

BreatheloT

Ilinca Sebastian-Ionut 341CA

I. Introducere

I.I Descriere generală

Proiectul “BreatheloT” reprezintă un ansamblu de componente ce lucrează împreună pentru a crește calitatea vieții consumatorului prin aducerea informațiilor relevante despre aerul din casa acestuia la un pas distanță, oriunde s-ar afla acesta. Fie ca dorim un consum energetic mai mic prin a porni purificatorul de aer când mergem spre casa sau sa monitorizam constant calitatea aerului inspirat de către locuitorii casei, ca de exemplu copilul ,pentru a i asigura un mediu cât mai bun si sănătos , iar pentru acest aspect ,proiect se potrivește extraordinar.

Sistemul este reprezentat de integrarea unui purificator de aer de dimensiuni mari (Xiaomi Smart Air Purifier 4) cu o plăcută ESP32 ce are modul de Wi-Fi integrat(necesară atât pentru transmiterea informațiilor de la aparat cât si pentru a putea face posibilă controlarea acestuia de la distanță), prin intermediul unui server local.

Datele colectate(cum ar fi umiditatea, numărul de particule PM2.5, scorul asignat aerului) vor fi transmise către o platforma cloud cu ajutorul plăcuței ESP32 menționată în rândurile de mai sus. Acestea, din urma, vor fi accesate cu ajutorul unei aplicații mobile, totodata, prin aceeași aplicație mobilă se dorește controlul aparatului în ceea ce privește modurile de operare(turbo, slow, medium, etc), fiind disponibil pentru fiecare în parte un buton.

I.II Obiective

1. Monitorizarea datelor despre aerul din casa în timp real datorită utilizării senzorilor de înaltă performanță prezenți deja în purificator.

2. Setarea vitezei ventilatorului cât si pornirea/oprirea aparatului cu ajutorul actuatorilor.
3. Transmiterea datelor către o platforma cloud cunoscuta pentru stocare si vizualizare ulterioară.
4. Controlul aparatului de la distanță printr-o aplicație mobilă.
5. Optimizarea interacțiunii utilizatorului printr-o interfață prietenoasă si accesibilă.

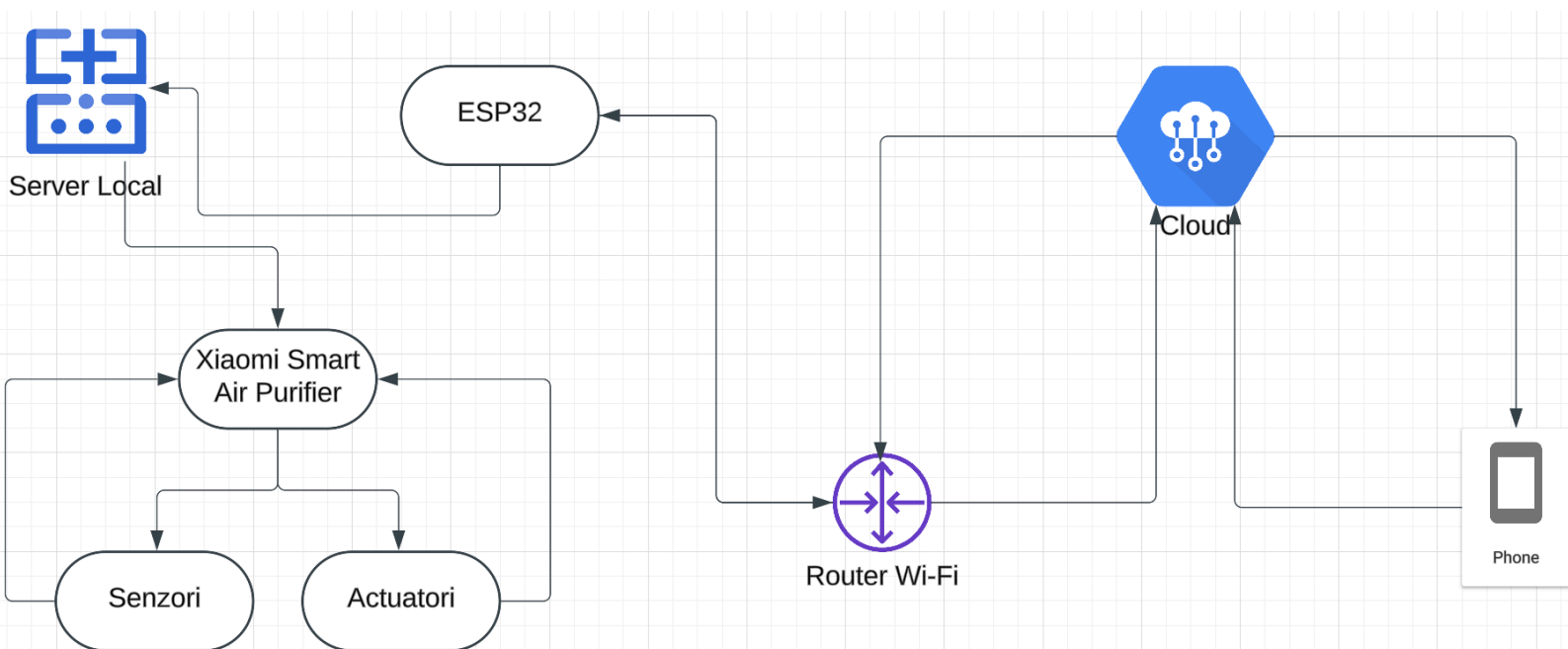
II. Arhitectura

II.1 Descrierea componentelor

- ESP32 -> punct central pentru colectarea datelor de la serverul local si transmiterea acestora către platforma cloud.
Microcontroller cu suport Wi-Fi pentru a fi posibilă obținerea si trimiterea datelor menționate mai sus.
- Xiaomi Smart Air Purifier 4 -> dispozitiv principal ce include multitudinea de senzori utilizați pentru măsurarea calității aerului cât si actuatorii necesari pentru filtrarea aerului.
- Actuatori -> sistemele interne ale aparatului ce sunt controlate prin aplicație (aceste sisteme sunt formate din ventilatoare, dar si din mecanismele de curățare cât si din piese responsabile pentru ionizarea aerului)
- Senzori -> senzorii încorporați in purificator(PM2.5, temperatura, umiditate) vor fi utilizați pentru a monitoriza cu precizie condițiile aerului din încăpere.
- Aplicatie de control: aplicatie mobilă cu diferite funcții pentru monitorizare si control, prezentate printr-o interfață prietenoasă.

- Server Local -> dispozitiv ce transmite date către ESP32 și interceptează comenzi, transmise mai departe purificatorului.
- Cloud -> constituit din două servere, unul controlat de ThingsBoard (pentru livrarea informațiilor partajate de aparat, nefiind necesare metode de securitate extra, întrucât atacatorul nu are ce face cu niște simple informații) și celălalt de către HiveMQ (comenzile necesită criptare TLS datorită faptului că nu dorim ca un atacator să trimită comenzi către aparat).

II.II Diagrama topologiei rețelei



I.III Protocoale de comunicație folosite

- Server Local <-> Xiaomi Smart Air Purifier 4 -> protocolul MLoT este ales întrucât reprezintă o soluție solidă și sigură în comunicația cu aparatul de purificat aerul (disponibil prin miocli).

