



**UNIVERSITATEA DE VEST DIN TIMIȘOARA
FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ
PROGRAMUL DE STUDII DE LICENȚĂ: Informatică**

LUCRARE DE LICENȚĂ

COORDONATOR:

Lect. Dr. Liviu Octavian Mafteiu-Scai

ABSOLVENT:

Bobosila Victor

TIMIȘOARA

2020

UNIVERSITATEA DE VEST DIN TIMIȘOARA
FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ
PROGRAMUL DE STUDII DE LICENȚĂ : Informatică

SISTEM INTELIGENT PENTRU CASA : HOME AI

COORDONATOR:

Lect. Dr. Liviu Octavian Mafteiu-Scai

ABSOLVENT:

Bobosila Victor

TIMIȘOARA

2020

Abstract

To start off things, we must discuss the need for an accessible smart home system. All the technological advancements were aimed at making our daily life easier, such as the phone, which allowed us to better communicate with each other, the internet, which allowed us to find information faster, wi-fi, which allowed us to be connected to the internet at all times, and so much more. And now, to come back to our main topic, an accessible smart home system aims to improve our day by day lives, especially during those hard times, with all the advantages it brings us, such as helping us keep track of the temperature in the house, so we can adjust accordingly, potentially saving us some money. Maybe we forgot a light on somewhere in the house, and without even getting up we can turn it off, making our lives easier. Those are just some examples of what is possible, but the idea is clear, making our lives easier is important.

Cuprins

Introducere	6
1 Dezvoltarea unui sistem inteligent pentru case	7
1.1 Despre device-uri IoT	7
1.2 Utilizarea device-urilor IoT pentru dezvoltarea unei case inteligente . .	8
1.2.1 Automatizarea sistemului de iluminat	8
1.2.2 Automatizarea sistemului de încălzire și controlul consumului de apă	8
1.2.3 Automatizarea electrocasnicelor	8
1.3 Avantajele unei case inteligente	9
1.4 Utilizarea unui hub pentru controlarea device-urilor IoT	10
1.4.1 Despre Hub-uri	10
1.4.2 Exemple de Hub-uri	10
1.4.3 Folosirea unui raspberry pi pe post de hub	10
2 Clasificarea sistemelor inteligente pentru case	11
2.1 Power Line Home Automation System	11
2.2 Wired Home Automation System	11
2.3 Wireless Home Automation	11
3 Exemple de sisteme inteligente pentru case	12
3.1 Amazon Echo	12
3.2 Wink Hub 2	12
3.3 Samsung SmartThings	13
3.4 Google Nest Hub	13
3.5 Apple HomeKit	13
4 Contributii proprii	15
5 Descriere aplicatie	16
5.1 Arhitectura aplicatiei	16
5.2 Interpretarea arhitecturii	17
5.2.1 MQTT Broker	17
5.2.2 Django server	17
5.3 Hardware	17
5.4 Tehnologii Folosite	18
5.4.1 Conectarea la dispozitive IoT	18
5.4.2 Trimiterea si citirea mesajelor MQTT	18
5.4.3 Interfata utilizatorului	18
5.5 Implementarea aplicatiei	18

5.5.1	Configurarea server-ului MQTT	18
5.5.2	Functionalitate	19
6	Concluzii	35
	Bibliography	36

Introducere

Domotica este o aplicație a calculatoarelor și roboților pentru aplicații casnice. Este un cuvânt compus din *domus* (latină, însemnând casă) și informatică. Domotica țintește să integreze și să-și răspândească aplicațiile pretutindeni.

O casă cu un sistem domotic instalat trebuie să aibă multe calculatoare, probabil aranjate într-un perete, pentru a permite proprietarului să controleze aplicațiile din toate părțile casei.

O casă cu un astfel de sistem instalat este capabilă să sune la poliție sau la pompieri, cu multă ingeniozitate și cu o largă varietate a plății, față de sistemele cu alarmă normală. În mod frecvent sistemele domotice trebuie să colecteze date de la diverși senzori și să facă diferite lucruri cum ar fi: ajustarea luminii și selectarea melodiilor preferate de fiecare membru al casei, atunci când ei intră sau pleacă din camera personală. Acest sistem presupune ca fiecare membru să poarte asupra lui un tag RFID, pe câtă vreme cele mai sofisticate detectează mișcarea, funcțiile vitale și multe alte caracteristici individuale.

