-ul sau ambeii senzori să-l depasească. Am ales aceasta metodă pentru evitarea alarmelor false generate de lumina prea puternica care lovește senzorul.

Pentru detectarea alunecărilor de teren, mă folosesc de modulul cu accelerometru și giroscop. Diferenta dintre alunecarea de teren și

simplă tăiere a poate fi făcută cu ajutorul accelerometrului, care detectează o accelerare de intensitate mare și liniară pentru tăierea unui copac și o accelerare neliniară în cazul alunecării de teren.

2. Arhitectura:

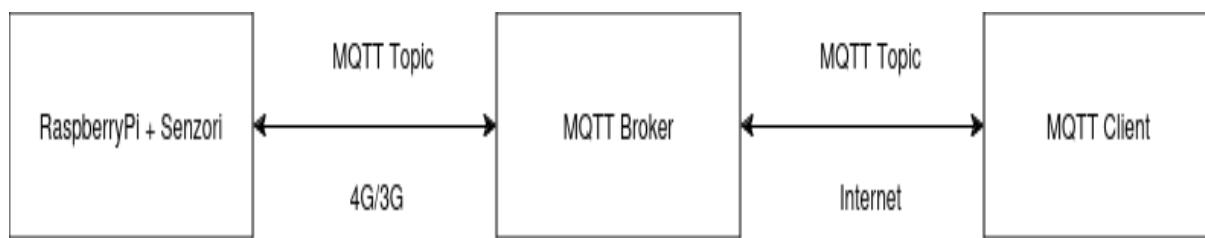


Pentru început, am conectat modulul cu senzorul de gaz, capabil să detecteze și fum, la VBUS, pentru a asigura o alimentare de 5V, ceilalți pini au fost conectați la GND și la pinul 31 al placutei Raspberry Pi Pico W, pentru că acesta este un pin analogic.

Modulul cu fotorezistor este capabil să funcționeze și la 3.3V, dar și la 5V, de aceea, am decis să il conectez și pe acesta la VBUS. Celalți pini sunt GND și un pin de GPIO de pe placuta (pin-ul 32).

Modulul de accelerometru și giroscop va fi conectat la 3V3 (OUT), pinul 36 de pe placuta, la GND, SCL la pinul 12 al placutei și SDA va fi conectat la pinul 11 (pini de GPIO). AD0 va fi lăsat deconectat. Pinul INT (pentru întreruperi) poate fi lăsat deconectat, în funcție de nevoie, altfel, el poate fi conectat la orice pin de GPIO.

Modulul GSM SIL800L, din cauza tensiunii de alimentare necesare aflată în intervalul [3V4, 4V4], va fi alimentat tot de la VBUS, folosind un modul DC-DC care va avea ca output în jur de 3V9-4V4 pentru a asigura funcționalitatea optimă a acestuia. Ca pini, Pin-ul 1 va fi conectat la RXD și pin-ul 2 la TXD.