- ESP32 <-> Server Local -> MQTT reprezintă o alegere facilă pentru partajarea de comenzi și informații despre statusul actual putând împărți pe categorii tipurile de mesaje primite, cu alte cuvinte, ne putem abona la topicul “info” pentru statusul aparatului și aerului, iar pentru a seta modul de funcționare al ventilatoarelor să ne abonăm la “ventsCommand”.
- ESP32 <-> Cloud -> protocolul HTTP va asigura această comunicare privind transmiterea informațiilor, pe când MQTT este folosit pentru a intercepta comenzi.
- Cloud <-> Device -> protocolul HTTP este o alegere ușoară de făcut întrucât se mulează perfect pe cerințele proiectului.

III. Implementare

Conform cerinței, trebuie să descrii pașii de configurare a hardware-ului, software-ului, dar și a sistemului de alertare și notificare. Această documentație își propune să cuprindă acești pași în secțiuni diferite clar delimitate, cu toate acestea, datorită integrării componentei software cu hardware împreună cu sistemul de alertare și notificare.

Asadar, este posibil să fie menționate alte secțiuni (exemplu: în secțiunea de hardware este necesară detalierea unei chestiuni software). De asemenea, serverul local, serviciul cloud, cât și aplicația, nu sunt luate în considerare întrucât pentru acestea ne interesează strict programul pe ele.

III.I Configurarea hardware-ului

- Purificatorul de aer este prima componentă cu care începem configurarea. Acestuia îi trebuie asigurată alimentarea de la o sursă de energie stabilă, de preferat fata de priza normală este o sursă de alimentare neprelucrată. Prin folosirea instrucțiunilor prezentate pe website-ul

comerciantului, asiguram conectivitatea prin wi-fi la router(ne folosim si de un cont Xiaomi). Mai departe, ne folosim de un program prezent in dockerhub(componenta software, pull-ul va fi automat realizat). Am scris un fișier docker-compose ce asigura prin anumite variabile de mediu parcare input-ului către program cu credentialele contului Xiaomi. Ne conectam la localhost, pe portul definit in docker-compose, si așteptăm sa ne returneze atât token-ul ce validează comenzile primite de către purificator, cât si adresa IP a acestuia.

- Placuta ESP32 are nevoie de un firmware special pentru a putea rula cod MicroPython, așadar operația de flashing este realizată pe modul. Pentru acest lucru a fost nevoie, datorită necesității unui mediu virtual pentru esptool, de schimbarea permisiunilor pe portul la care este conectat plăcută, in cazul meu acesta fiind `ttyUSB0(sudo chmod 777 /dev/ttyUSB0)`.

III.II Configurarea software-ului

- Pentru a putea fi citită de către laptop portul la care este conectat ESP32, a fost nevoie de descărcarea driver-ului CP210x de la Silicon Labs(după ce a fost inspectată plăcuta pentru a verifica daca nu trebuie in loc CH340). Totodata, firmware-ul a fost descărcat de pe website-ul oficial MicroPython. Pentru rularea codului pe plăcută(cât si pentru flashing) a fost nevoie atât de esptool, cât si de adafruit-ampy, iar pentru a interactionare a fost folosită extensia PyMakr de pe VSCode. In codul MicroPython sunt necesare configurarea parametrilor precum credentialele wifi-ului, ale contului de la HIVEMQ, cât si configurarea id-ului pentru brokeri.
- Serverul local a avut nevoie de configurarea unui docker intern drept broker MQTT(la rândul lui, docker-compose a avut nevoie de un fișier de configurare), dar si de instalarea unui mediu virtual python, unde a fost instalată dependența miiocli pentru a putea rula scripturile puse la

dispoziție(scripturi ce asigura rularea comenzilor cât si obținerea informațiilor in timp real). Script-urile s-au configurat in funcție de ip-ul si token-ul prezent la acel moment, necesita o configurare ulterioară in cazul schimbării locației dispozitivului. A fost necesară si configurarea id-ului client pentru MQTT.

- Aplicatia mobila, in cod, are o configurare extra realizată pentru îmbunătățirea experienței, si anume un timer cu ajutorul căruia sunt trimise mereu cereri de obținere a datelor “fresh” stocate in cloud.

III.III Configurarea sistemului de alerta si notificare

- M-am folosit de librăria smtplib din python pentru a putea trimite mail către utilizator in momentul in care calitatea aerului este deplorabilă. Cu toate acestea, un pas auxiliar a fost necesar, configurarea unei noi parola pe yahoo pentru a permite aplicației terțe sa trimită mesaje.

IV. Vizualizare si procesare de date

- Dorind o interfața simplificata si ușor de folosit, s-a ales ca metoda de vizualizare un scurt text, inspirat de modelul JSON, prin care utilizatorul sa poată citi ușor diferite valori, cum ar fi calitatea aerului, Child Lock si multe altele. Pentru a accesa aceasta vizualizare, este disponibil in Home un buton dedicat.
- Metoda de procesare a datelor începe de la interogarea purificatorului pentru date, mai apoi fiind necesară parcare a acestora si introducerea lor într-un dicționar sub forma de string(a valorilor), totodata encodeate cu %20 pentru a putea fi înțelese de ThingSpeak la trimiterea request-ului(din partea plăcuței după ce primește acest dicționar de la broker-ul MQTT local, realizat cu Docker). Fiind acum datele disponibile in ThingSpeak, Flutter le interpretează pentru a putea fi afișate ca text in aplicația mobila.

V. Securitate

- Pentru a folosi securitatea prin criptare si autentificare doar acolo unde este nevoie, transmiterea informațiilor despre aer prin ThingSpeak se realizează doar cu o cheie pentru API generată pe site.
- Cu toate acestea, comenzile reprezintă un lucru exclusiv confidențial, așadar accesul este autorizat la broker-ul HiveMQ, de asemenea conexiunea fiind TLS, pentru a asigura criptarea datelor extrem de sensibile.
- Asadar, metodele de securitate implementate si abordate sunt următoarele: un API key dedicat pentru a putea transmite informații despre aer si setări ale device-ului si criptare alături de autentificare in ceea ce privește trimiterea comenzilor. De mentionat este si ca trimiterea notificării prin email este de asemenea securizata de autentificare si criptare.

VI Provocari si soluții

VI.I Provocari intampinate

De-a lungul proiectului, dificultăți multiple au fost întâmpinate, de la provocări ce s-au rezolvat in 30 de minute, pana la lucruri nefuncționale ce au avut nevoie de o întreagă zi pentru a fi rezolvate.

Aceste dificultăți vor fi categorizate după gradul de ușurință cu care au fost rezolvate.

Dificultati rezolvate in scurt timp(20-30 de minute):

- Conectare eșuată a plăcuței ESP32 la Wi-Fi. Deși parola si ssid-ul sunt corecte, plăcuta, din diverse motive, eșuează a se conecta la router. Metoda prin care a fost soluționată aceasta problema a fost una de tip brute-force, prin rularea de mai multe ori cu pyMakr a codului python, pana când conectarea a reușit.
- Placuta a ramas fără memorie datorită print-urilor multiple si cantității mari de informație, acest lucru rezolvându-se prin folosirea unui garbage collector apelat la fiecare final de funcție.