Câteva sarcini îndeplinite de domotică :

- Controlează draperiile, ferestrele dintr-o locație, toata ziua, fără interacțiunea omului.
- Deschide sau blochează și deblochează poarta și intrarea în garaj, cu un control separat sau global.
- Controlează clima din interiorul caselor. Prin apăsarea unui buton se poate seta încălzirea pe timp de noapte; lumina când nu ești într-o încăpere; să închidă poarta după plecare...
- Asigură lumina potrivită la locul potrivit; sistemele domotice pot asigura și memora intensitatea luminoasă în funcție de preferințele persoanelor.
- Pot pregăti inteligent - grădinile prin pornirea stropitoarelor atunci când solul este prea uscat; - și alte lucrări cum ar fi cele canal; peluzele sunt udate doar atunci când este nevoie iar persoanele pot starbate peluzele fără frica că pot fi udate
- Pot aprinde lumina doar atunci când o persoană este prin preajmă(uneori cu rol de alarmă).

Un dezavantaj al caselor ce au sistem domotic instalat este că sunt scumpe. Unele sisteme pot fi extrem de scumpe atunci când sunt instalate. Există însă și posibilitatea ca sistemele să cedeze și să lase casele fără încălzire și fără o iluminare corespunzătoare. Aceste inconveniente pot fi îndepărtate prin folosirea unor tehnici și tehnologii de siguranță.

Capitolul 1

Dezvoltarea unui sistem inteligent pentru case

1.1 Despre device-uri IoT

Internet of Things (IoT), tradus în română ca Internetul lucrurilor, se referă la miliardele de device-uri existente în întreaga lume, care sunt conectate la internet, colectând și schimbând date. De la periute de dinți la utilaje, dispozitivele comerciale și industriale sunt dotate cu chip-uri prin intermediul cărora colectează și comunică diferite informații.

Din punct de vedere comercial, multe dintre aceste obiecte vizează îmbunătățirea a ceea ce e cunoscut ca Quality of Life (QoL), în traducere calitate a vieții, ușurând responsabilitățile zilnice ale persoanelor. Cu toate acestea, printre ele se strecoară și diferite device-uri mai puțin folosite cum ar fi: Egg Minder (o tavă care îți spune câte ouă mai ai și cât de proaspete sunt acestea) sau Shuttereaze (un dispozitiv care trage perdele sau jaluzelele automat).

În lumea Internetului Lucrurilor, toate obiectele care sunt conectate la internet pot fi grupate în trei mari categorii:

- obiecte/device-uri care colectează informație și apoi o transmit mai departe - fermierii primesc informații despre umiditatea din sol și udă plantele în consecință;
- obiecte/device-uri care primesc informație și acționează în consecință - obiectele execută ceea ce tu le ceri, prin comenzi de la distanță;
- obiecte/device-uri care fac ambele operațiuni - nu mai ești tu nevoit să stingi sau să aprinzi lumina în funcție de informațiile pe care le primești din mediul în care trăiești, ci un sistem face acest lucru în locul tău (sistemul de iluminat inteligent, de exemplu).

1.2 Utilizarea device-urilor IoT pentru dezvoltarea unei case inteligente

1.2.1 Automatizarea sistemului de iluminat

Eficiența unui sistem de iluminat constă și într-un consum calculat de energie electrică. Becurile LED, de pildă, pot fi conectate la senzori de lumină și de poziție și totodată pot fi controlate printr-un sistem inteligent. Și este vorba nu numai de interiorul casei tale, ci și de terasă sau grădină, dacă deții un astfel de spațiu. Astfel, în funcție de cantitatea de lumină naturală, ele se vor închide sau se vor aprinde automat. Dacă vrei să lumineze mai puternic în anumite intervale orare, le poți, de asemenea, programa să-și regleze intensitatea.

Un sistem de control inteligent, care permite comunicarea între LED-uri și storuri, asigură menținerea unui nivel optim de căldură și lumină. Mecanismul detectează cantitatea de lumină naturală dintr-o încăpere și, în funcție de informația primită, reglează luminozitatea instalațiilor de iluminat și ridică sau coboară jaluzelele.

Apoi, astfel de sisteme inteligente de iluminat pot fi programate să creeze o anumită atmosferă, schimbând culorile luminilor ca să obții settingul dorit pentru o seară de vizionat filme sau pentru o cină în grădină, alături de cei dragi. Totul cu ajutorul tehnologiei Wi-Fi, pe care o poți folosi inclusiv pentru a-ți încărca smartphone-ul.