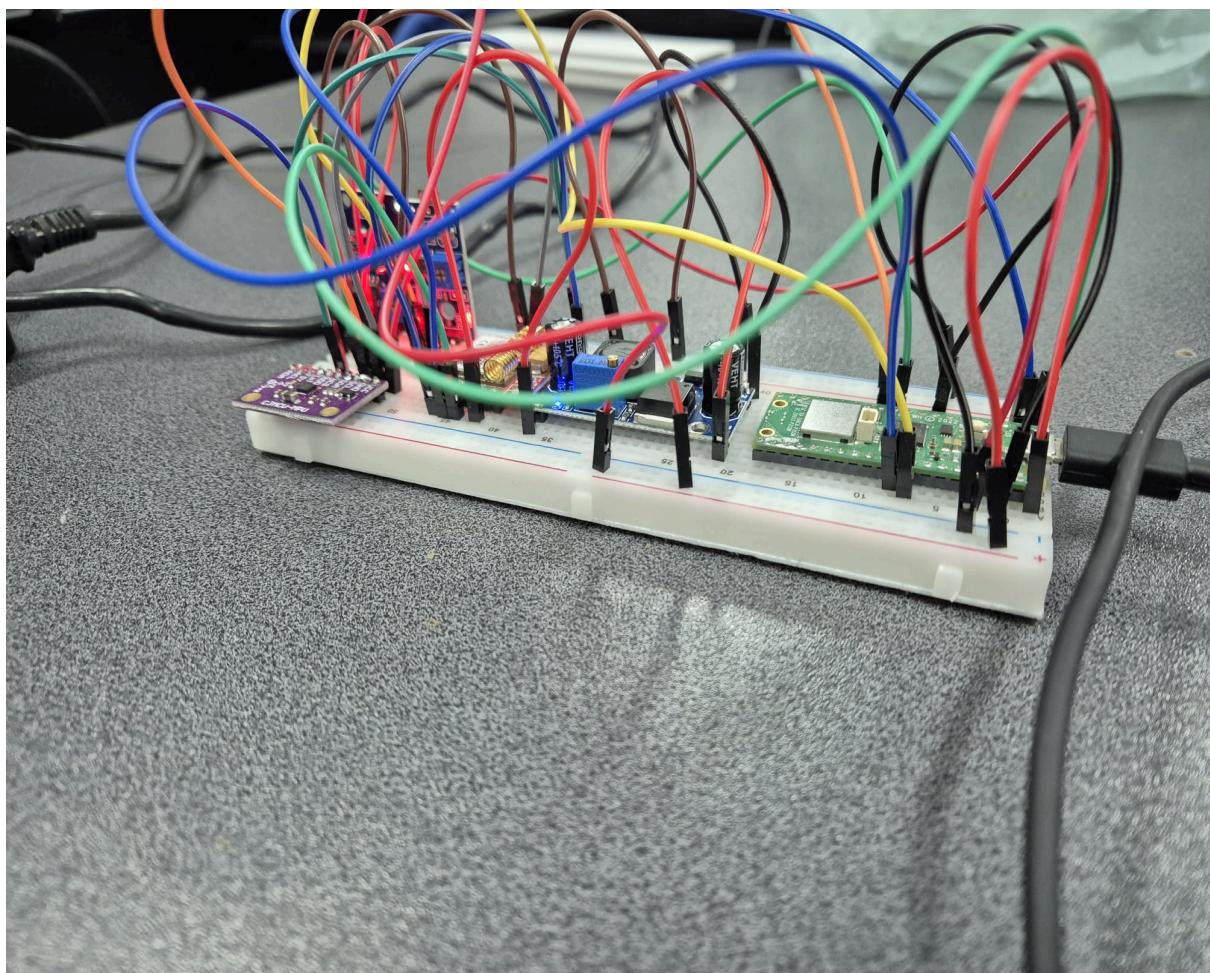


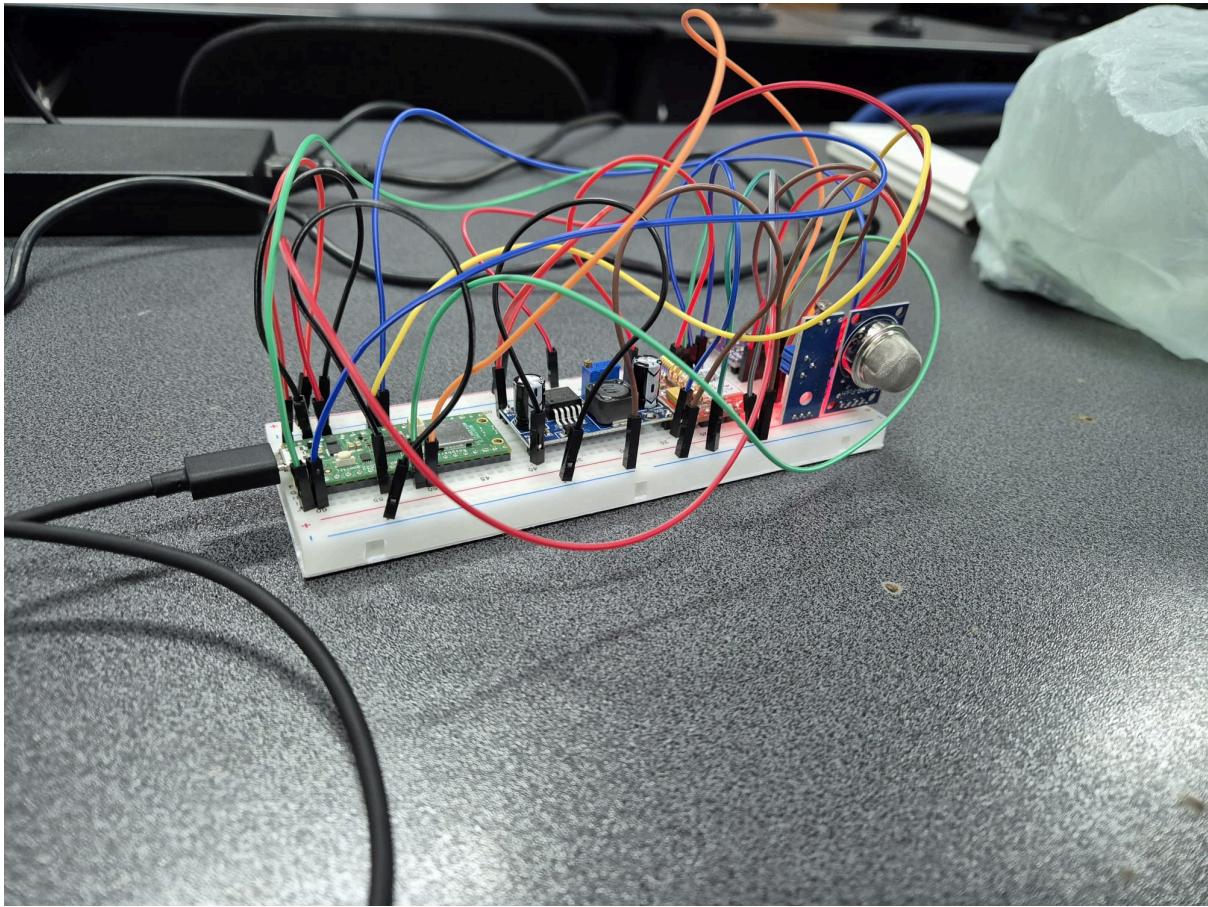
Brokerul va fi monitorizat folosind Grafana, unde am configurat un dashboard cu 3 grafice, unul pentru giroscop, unul pentru fotorezistor și altul pentru senzorul de CO₂.

Dashboard Grafana:



Circuitul fizic:





3. Implementare:

Pentru realizarea proiectului, am început prin conectarea componentelor pe breadboard, la fel cum le-am descris în capitolul anterior, "Arhitectura".

După ce m-am asigurat ca piesele au fost conectate, am instalat MicroPython pe placuta Raspberry Pi Pico W, urmărind indicațiile din laborator, dar alegand un bootloader specific placutei mele, de pe link-ul următor: https://micropython.org/download/RPI_PICO_W/.

Pentru Brokerul MQTT am ales sa folosesc Mosquitto, deoarece a fost deja configurat în timpul laboratorului.

Pentru instalarea mosquitto, am urmat pașii descriși în laborator:

1. Descarcare Mosquitto de pe link-ul:

<https://mosquitto.org/download/>

2. Instalarea Mosquitto și adăugarea acestuia în PATH.

Pentru a rula mosquitto, voi rula într-un Command Prompt comanda: mosquitto -v -c "C:\Program Files\mosquitto\mosquitto.conf". Am ales sa rulez pe Windows, deoarece pe Linux am intampinat cu rularea Brokerului MQTT.

Ca tool de monitorizare, am ales sa folosesc Grafana, deoarece l-am configurat o data în timpul cursului si datorită extensiei de MQTT prezenta deja in Grafana, totul a fost destul de ușor de configurat. O problema pe care am intampinat-o pe parcursul configurarii a fost selectarea unor transformări pentru crearea unor grafice.

În dashboard, am configurat 3 grafice, unul pentru informațiile primite de la accelerometru, unul pentru luminozitate și altul pentru pentru CO2. Datele nu sunt stocate, datele din grafice aparand doar de la pornirea placutei și odată cu transmiterea primului mesaj către MQTT Broker.

Pe placa, vor fi încărcate următoarele fișiere:

- simple.py : contine codul care realizeaza conectarea la internet
- constants.py : contine diferite constante utilizate in fisierul main.py
- mpu6500.py : biblioteca folosita pentru preluarea datelor de la modulul cu accelerometru si giroscop.