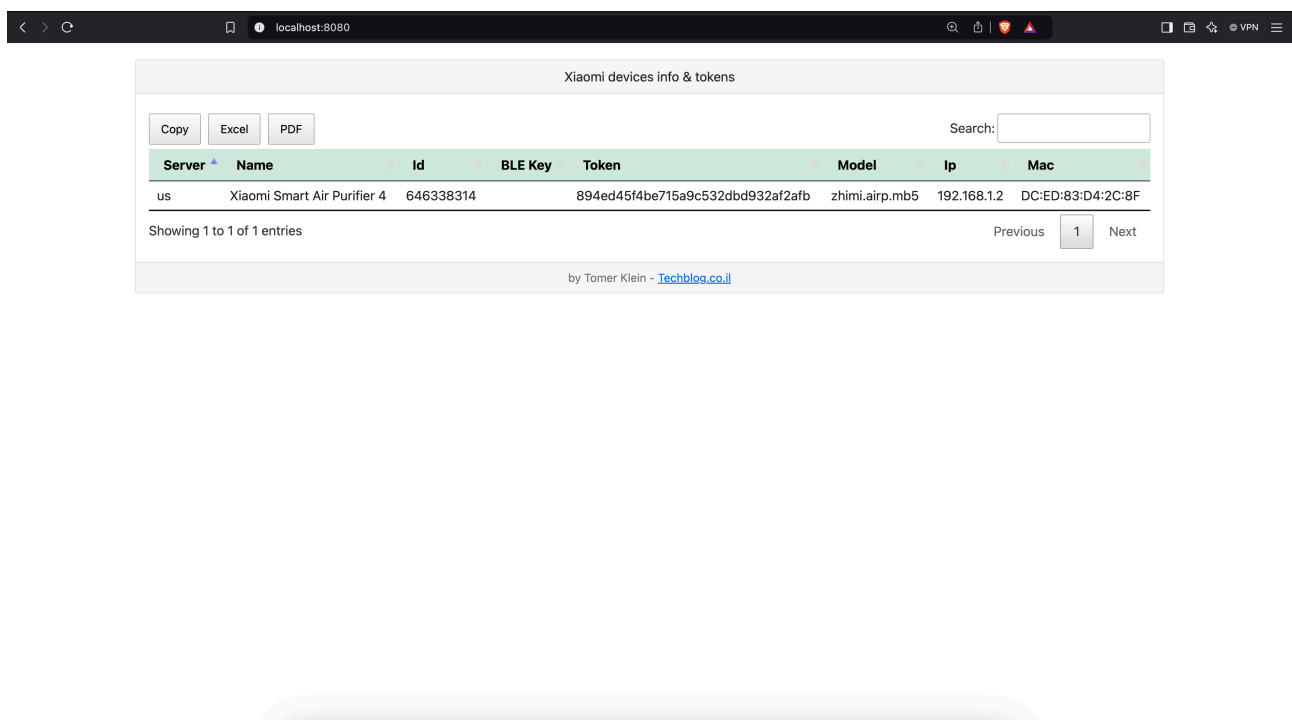
- Parsarea datelor primite de la purificator in serverul local. Întrucât datele veneau alături de space-uri si caractere de genul (m₃, °C), daca nu ar fi fost parsate, plăcuta ar fi trimis către ThingSpeak un url invalid. Problema a fost descoperită când in aplicație se observau numai câmpuri null la datele specifice aparatului.
- Trimiterea unui email de avertizare a fost încă o problema întâmpinată de dificultate ușoară. Practic yahoo are un mod de a nu lasa aplicațiile terțe sa trimită emai-uri in locul tău, întrerupând comunicația, chiar daca parola e buna. Pentru a putea face totuși asta, a fost nevoie sa intru pe email pe website si sa selectez opțiunea de a genera o parola specială pentru o aplicație terță, acest lucru făcând ca lucrurile sa meargă perfect. De asemenea, a fost nevoie de tls in sesiunea de comunicare pentru trimiterea mail-ului.
- Pentru a putea migra aplicația pe un telefon, o subscripție pentru Apple Developer a fost necesară, cu toate acestea, nici partea de demo nu a funcționat întrucât pe telefon nu am avut suficientă memorie pentru a încarcă varianta de iOS Beta. Workaround-ul folosit a fost simularea unui telefon pe laptop.

Dificultati rezolvate in timp mediu(2-4 ore):

- Crearea unui stateful widget in flutter care sa fie updated la fiecare 3 secunde. Majoritatea timpului petrecut a fost pentru a înțelege documentația pentru a o putea aplica.
- Refuzarea plăcuței de a încarca codul. Pentru a putea scrie cod microPython ca mai apoi sa fie încărcat si sa meargă pe plăcută, aceasta a trebuit ‘formatata’, lucru care a durat mult timp din cauza a doua chestii: cablul folosit nu avea posibilitatea trasnmiterii de date, ci doar încărcarea dispozitivului conectat; refuzarea de către port a conectării pentru formatare, lucru rezolvat prin schimbarea permisiunilor.

Dificultati rezolvate greu(7-10 ore):

- Aparatele de la Xiaomi sunt controlate prin intermediul unui utilitar in Command Line. Din acest motiv, pentru o extra securitate, o comanda scrisă in CLI va avea nevoie de un token asignat dispozitivului. Inițial, încercarea de a obține token-ul a fost prin reverse engineering, conectându-mă la router si capturând pachete de date cu WireShark, ca mai apoi sa le analizez. Cu toate acestea, token-ul obținut era departe de a fi cel bun pentru ca datele erau criptate. A fost apoi încercarea de a găsi ceva in aplicația mobila, fără niciun rezultat. Ceea ce a mers in schimb a fost rularea unui docker-compose pe o imagine ce promitea ca exact asta face, pentru un cont Xiaomi dat, caută in Cloud-ul celor de la Xiaomi si returna pe un port dorit numele device-urilor, id-ul, modelul(lucru ce ajuta in utilizarea unor biblioteci python pentru a ușura procesul(cu toate acestea, nu merge scris cod python pentru zhimi.airp.mb5 întrucât nu exista suport pentru acest aparat deocamdată), token-ul, adresa MAC cât si IP-ul.



The screenshot shows a web browser window with the address bar displaying 'localhost:8080'. The web application has a title 'Xiaomi devices info & tokens'. At the top, there are buttons for 'Copy', 'Excel', and 'PDF', and a search bar. Below these is a table with the following columns: 'Server', 'Name', 'Id', 'BLE Key', 'Token', 'Model', 'Ip', and 'Mac'. The table contains one entry for a device named 'Xiaomi Smart Air Purifier 4' on the 'us' server. Below the table, it says 'Showing 1 to 1 of 1 entries'. At the bottom, there is a footer that reads 'by Tomer Klein - [Techblog.co.il](https://techblog.co.il)'.

Server	Name	Id	BLE Key	Token	Model	Ip	Mac
us	Xiaomi Smart Air Purifier 4	646338314		894ed45f4be715a9c532dbd932af2afb	zhimi.airp.mb5	192.168.1.2	DC:ED:83:D4:2C:8F

VI.II Solutii Software folosite

Pentru implementarea acestui proiect a apărut o multitudine de întrebări valabile pentru orice proiect 'free-style'. Cu ce

tehnologie lucrez? De ce am nevoie? Vreau sa fie super-fast sau doar să-și facă treaba dar să-l termin rapid? Este ușurința de utilizare mai presus decât viteza de execuție? Care date au nevoie de extra-safety? Ce pot folosi pentru a avea o aplicație cross-platform care sa satisfacă cerințele utilizatorului astfel încât sa exprim minimalism pentru a nu fi plictisit dar si funcționalitate?