1.2.2 Automatizarea sistemului de încălzire și controlul consumului de apă

Cu ajutorul sistemelor automatizate, al aplicațiilor moderne și al unui termostat inteligent, poți programa centrala termică să funcționeze doar în anumite intervale orare. În acest mod, poți porni căldura de pe telefon, cu jumătate de oră înainte de a ajunge acasă, ca să îți asiguri confortul când intri în locuință și să ai garanția, în același timp, că nu ai consumat energie inutil.

Prin aplicațiile inteligente, poți avea acces și la grafice de consum atât în ceea ce privește energia, dar și legate de consumul de apă caldă menajeră, ca să vezi cum îți poți optimiza sistemul, astfel încât să faci economie și, implicit, facturile la utilități să fie mai mici.

1.2.3 Automatizarea electrocasnicelor

Un frigider care, practic, vorbește cu tine, anunțându-te ce alimente mai ai în el, când expiră și care te avertizează că este momentul să mergi la cumpărături, dar și un sistem de iluminat care este interconectat cu televizorul inteligent, astfel încât imaginea acestuia se ajustează în funcție de lumina din casă pot fi de un real ajutor. O simplă telecomandă smart te poate ajuta, prin intermediul telefonului, să controlezi televizorul, combina muzicală, home cinema-ul, sistemul de iluminat, aerul condiționat și alte device-uri din casă.

Există senzori care pot fi cei mai buni prieteni ai tăi într-o dimineață în care ești pe fugă. Ei pot porni discret radioul sau televizorul, să deschidă storurile sau să pornească expressorul. Prizele inteligente te scutesc de grija că ai uitat ceva deschis prin casă înainte de a pleca la serviciu sau în vacanță.

1.3 Avantajele unei case inteligente

O casă inteligentă, definită prin Internet of Things sau Internetul Lucrurilor, îți oferă o serie de avantaje pe care este bine să le iei în calcul înainte să îți spui că investițiile în gadgeturile de ultimă generație sunt destul de consistente. Gândește-te că este un proiect cu bătaie lungă, ale cărui efecte se văd în timp. Prin urmare, amortizarea investiției inițiale nu vine imediat, dar este reală. Iată câteva dintre motivele pentru care ți-ai putea transforma oricând locuința în smart home:

- prelungești durata de viață a sistemelor folosite - durata de viață a unui sistem de încălzire prin pardoseală, a unei centrale termice ori a conductelor care trec prin casa ta depinde foarte mult de modul în care folosești dispozitivele respective; un sistem inteligent de control, folosirea senzorilor, interconectarea device-urilor din casă îți permit utilizarea mai eficientă și corectă a unor astfel de sisteme; ele nu vor funcționa inutil, nu se vor roda fără sens;
- crești eficiența energetică a casei - performanța energetică a unei case este determinată de energia consumată sau estimată pentru a răspunde necesităților cum ar fi încălzirea, prepararea apei calde de consum, răcirea, ventilarea și iluminatul; Internet of Things îți permite un consum eficient de resurse, accesul la grafice, ca să poți face analize reale și estimări realiste; acest lucru se reflectă, desigur, în cheltuieli mai mici cu utilitățile;
- îți asiguri confortul și un nivel în plus de securitate - prin programarea device-urilor din locuință, ești scutit de eforturi suplimentare, economisești timp, nu mai ai grija de a porni sau de a închide aparatura din casă, ai acces mai rapid și mai ușor la gadgeturile tale; te asiguri că ești în siguranță prin folosirea sistemelor de supraveghere video și de închidere inteligentă a locuinței; apoi, există aplicații prin care poți fi informat inclusiv dacă apar avarii la instalațiile din casă (sanitare, de gaz, electrice);
- alegi tipul de smart home pe care îl dorești - ai o varietate de posibilități să-ți transformi locuința în smart home, în funcție de nevoile tale și ale celor dragi; poți opta pentru obiecte smart (electrocasnice, sisteme home cinema), sisteme inteligente (aparate care comunică între ele), sisteme de control conectate care pot fi accesate, prin rețea, de la distanță, tehnologii moderne care ajung să-ți anticipeze nevoile (să aprindă lumina la baie, la 6:30, atunci când te trezești, să monitorizeze frecvența cu care folosești anumite obiecte din casă).