3.1 Conectarea la internet:

Pentru a mă conecta la internet am folosit codul din fisierul: simple.py, folosit și în cadrul laboratoarelor, conectându-mă la hotspotul făcut de laptop.

3.2 Trimiterea mesajelor către Brokerul MQTT:

Mesajele sunt trimise pe topicul "senzori" sub forma unui dicționar, pentru a putea fi mai ușor tratate de către Grafana și afișate mai ușor în graficele din Dashboard.

3.3 Bot-ul de Telegram:

Pentru a trimite mesaje către chat-ul de Telegram atunci când s-a produs un hazard, am urmat pașii descriși în laborator:

1. Am folosit token-ul API furnizat de BotFather și chat id-ul rezultat din conversatie.
2. Am folosit metoda post din modului requests către un URL custom de forma:
https://api.telegram.org/bot{bot_token}/sendMessage, pentru a trimite mesajului.

3.3 Server-ul Flask:

Am creat un server self hosted folosind Flask. Acesta este simplu, alcătuit din doar din două endpoint-uri, root ("/") unde vor fi randate paginile HTML în funcție de câmpul "Warning". Endpoint-ul "/info", va trata request-ul primit de la placuta.

Serverul va folosi o conexiune https, creata cu ajutorul unui certificat self signed folosind comenziile următoare:

```
openssl genrsa -out ./cert/CA/rootCA.key 4096
```

```
openssl req -new -x509 -days 365 -key  
./cert/CA/rootCA.key  
-subj="/C=R0/ST=Bucharest/O=Poli/CN=Poli CA" -out  
./cert/CA/rootCA.crt
```

```
openssl req -newkey rsa:2048 -nodes -keyout  
./cert/CA/forestiot.local.key  
-subj="/C=R0/ST=Bucharest/O=Poli/CN=*.forestiot.local  
" -out ./cert/CA/forestiot.local.csr
```

```
openssl x509 -req -extfile <(printf  
"subjectAltName=DNS:forestiot.local,DNS:*.forestiot.local") -days 365 -in ./cert/CA/forestiot.local.csr  
-CA ./cert/CA/rootCA.crt -CAkey ./cert/CA/rootCA.key  
-CAcreateserial -out ./cert/CA/forestiot.local.crt
```

Pentru a simplifica accesul catre homepage-ul proiectului, am configurat nginx pentru a face continutul afisat de catre serverul Flask accesibil prin forestiot.local, la fel, am facut si pentru a face grafana accesibil prin: forestiot.local:3000/.

3.4 Sistemul de alertare și notificare:

Sistemul de alertare și notificare este constituit din serverul și BOT-ul de Telegram, ambele descrise mai sus. De fiecare data cand placuta va trimite un mesaj către MQTT Broker, aceasta va trimite și un POST request către server. Pe server, este verificat campul "Warning" din mesaj, care va indica dacă s-a produs un hazard și ce fel de hazard a avut loc, iar site-ul va afișa pagina HTML conform acestuia. Campul "Warning" poate avea 3 valori: None, va fi afișat "No hazards" și o pagina HTML doar cu mesajul menționat anterior; "Fire", va fi afișată pagina aferentă unui incendiu și "Landslide", va fi afișată pagina aferentă unei alunecări de teren.

Pe langa un mesaj de avertizare, o pagina care anunță un hazard va contine si un tabel în care se vor afla cele mai importante informații captate de senzori la momentul producerii hazardului. La început, acest tabel va avea valori dummy, care vor fi înlocuit ulterior cu cele reale, luate de la senzori.

Pagina afișată atunci cand există un incendiu:

Warning: Fire Detected!

Please be cautious and take necessary precautions.

Fire Warning

Parameter	Value
Luminosity	12487.0
CO2	30517.4
X Axis	-0.5027823
Y Axis	0.1580173
Z Axis	9.504981

Pagina afișată atunci cand există o alunecare de teren:

Warning: Landslide Detected!

Please evacuate the area immediately and stay safe.