In rândurile următoare voi descrie răspunsurile la întrebările de mai sus cumulate in paragrafe.

Concluzia la care am ajuns, pentru a îndeplini cerințele proiectului a fost ca am nevoie de un mediu de stocare cloud(poate chiar doua, cu doua protocoale de comunicație diferite). Datorită faptului ca aceasta materie este despre cum microprocesoare si microcontrollere ajuta la interconectarea multiplelor dispozitive, am ales sa am integrată o **plăcută ESP32**(pe lângă cea din aparat), dar acum exista următoarea problema, aceasta plăcută fiind foarte mica ca memorie si putere de procesare si nu pot pune pe ea utilitarul necesar pentru conectarea cu purificatorul, așadar am nevoie de un **server local**.

Pentru a lega cele doua componente bold-uite mai sus, am avut nevoie de o componenta ce primește si transmite mesaje mai departe, care sa funcționeze pe partea de întreruperi foarte bine; așadar am folosit protocolul MQTT host-uit pe serverul local. Fiind facilă utilizarea acestui serviciu prin Docker, am folosit aceasta varianta, fiind una foarte comoda care are rezultate imediate si rulează indiferent de platforma.

Într-adevăr, performanta este puțin compromisă, dar in acest proiect focusul a fost pe **funcțional**, dar si pe implementarea cât mai rapidă si corecta, cu atât mai mult ca este un serviciu auxiliar de nevoie, juxtapunând astfel ca decizia cea mai buna e folosirea serviciului descris mai sus, adăugând un lucru in lista “chestiilor care merg si atât, fără sa am nevoie de a depana probleme”.

Un lucru necesar de obținut fără de care tot proiectul ar fi fost inutil este prezenta si aflarea unui token ce permite controlul aparatului. Pentru descoperirea acestuia, proiectul a depins de o

image docker ce caută prin serverele companiei producătoare si afla acest token.

Este imperios necesară menționarea faptului ca acest serviciu face parte din categoria “nu știam că-mi trebuie așa ceva, dar am aflat asta pe parcurs”.

Cu scopul stabilirii conexiunii purificatorului de aer la internet, aplicația companiei a fost necesară, in acest mod atribui aparatului contul personal Xiaomi, permițând serviciului menționat mai sus sa caute in internet detalii(“scraping the internet”).

Pentru a urma o linie consecventa pe partea de edge-computing(in ceea ce privește plăcuta ESP32 si serverul local), limbajul python este utilizat alături de bibliotecile necesare rulării, pentru ambele device-uri, primul exprimat intre paranteze rulând microPython datorată dimensiunilor reduse.

Tot pe partea de edge-computing, privind comunicarea intre cele doua dispozitive auxiliare, a fost ales un broker de mesaje MQTT reprezentând un acces rapid, ușor de integrat prin serviciul Docker.

Trebuie menționat faptul ca prin fișierul de configurare al broker-ului se specifica permisiunea conexiunilor anonime, neexistând vreo necesitate de autentificare.

Aceasta abordare a fost aleasa din doua motive: comunicarea se realizează in rețeaua locală, așadar ambele programe sunt “de-ale casei”; datorită firewall-urilor existente pe care le pot atribui ruter-ului de acasă cât si server-ului, am posibilitatea de a crea niște reguli de rutare, astfel încât dispozitivele ce doresc conectarea pe server la portul configurat si nu îndeplinesc condiția minima si necesară de a avea adresa IP a plăcuței atribuită de router la conectarea la internet vor avea orice pachet **dropped**.

Urmatoarele rânduri fac referire la întrebările: “de ce am nevoie?”, “Care date au nevoie de extra-safety?”, in legătură cu stocarea datelor in Cloud. Pornind de la prima întrebare, ajungem la concluzia că-mi trebuie un serviciu Cloud stabil, fără date eronate, consacrat de mulți ani.

Pentru comenzi e necesar sa avem un server MQTT, cu avantajul fata de HTTP ca acestea ajung in scurt timp la ESP32 fără ca plăcuta sa fie nevoită sa interogheze mereu serverul de Cloud.

Pe de alta parte, la pornirea aplicației si verificarea datelor de la purificator, un serviciu de genul MQTT ar fi power-consuming, fiind abonat mereu la broker, astfel, preferam sa avem o metoda prin care ne sunt afișate rezultatele de la o interogare HTTP, reîmprospătate la o perioada de 5 secunde.

Inspectand a doua întrebare din paragraful de mai sus, putem concluzia ca daca datele ce reprezintă informații despre purificator nu sunt atât de relevante atacului cibernetic, astfel, ca un simplu API key inclus in URL își face treaba foarte bine.

Totodata, nu dorim sub nicio forma si am fi foarte deranjați daca un atacator ar obține acces la a trimite comenzi, așadar, avem nevoie de criptarea datelor, dar si de o metoda de autorizare a accesului la broker.

In concluzie, in ceea ce privesc soluțiile Cloud, avem doua opțiuni viabile ce se mulează perfect pe cerințele descrise. In ceea ce privește partea obținerii de date prin HTTP, vom crea un cont pe platforma ThingSpeak si vom crea o noua tabela, cu 8 field-uri, fiecare field aparținând de o proprietate a aparatului.

A doua soluție este reprezentată de platforma HiveMQ, cu ajutorul căreia datele sunt criptate, conexiunea este de încredere, iar conectarea la broker necesita autentificare.

O varianta inițială a acestui proiect a constatat într-o aplicație web, fiind facilă scrierea front-end-ului, cu toate acestea, rezumând nevoile actuale ale generației predominante, nimeni nu ar vrea sa mai deschidă browser-ul pentru a-și controla aparatul, ci majoritatea vor o aplicație de sine stătătoare cu care să-și facă treaba.

Cu toate acestea, se dorește a se găsi o punte comuna astfel încât să fie mulțumită și minoritatea care preferă, dintr-un motiv sau altul, să intre pe un browser. Așadar, aplicației i s-ar preta 2 variante: cod scris atât pentru o aplicație(iOS, Android), dar și pentru browser web sau un singur cod scris pe regula “write once, run everywhere”.

Cum proiectul nu are drept focus central performanța, având în vedere că nu e o aplicație I/O intensivă, dar nici CPU intensivă, dar are totuși o perioadă de producție care constrânge și necesită viteza de implementare, a fost aleasă a doua opțiune.

În cele din urmă, proiectul s-a confruntat cu o problemă de alegere, întrucât cu cât mai multe opțiuni există, cu atât mai greu alegi una. Din multitudinea de framework-uri disponibile care presupun scrierea unui singur cod pentru toate platformele(React Native, Flutter, Ionic, KMM, Xamarin), Flutter a reprezentat cea mai atractivă soluție, fiind foarte cunoscut și existând mult suport pentru el, atât din site-uri precum Stack Overflow dar și documentația aferentă.