1.4 Utilizarea unui hub pentru controlarea device-urilor IoT

1.4.1 Despre Hub-uri

Un hub este folosit pentru a conecta device-uri IoT și a controla comunicarea între ele.

Uneori denumit un pod de casă inteligentă, un hub colectează și traduce diverse comunicații de protocol de pe dispozitive de casă inteligentă. De exemplu, dacă un smartphone, care nu folosește Zigbee pentru a comunica, vrea să vorbească cu un dispozitiv de blocare inteligent, care folosește doar Zigbee și nu Wi-Fi sau Bluetooth nativ pentru smartphone, hub-ul acționează ca un traducător între cei doi.

Huburile pentru casă inteligentă sunt capabile să controleze multe dispozitive și sisteme inteligente pentru casă și IoT, inclusiv senzori inteligenți pe termostate, becuri, prize și comutatoare, încuietori și senzori de ușă, sonerii, deschizători de uși de garaj, monitoare de energie și tratamente / acoperiri pentru ferestre și senzori. De asemenea, pot controla senzori de mișcare, senzori de inundații și scurgeri, radio și difuzoare inteligente, sisteme și camere de securitate, detectoare de fum și monoxid de carbon, controlere de irigare și ventilatoare, încălzitoare de apă și alte aparate de uz casnic.

1.4.2 Exemple de Hub-uri

Câteva din cele mai populare hub-uri pentru o casă inteligentă sunt :

- Samsung SmartThings Hub
- Wink Hub
- Insteon Hub
- Logitech Harmony Hub
- Philips Hue Bridge
- Securifi Almond 3
- Amazon Echo Plus
- Google Nest Hub

1.4.3 Folosirea unui raspberry pi pe post de hub

În cazul în care un utilizator nu dorește să își cumpere un hub de la diferite companii, poate alege să își creeze unul. Cu ajutorul unui raspberry pi, acesta se poate programa pentru a comunica cu device-uri IoT și senzori IoT, sau se pot folosi diferite programe deja create precum WebThings Gateway de la mozilla

Capitolul 2

Clasificarea sistemelor inteligente pentru case

2.1 Power Line Home Automation System

Această automatizare este ieftină și nu necesită cabluri suplimentare pentru a transfera informațiile, ci folosește liniile de alimentare existente pentru a transfera datele. Cu toate acestea, acest sistem implică o mare complexitate și necesită circuite și dispozitive de conversie suplimentare.

2.2 Wired Home Automation System

În acest tip de automatizare, toate echipamentele de acasă sunt conectate la controlerul principal (controler logic programabil) printr-un cablu de comunicație. Echipamentul este atașat cu actuator pentru a comunica cu controlerul principal. Întreaga operațiune este centralizată de computerul care comunică continuu cu controlerul principal

2.3 Wireless Home Automation

Aceasta este extinderea și avansarea automatizării prin cablu care utilizează tehnologii fără fir precum IR, Zigbee, Wi-Fi, GSM, Bluetooth etc., pentru realizarea unei operări la distanță. De exemplu, automatizarea casnică bazată pe GSM asigură controlul echipamentelor casnice printr-un SMS către modemul GSM.

Capitolul 3

Exemple de sisteme inteligente pentru case

3.1 Amazon Echo

Echo se conectează la Internet prin rețeaua WiFi de acasă. Este întotdeauna activat și ascultând cuvântul magic pentru a-l trezi. Odată ce aude asta, dispozitivul adună comenzile vocale care urmează și le trimite către un serviciu natural de recunoaștere a vocii în cloud numit Alexa Voice Service, care le interpretează și trimite înapoi răspunsul corespunzător. Dispozitivul are o serie de microfoane care vă pot prelua vocea din întreaga cameră, chiar și peste muzică și alte zgomote de mediu.

Amazon adaugă tot mai multe servicii la Echo și a făcut ca serviciul cloud Alexa să fie disponibil pentru utilizare de către dezvoltatori terți, deschizându-l la o mulțime de posibilități viitoare.

Echo poate fi folosit ca parte a casei inteligente pentru a face lumini, aparate și hub-uri inteligente pentru casă compatibile cu activarea cu voce.

Puteți face ca Echo să execute seturi de acțiuni mai complexe prin configurarea unui canal Alexa pe site-ul terță IFTTT („If This, Then That”), care setează declanșatoare pentru anumite acțiuni. Aceasta înseamnă că puteți utiliza Echo pentru a porni aparatele atunci când alarme pornesc, pentru a controla un termostat Nest, pentru a vă găsi telefonul sau pentru a adăuga lucruri la lista de sarcini Evernote prin comandă vocală.