Landslide Warning

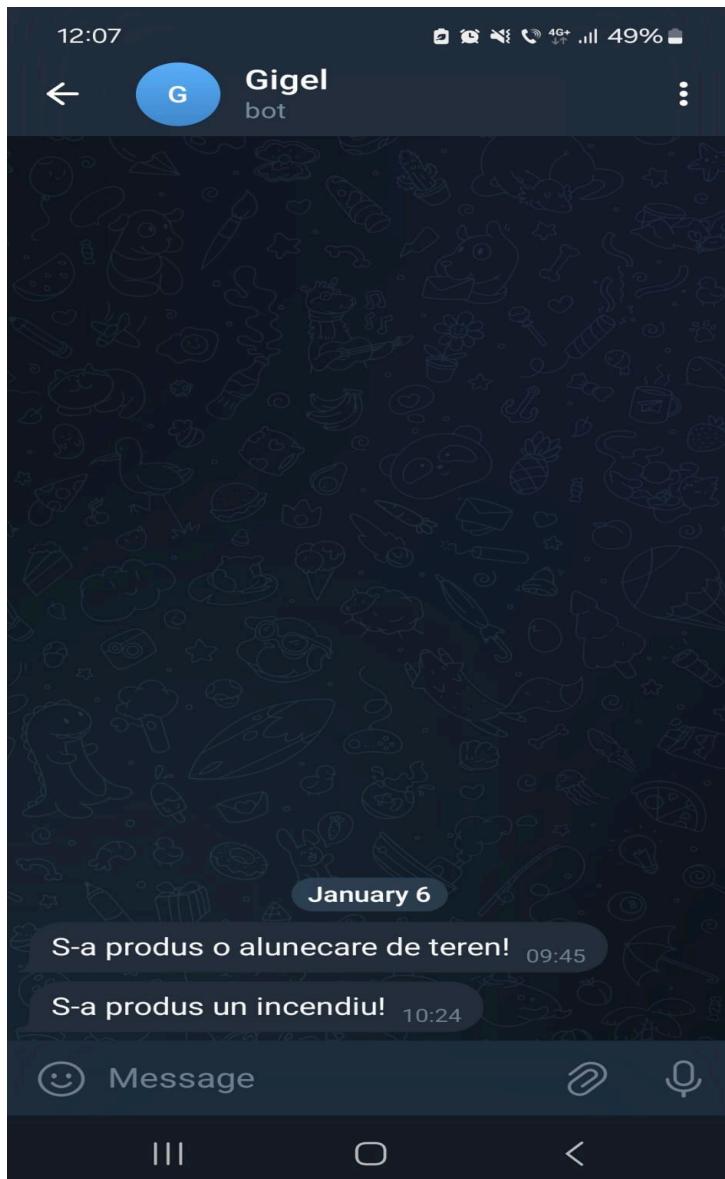
Parameter	Value
Luminosity	12288.95
CO2	28597.25
X Axis	2.973598
Y Axis	0.05985504
Z Axis	9.315839

Pagina afișată atunci cand nu există probleme:

No Hazards

Pentru a nu verifica mereu site-ul pentru detectarea unor hazarduri, voi folosi BOT-ul de Telegram pentru a trimite un mesaj utilizatorului de fiecare data cand se produce un hazard. Mesajele sunt trimise automat, de fiecare data cand sunt depasite threshold-urile setate pentru fiecare senzor.

Mesaje trimise de către BOT-ul de Telegram:



4. Vizualizarea și Procesarea de Date:

Inițial, pentru a citi datele de la modulul cu accelerometru și giroscop am folosit un filtru trece-jos pentru a ma asigura ca iau în considerare doar "slow changes" ale valorilor, încercând să compensez

pentru imprecizia senzorului. Însă, am observat ca aceasta metoda nu producea rezultate satisfăcătoare, avand foarte multe valori false-positive, în care era detectată o mișcare "rapidă" atunci când nu era cazul. Pentru a rezolva aceste probleme am folosit o biblioteca pentru modulul MPU6500 (<https://github.com/tuupola/micropython-mpu9250/blob/master/mpu6500.py>) și am observat rezultate mult mai bune. Aceleași probleme le-am întâmpinat și la giroscop, unde doar citeam valorile și le transformăm din intervalul [0, 65k] în [-32k, 32k]. Chiar dacă valorile date de giroscop nu influență la fel de mult rezultatele, deoarece alunecările de teren erau de cele mai multe ori date de deplasarea liniara calculată cu ajutorul datelor obținute de la accelerometru, am ales să folosesc biblioteca menționată anterior și pentru a citi valorile înregistrate de giroscop.