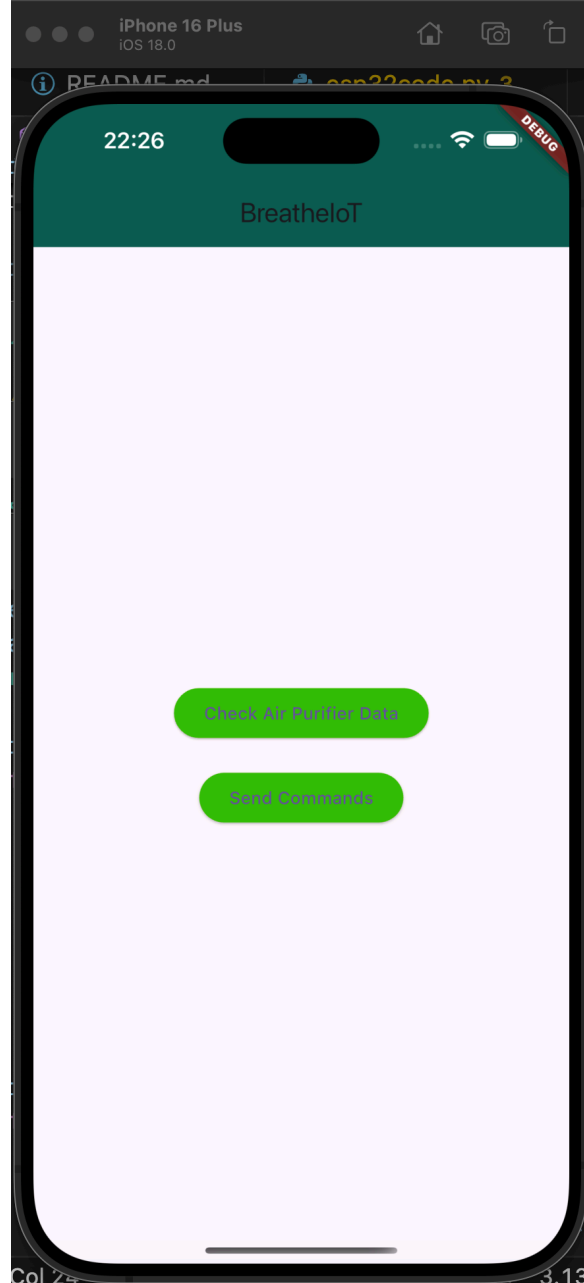
VII Prezentarea aplicației

Datorită stilului actual de reprezentare a informațiilor prin informații dedicate și totodată din cauza focusului proiectului ce vizează funcționalitatea(în egală măsură și simplitatea în utilizare); aplicația este una simplistă, prezentând toate nevoile utilizatorului într-un mod cât mai simplu de înțeles.

Codul a fost rulat pe un dispozitiv emulat(iPhone 16 plus).

VII.I Ecranul principal(Home)

Se observă în figura atașată mai jos prezenta a două butoane, în mijlocul ecranului pentru a evidenția existența acestora, user-ul nemaifiind nevoit să se confrunte cu o multitudine de setări, configurații și opțiuni, doar ce îl interesează.

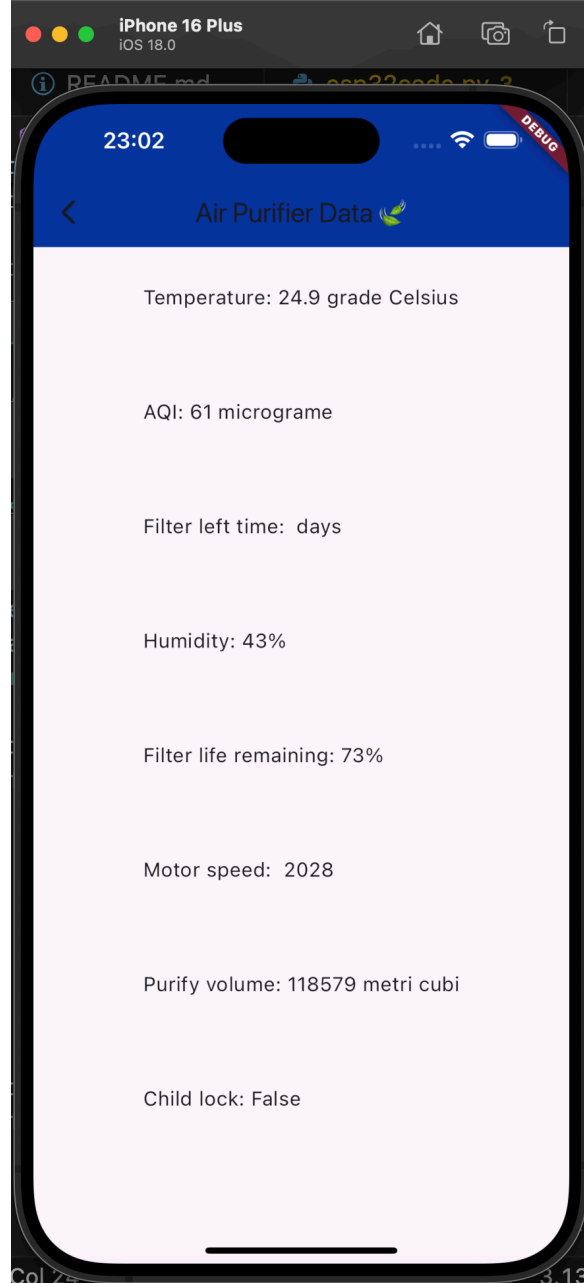


Primul buton exprima scurt funcționalitatea sa, este un buton ce de îndată ce este apăsat, arată ultimele date încărcate în cloud privind informații importante atât despre purificator în sine, cât și despre aerul prezent în locuința propriei utilizatorului (presupunându-se că aparatul este amplasat în propria sa locuință).

La o scurtă privire aruncată celui de-al doilea buton prezent în meniul Home, ne putem da seama de funcționalitatea acestui buton, și anume că ne duce către o pagină dedicată trimerii comenzilor.

VII.II Ecranul pentru vizualizarea datelor

Urmand logica prezentată mai sus, accentul este pus pe simplitatea cu care utilizatorul interacționează cu această aplicație.