3.2 Wink Hub 2

Wink Hub 2 este unul dintre cele mai versatile dintre hub-urile recomandate pentru automatizarea casei. Hub-ul în sine poate fi configurat fără fir sau conectat la router cu un cablu Ethernet. După aceea, descărcați aplicația Wink pe dispozitivul dvs. Android sau iOS, apoi controlați totul din interiorul aplicației. De acolo puteți selecta cu ce alte dispozitive inteligente doriți să vă conectați și să le controlați setările în consecință.

Unde excelează Wink Hub este abilitatea de a seta condiții pentru celelalte dispozitive inteligente în mod implicit, făcându-l cu adevărat centrul sistemului dvs. de automatizare a casei. Folosind protocoalele IFTTT - IF, Then, This, That - puteți seta dispozitivele inteligente să activeze și să facă lucruri specifice atunci când sunt îndeplinite anumite condiții. De exemplu, puteți seta luminile inteligente să se aprindă automat la anumite ore sau încălzirea sau aerul condiționat să se aprindă la temperaturi stabilite sau să le spuneți camerelor de securitate să înregistreze când este activat un senzor de ușă. În acest fel, casa ta poate deveni nu doar conectată, ci controlată automat. Singurul dezavantaj al acestui fapt este că aplicația Wink 2 ar putea să nu ofere aceeași reglare fină pe care ar putea să o ofere și aplicațiile dedicate pentru dispozitivele inteligente, cum ar fi comanda unui termostat inteligent care să fie pornit-oprit între datele de vacanță.

3.3 Samsung SmartThings

Hub-ul Samsung SmartThings își propune să aducă sistemul de casă inteligentă împreună cu un singur hub de automatizare a casei, care este operat printr-o aplicație mobilă. Samsung produce, de asemenea, o serie de dispozitive periferice pentru a lucra direct cu acesta, nu în ultimul rând senzori de mișcare, senzori multifuncționali și butoane inteligente. Hub-ul Samsung SmartThings este compatibil cu majoritatea dispozitivelor inteligente terțe, deși nu are atât de multe opțiuni de conectivitate ca unele, fiind limitat în principal la protocoalele Z-Wave și Zigbee. Totuși, ceea ce oferă este personalizări IFTTT simple pentru a face o casă cu adevărat inteligentă.

3.4 Google Nest Hub

Google Nest Hub este un alt sistem des folosit pentru configurarea unui sistem de automatizare a casei. Folosește Google Assistant pentru a efectua sarcini folosind comenzi vocale, dar funcționează în principal ca o interfață vizuală, acționând efectiv ca o tabletă. Poate afișa informații meteo locale, evenimente și notificări, programări zilnice, precum și reda muzică și videoclipuri de pe YouTube și alte servicii. Atunci când nu este utilizat, poate afișa un ceas pe ecran sau poate servi ca o ramă foto digitală afișând imagini din contul dvs. Google Photo.

De asemenea, funcționează, desigur, ca un hub central de automatizare a casei și se conectează cu ușurință cu multe alte dispozitive inteligente. Se poate controla luminile, încălzirea, securitatea și gama tipică a casei inteligente prin comenzi vocale sau interfața cu utilizatorul glisând sau atingând opțiunile ecranului, după cum este necesar.

3.5 Apple HomeKit

HomeKit este un framework software pregătit chiar pe dispozitivele Apple, iar ideea este simplă - în loc să existe o grămadă de aplicații inteligente pentru acasă pe smartphone, care nu se sincronizează neapărat, HomeKit le reunește pe toate, oferind control de pe dispozitivele tale.