Pentru a detecta un incendiu, citesc valorile date de senzorul de CO2 și de luminozitate, pe care, pentru a obține anumite valori cu mai puține fluctuații mai mici, am decis că înainte de obținerea rezultatului să citesc 20 de sample-uri și voi returna media valorilor. Valorile calculate sunt comparate cu anumite threshold-uri, iar dacă ambele sunt depasite, voi trimite avertizarea de incendiu.

Pentru a detecta un incendiu, citesc valoare analogică de la senzori (cel de CO2 și de la fotorezistor) și dacă, împreună trec de un anumit prag, voi trimite avertizarea de incendiu.

5. Securitatea:

Initial, am folosit un certificat self signed, iar pentru crearea acestuia m-am folosit de biblioteca: "cryptography", mai exact de modulul x509, care este folosit pentru crearea și utilizarea certificatelor X.509. Am creat o cheie privată folosind funcția "generate_private_key"

din rsa, cu o dimensiune a cheii de 2048 de biti, deoarece aceasta dimensiune parea sa fie cea mai des intalnita. După creare, am salvat cheia in fisierul "key.pem", necriptata. Următorul pas a fost crearea unui subiect și a unui issuer. Certificatul a fost făcut cu ajutorul clasei CertificateBuilder din biblioteca x509.

Nu am reusit sa fac ca acest certificat sa fie recunoscut de browser, de aceea am construit unul nou folosind comenziile de openssl date mai sus.

Asigurata de comunicarea HTTPS a serverului. Placuta trimite POST request-ul prin intermediul metodei post din modulul de urequests care este capabil sa foloseasca intern modulul de ssl pentru crearea unor "secure sockets - over HTTPS".

6. Provocări și soluții:

6.1 Achiziționarea componentelor:

Prima problema a constat în alegerea unor componente care să funcționeze la tensiuni de alimentare similare. Am reușit să achiziționez un senzor de gaz / CO₂ care are nevoie de o tensiune de alimentare de 5V, un modul cu giroscop și accelerometru care este alimentat la 3V3 și un modul GSM care are nevoie de alimentare între 3V4 și 4V4. Am reușit să alimentez primele 2 componente la pinii placutei Raspberry Pi Pico W, iar pentru modulul GSM, a achiziționat un modul DC-DC pentru a reduce tensiunea de alimentare de la 5V la 4V2.

6.2 Asamblarea tuturor componentelor pe breadboard:

Din cauza numărului mare de componente dar și a spațiului limitat de pe breadboard, a fost dificil să găsesc o configurație capabilă să acomodeze toate piesele.

6.3 Interferențele dintre componente:

Din cauza problemelor de la punctul (1), cablurile ajung să fie destul de aglomerate, la fel și senzorii și unele componente produc rezultate influențate de alte componente (senzorul de CO₂ își schimba valoare atunci când fotorezistorul detectează lumina). Pentru a rezolva aceasta problema, am ales să modific Pin-ul la care este conectat fotorezistorul, astfel, cele 2 componente vor folosi diferite ADC-uri.

6.4 Alegerea unei metode care asigura transmiterea sigură a datelor:

Prima data am încercat să folosesc OpenSSL care presupunea instalarea diferitelor tool-uri pe Windows, după aceea am ales să merg pe implementarea folosind un server Flask cu HTTPS. La final am pastrat implementarea serverului Flask care primește anumite certificate ca argument, acestea fiind create cu ajutorul openssl, dar pe Linux, nu pe Windows :).

6.5 Valorile neconstante date de senzori:

Deși la început, implementarea unui filtru "trece-jos" pentru a reduce zgomotul a părut suficientă, încă intampinam suficient zgomot, de aceea am ales să mă folosesc de o bibliotecă deja implementată, obținând rezultate mai bune.

6.6 Modulul GSM:

După mai multe încercări, nu am reușit să fac modulul GSM să se conecteze la rețea. Am încercat să fac debug folosind comenzi de AT, neprimind niciun răspuns, nereusind să îmi dau seama care este problema.