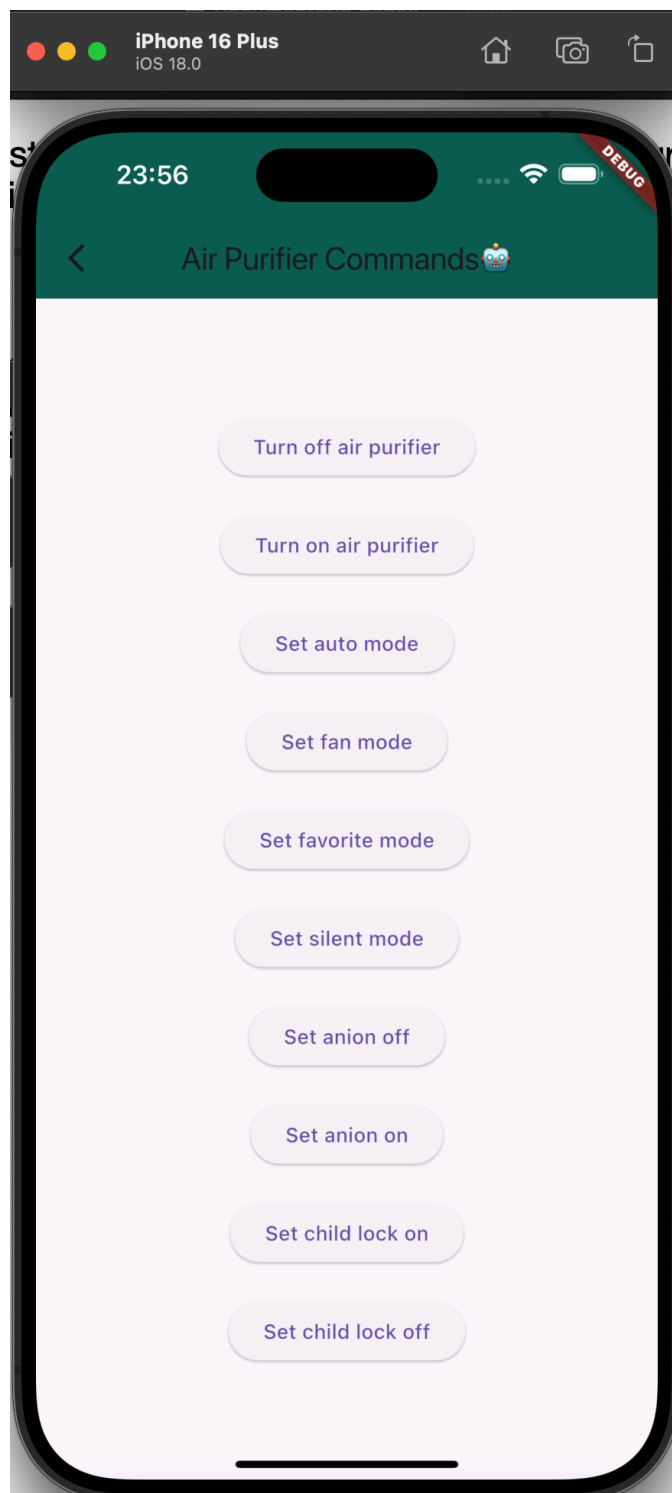


Putem observa ca aplicația prezintă caracteristicile a 8 field-uri de interes, pe rând, fiind: temperatura încăperii in care se afla purificatorul(exprimată in grade Celsius), AQI(air quality index, indicator de tipul “Lower is better”, ce indica câte micrograme de particule de tipul PM10, PM2.5 sunt in aerul inspirat pe o unitate de volum), Filter left time(field ce are ca valoare un număr de zile ce ajuta utilizatorul sa știe când anume trebuie schimbat filtrul in folosință), Humidity(componenta procentuala ce exprima umiditatea aerului din încăpere), Filter life remaining(exprimat in procente, arată un nivel de sănătate al filtrului, indicat fiind ca pe la 20% sa fie înlocuit, Motor speed(număr ce indica turația aparatului), Purify volume(numărul de metri cubi purificați), Child lock(o variabila ce spune daca blocajul pentru copii este activat.

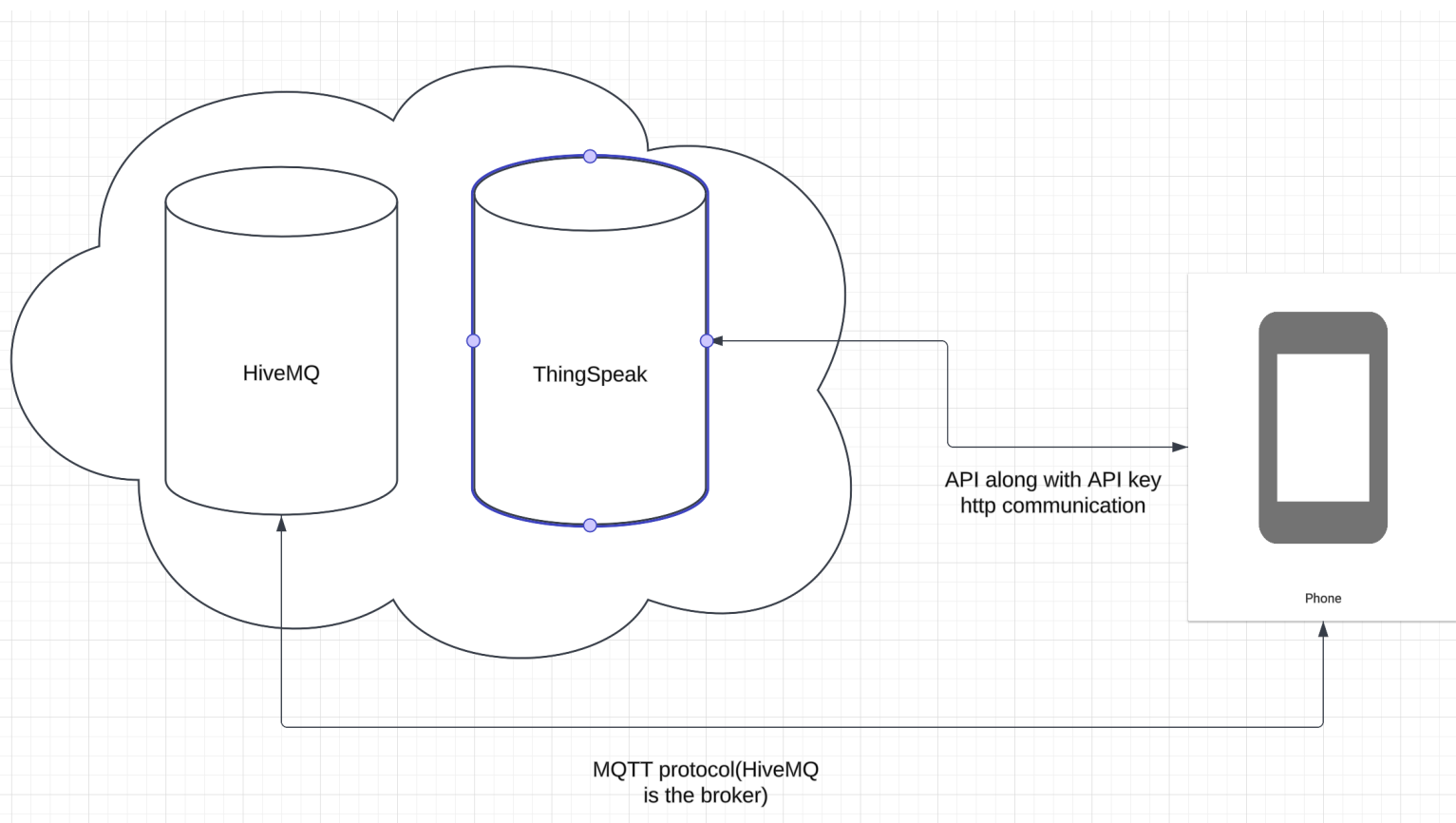
S-a ales pentru aceasta pagina o tema de albastru sus, alături de un emoji specific temei proiectului in cauza.

VII.III Ecranul pentru trimiterea comenzilor

Ecranul menționat are ca tema o nuanță de verde închis alături de un text prin care se indica faptul ca următoarele butoane, odată apăsate, creează comenzi ce vor fi trimise aparatului pentru acționare, text îmbogățit cu un emoji 🤖, care indica inovația prin tehnologie.



VIII Vizualizarea in mod grafic a protocoalelor de comunicație folosite



VIII.I Conexiunea dintre aplicația mobilă și serviciile de stocare cloud folosite

Se poate observa din figura atașată mai sus ca telefonul nu comunica în mod direct cu purificatorul de aer, ci doar cu bazele de date cloud, susținute de către HiveMQ și ThingSpeak.