De fapt, Apple HomeKit are două părți. HomeKit în sine este un protocol, o tehnologie software de fundal pe care dispozitivele trebuie să o respecte pentru a avea acces la club (de asemenea, este foarte sigur). Apoi, este elementul HomeKit pe care îl veți vedea pe iPhone, iPad sau Mac, aplicația Home

Capitolul 4

Contributii proprii

Deoarece aplicatia este destinata comunitatii open-source, proiectul v-a fi comparat cu proiecte asemanatoare. Cele mai mari proiecte open-source din acest domeniu sunt urmatoarele:

- <https://github.com/nurikk/z2m-frontend>. Acest proiect este cel mai mare si inca este mentinut de comunitate. El serverste ca partea de frontend pentru proiectul Zigbee2MQTT. Aplicatia este scrisa in TypeScript si se foloseste de WebSockets pentru a comunica cu server-ul MQTT. Este usor de folosit si setat, insa datorita limbajului folosit, compilarea codului dureaza mult.
- <https://github.com/yllibed/Zigbee2MqttAssistant>. Aceasta aplicatie scrisa in C# (ASP.NET core 3.1 LTS), este folosita ca si o interfata web pentru a controla device-urile IoT din retea. Aplicatia este grea de configurat in cazul in care utilizatorul nu foloseste Docker sau HASS.IO, iar comunicarea intre device-uri este lenta datorita libreriei paho mqtt pentru C#. Din pacate este posibil ca acest proiect sa fi fost abandonat, intrucat aplicatia nu a mai primit update-uri de aporape un an.

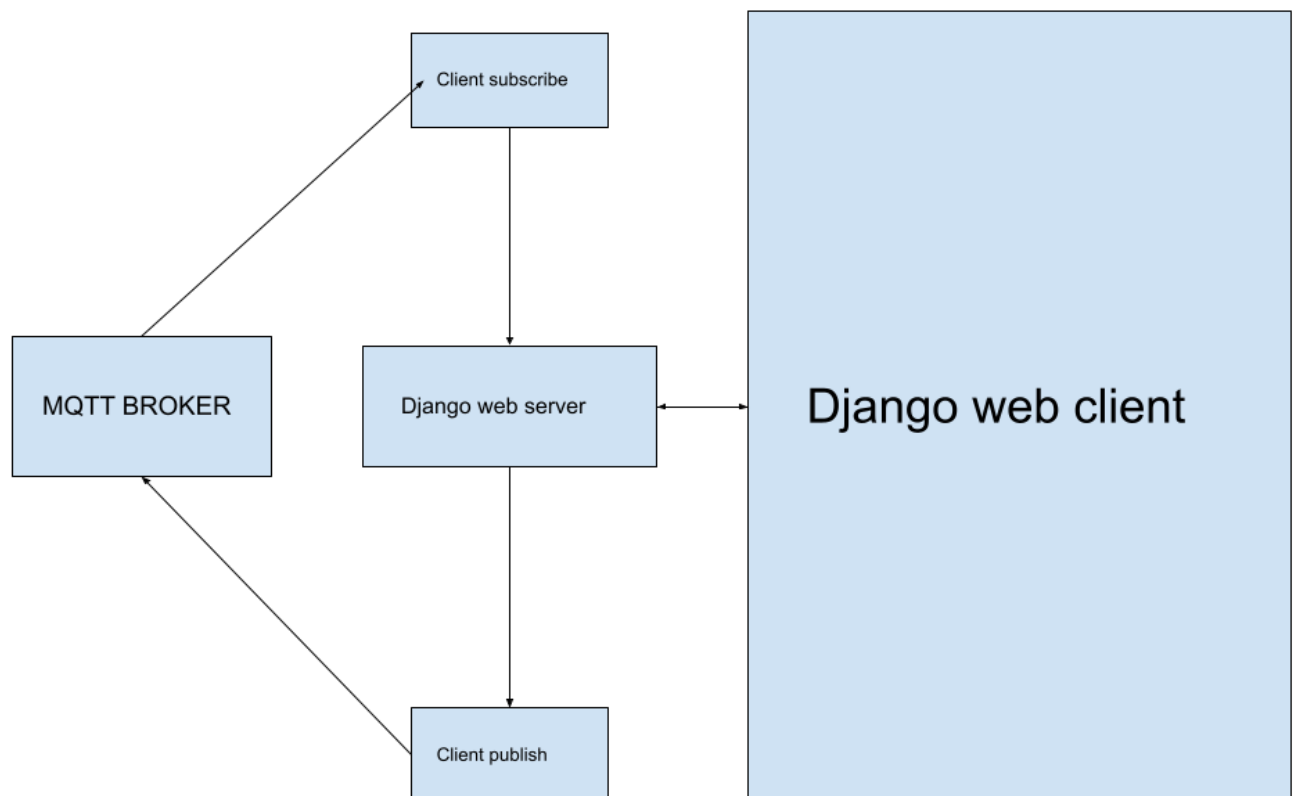
Proiectul discutat in aceasta lucrare de licenta, este unul unic, intrucat aceasta aplicatie este singura ce foloseste Django, fara a mai fi nevoie de alte module. Aplicatia poate fi descarcata si folosita imediat dupa, cu conditia de a avea un server MQTT pornit.