6.7 Configurare Grafana:

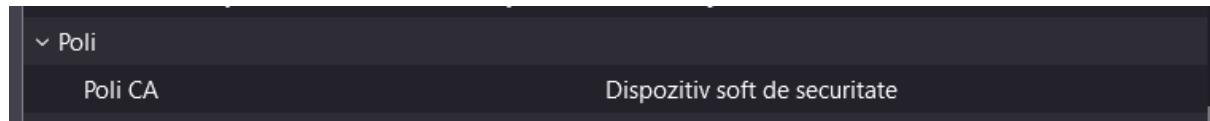
În timpul configurării am întâmpinat probleme la alegerea transformărilor corecte astfel încât datele trimise să fie automat afisate în grafic, care s-a rezolvat prin trial-and-error cu opțiunile date de Grafana. De asemenea, o alta problemă pe care am întâmpinat-o a fost formatul mesajului către MQTT Broker astfel încât acesta să fie interpretat automat de Grafana. La final, am ales să merg pe JSON, fiind parsat automat.

6.8 Afisarea paginii HTML:

După trimiterea unor mesaje și afișarea paginii de eroare, aceasta se schimba după fiecare mesaj trimis, fiind destul de greu de reperat un mesaj de avertizare. De aceea, am decis să pagina de avertizare cu una care afișează "No Hazards" doar după un număr fix de mesaje care anunță ca totul este în regula.

6.9 Crearea unui certificat self-signed:

Prima dată am reusit să creez un certificat self-signed folosind un script. Acest certificat nu era însă recunoscut de către browser, chiar dacă era adăugat în lista de certificate ale browser-ului respectiv. De aceea am decis să revin iar la opțiunea folosirii openssl, dar, pentru a evita mai multe probleme, am decis să imi construiesc și o domeniu local, folosind nginx (lucru care a dus și la apariția mai multor erori). După crearea domeniului local, am întâmpinat mai multe erori, tot din cauza că certificatul nu era recunoscut de către browser, lucru care a dus la crearea unui rootCA cu ajutorul comenziilor de SSL. După adăugarea certificatului în lista de certificate recunoscute de către browser, lucrurile par să meargă în momentul de fata.



6.10 Configurare Grafana cu HTTPS:

Din cauza domeniului configurat anterior, s-au produs anumite erori la configurarea HTTPS pentru Grafana, incercand diferite metode de a putea ajunge la dashboard folosind domeniul. Am incercat sa creez un nou subfolder in domeniu forestiot.local/grafana, lucru care nu a mers. Am decis la final sa ma folosesc de nginx pentru a face grafana accesibil folosind: forestiot.local:3000.

7. Resurse:

<https://ocw.cs.pub.ro/courses/priot/laboratoare/04>
<https://ocw.cs.pub.ro/courses/priot/laboratoare/05>
<https://ocw.cs.pub.ro/courses/priot/laboratoare/06>
<https://ocw.cs.pub.ro/courses/priot/laboratoare/08>
<https://mosquitto.org/>
<https://medium.com/gravio-edge-iot-platform/how-to-set-up-a-mosquitto-mqtt-broker-securely-using-client-certificates-82b2aaaef9c8>
<https://invensense.tdk.com/download-pdf/mpu-6500-datasheet/>
<https://www.instructables.com/Accelerometer-Gyro-Tutorial/>
<https://lastminuteengineers.com/mq2-gas-senser-arduino-tutorial/>
<https://www.pololu.com/file/0j309/mq2.pdf>
<https://www.youtube.com/watch?v=g-NvPPEj3oQ>
https://www.youtube.com/watch?v=7VW_XVbtu9k&t
<https://www.youtube.com/watch?v=5HuN9iL-zxU&t>
<https://www.educba.com/flask-https/>
<https://blog.miguelgrinberg.com/post/running-your-flask-application-over-https>
<https://core.telegram.org/bots/features>
https://makeblock-micropython-api.readthedocs.io/en/latest/public_library/Third-party-libraries/urequests.html
<https://grafana.com/docs/grafana/latest/setup-grafana/configure-grafana/>
<https://grafana.com/docs/grafana/latest/setup-grafana/>
<https://medium.com/activewizards-machine-learning-company/intro-to-grafana-installation-configuration-and-building-the-first-dashboard-bf408747e6a8>
<https://grafana.com/docs/grafana/latest/setup-grafana/set-up-https/https://grafana.com/docs/grafana/latest/setup-grafana/set-up-https/>