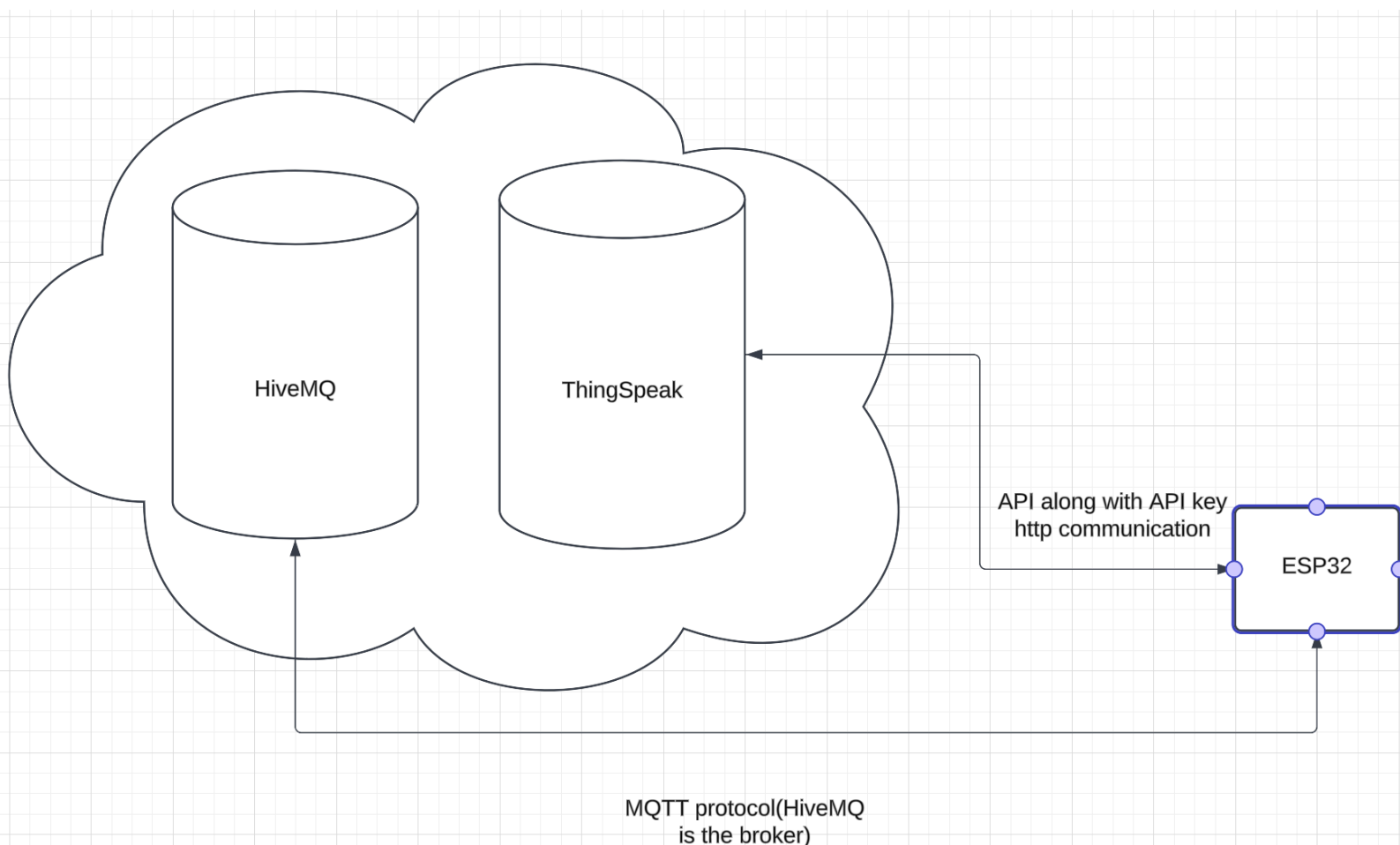
Asa cum a fost expus și într-un paragraf anterior, telefonul folosește două protocoale de comunicație(HTTP și MQTT) atât din considerente de securitate cât și datorită faptului că cererile HTTP sunt mult mai puțin solicitante când datele sunt trimise către broker-ul MQTT la interval de câteva secunde.

Cu toate acestea, și soluția privind ThingSpeak realizează o oarecare măsură de securitate prin introducerea unui API key, fără de care nu se pot trimite date către server.

VIII.II Conexiunea dintre plăcuta ESP32 si serviciile de stocare cloud utilizate

Fiind un proiect IoT, indiferent de faptul ca purificatorul de aer are deja integrat in interiorul acestuia un ESP32(microcontroller foarte performant si popular, cu un puternic impact asupra proiectelor IoT din întreaga lume), fiind “creierul” aparatului, având in vedere ca acesta controlează întreg device-ul, de la comunicația cu rețeaua Wi-Fi, pana la pornirea ventilatoarelor la o anumita treapta de putere, a fost nevoie de încă un astfel de microcontroller care sa comunice cu serviciile cloud folosite.

Microcontroller-ul folosit este de tip SILABS CP2102 și reprezintă o punte de comunicare intre cloud si serviciul docker pus in funcțiune in rețeaua locală ce reprezintă un broker MQTT.