Capitolul 5

Descriere aplicatie

5.1 Arhitectura aplicatiei

Figura 5.1: Arhitectura



5.2 Interpretarea arhitecturii

Arhitectura aplicatiei prezentata in figura de mai sus (Figura 5.1) consta din doua servere, unul ce se ocupa de comunicarea cu device-urile IoT din reseaua locala, si unul ce se ocupa cu preluarea datelor de la server-ul mentionat mai sus, si de modelarea lor pentru a fi vizibile pe aplicatia web.

5.2.1 MQTT Broker

Brokerul de MQTT este server-ul ce se ocupa de comunicarea cu device-urile IoT. Datorita naturii protocolului MQTT(Message Queuing Telemetry Transport), de tip publish-subscribe, acesta este responsabil pentru primirea tuturor mesajelor, filtrarea mesajelor, stabilirea cine este abonat la fiecare mesaj si trimiterea mesajului către clientii abonati.

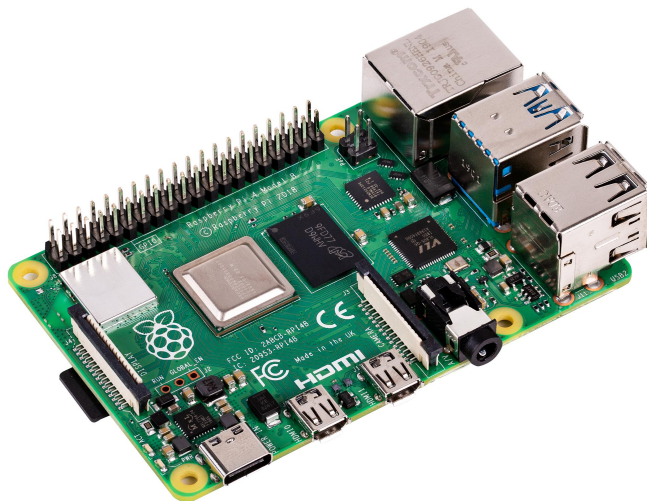
Clientii sunt de doua tipuri. Cei ce se aboneaza la server (subscribe) si asculta mesajele trimise de acesta, si cei ce trimit mesaje catre server (publish). Rolul lor este de a manipula device-urile IoT conectate la server, si de a trimite datele mai departe catre web server-ul Django.

5.2.2 Django server

Rolul web server-ului Django este de a receptiona datele primite de Broker-ul MQTT, cu ajutorul client-ului de subscribe. Cu ajutorul acestor date, server-ul le trimite mai departe catre client (pagina web) unde acestea sunt afisate si pot fi observate de utilizator. Acesta poate cu ajutorul interfetei web, sa trimita instructiuni inapoi spre server, de exemplu sa micsoreze luminozitatea unui bec, care la randul lui trimite instructiuni catre client-ul de publish ce v-a comunica instructiunile primite cu Broker-ul MQTT.

5.3 Hardware

Figura 5.2: RaspberryPi 4 Model B



Datorita naturii aplicatiei, exista cateva componente hardware necesare. Deoarece proiectul tinteste sa controleze device-uri IoT, avem nevoie de aceste device-uri, precum senzori de temperatura, de contant, de miscare, umiditate, becuri ce pot fi controlate, etc. In cazul lumii IoT, majoritatea device-urilor folosesc tehnologia ZigBee, deci pentru a putea avea un server ce comunica cu ele, avem nevoie de o antena ZigBee ce v-a fi atasata la masina folosita pe post de Broker MQTT. Aceasta masina la randul ei, nu are nevoie de multe resurse, deoarece atat serverul de MQTT cat si cel de Django nu consuma multe resurse de sistem. Din acest motiv, proiectul foloseste un RaspberryPi 4 Model B (8gb RAM)(Figura 5.2) pe post de hub, pentru a oferi portabilitate utilizatorului.

5.4 Tehnologii Folosite

5.4.1 Conectarea la dispozitive IoT

Pentru a facilita comunicarea intre device-uri si server, nu este suficient doar protocolul MQTT, intrucat acesta serveste doar ca o simpla poarta. Pentru a putea comunica stari si comenzi catre aceste dispozitive, proiectul utilizeaza Zigbee2MQTT, ce preia mesajele din reseaua ZigBee si le traduce in mesaje structurate pentru protocolul MQTT.