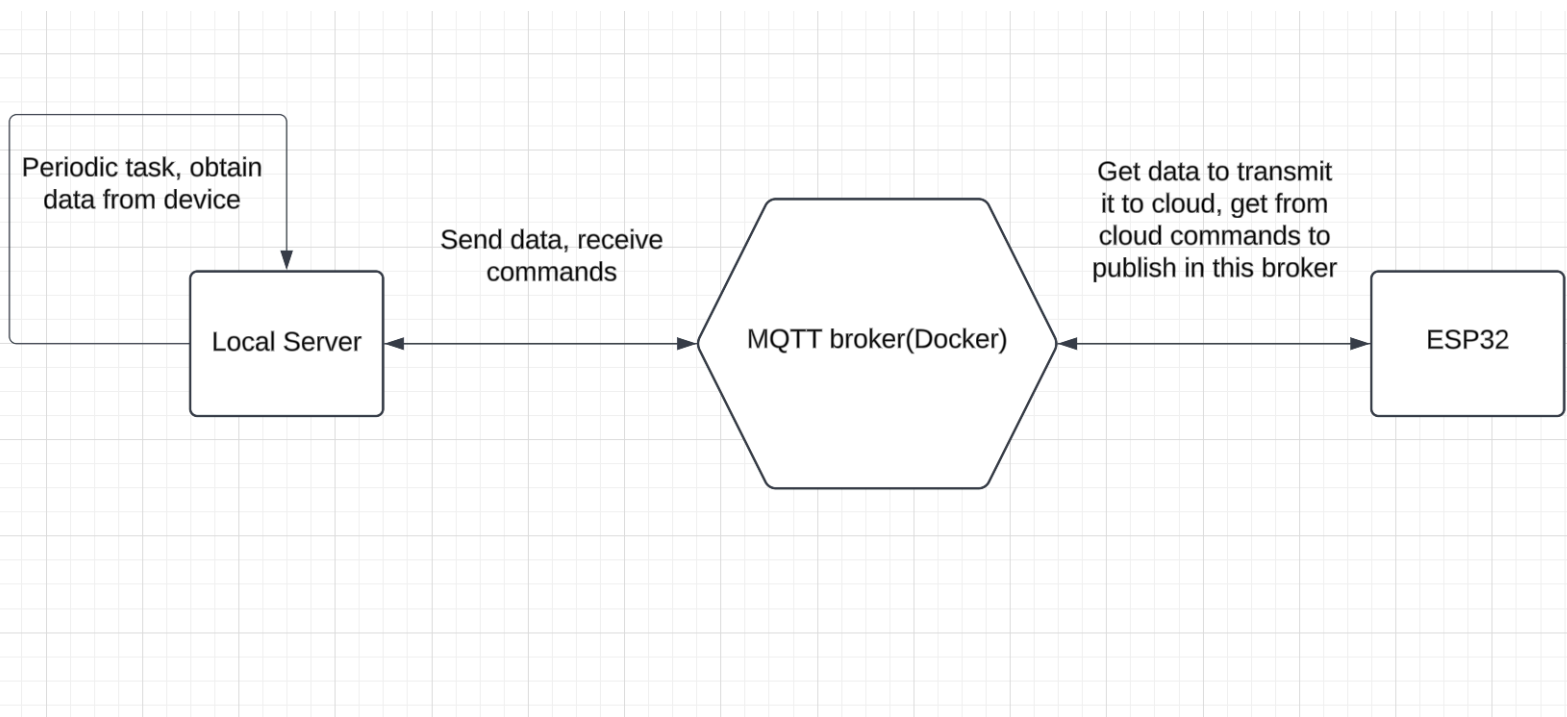


Asa cum se observa și-n figura de mai sus, diagrama respectivă cu cea in care telefonul comunica cu mediul cloud se aseamănă, totuși, rolurile se inversează, in sensul in care ESP32 primește date de la broker si uploadeaza către ThingSpeak.

VIII.II Conexiunea dintre plăcută ESP32, broker-ul MQTT din rețeaua locală si server-ul local

Cunoscand limitările unui ESP32, un server local cu o capacitate mult mai mare este fezabil pentru trimiterea comenzilor către purificator.

In figura de mai jos se poate observa cum, după ce serviciul docker este pornit, datorită faptului ca se afla in rețeaua locală, este accesibil ușor de către plăcuta si de către server-ul local.

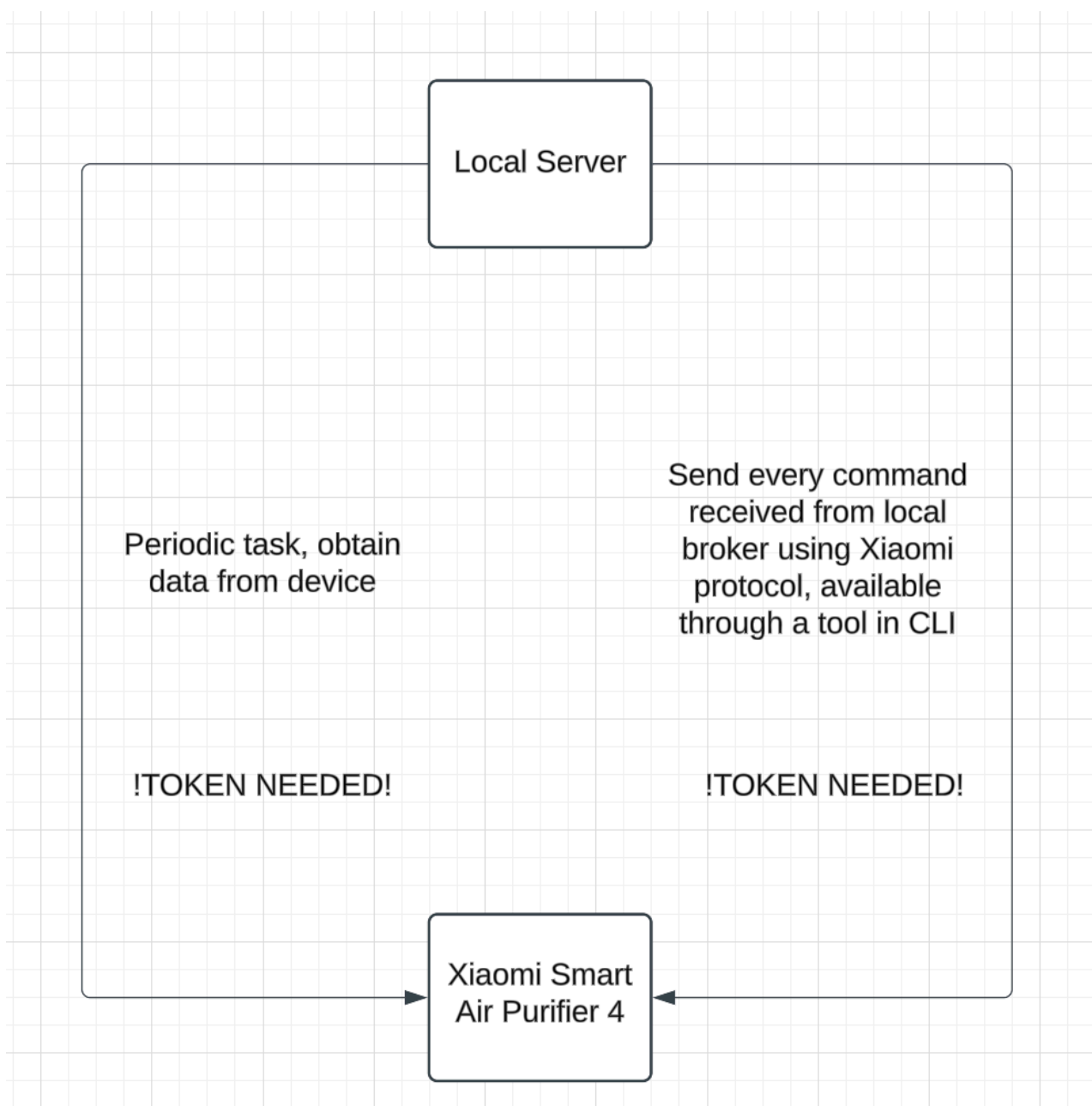


VIII.IV Conexiunea dintre server-ul local si purificatorul de aer

Datorita faptului ca nu este in interesul acestui subcapitol reprezentarea la un nivel mai apropiat de hardware, vom considera aparatul un black box ce funcționează in mod magic. Tot pentru a

înțelege strict comunicația, consideram ca device-ul este mereu conectat in priza si are o adresa IP mereu valida, iar token-ul nu se schimba chiar daca vine alt user să-l folosească cu un alt telefon, cont Xiaomi si alt server local.

Asa cum se observa in figura de mai jos, protocolul specific celor de la Xiaomi este folosit pentru comunicarea eficienta si corecta cu aparatul. Token-ul este necesar din punctul de vedere al autorizării accesului la date cât si la comenzi ce urmează a fi procesate.



IX *Feedback*

In ciuda dificultății ridicate a acestui proiect, dat fiind faptul ca am fost nevoiți sa ne alegem un proiect înainte de a apuca să-l implementam, neștiind de ce tehnologii o sa avem nevoie, cred ca in mod categoric pune in valoare instinctul ingineresc de “a încropi ceva”.

Consider ca acest proiect ar fi fost mult mai plăcut de a lucra la acesta daca nu ar fi o limita inferioara a numărului de pagini a documentației, nereprezentand exact o metoda extraordinara in a puncta cât de buna este o tema.