5.4.2 Trimiterea si citirea mesajelor MQTT

Aplicatia doreste sa consume si sa trimita instructiuni catre dispozitive pentru a le controla si automatiza. Astfel, proiectul foloseste libraria paho mqtt pentru python, pentru a crea clienti ce se conecteaza la broker.

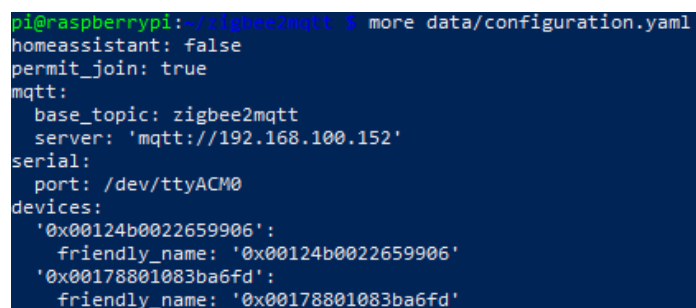
5.4.3 Interfata utilizatorului

Natura proiectului este, la baza, una web. Aplicatia foloseste framework-ul Django, fiind unul ce consuma putine resurse de sistem si in acelasi timp, unul folosit foarte des in dezvoltarea aplicatiilor web, intrucat este usor de folosit si de extins.

5.5 Implementarea aplicatiei

5.5.1 Configurarea server-ului MQTT

Figura 5.3: Zigbee2MQTT configuration.yaml



```
pi@raspberrypi:~/zigbee2mqtt $ more data/configuration.yaml
homeassistant: false
permit_join: true
mqtt:
  base_topic: zigbee2mqtt
  server: 'mqtt://192.168.100.152'
serial:
  port: /dev/ttyACM0
devices:
  '0x00124b0022659906':
    friendly_name: '0x00124b0022659906'
  '0x00178801083ba6fd':
    friendly_name: '0x00178801083ba6fd'
```

In figura 5.3 putem observa configurarea server-ului MQTT. Acesta vine cu majoritatea parametrilor gata configurati insa trebuie modificata adresa server-ului. In acest caz, proiectul foloseste adresa IP a RaspberryPi-ului, intrucat pe aceasta masina se porneste serviciul.

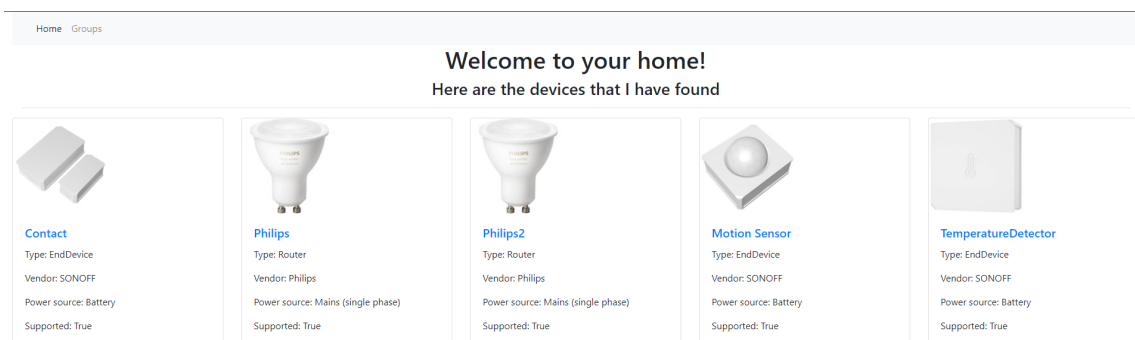
Restul configuratiei specifica urmatoarele lucruri:

- `homeassistant`: `false`. Acesta este un parametru optional, ce permite integrarea proiectului Home Assistant. Este setat cu optiunea `false` deoarece proiectul tintește sa construiască propriul sau Home Assistant
- `permit join`: `true`. Aceasta setare are rolul de a permite device-urilor noi sa se conecteze la retea
- `base topic`: `zigbee2mqtt`. Acesta este un sir de caractere folosit de broker pentru a filtra mesajele pe care le primeste si le trimite mai departe.
- `server`: `'mqtt://192.168.100.152'`. Aici observam adresa IP a server-ului
- `port`: `/dev/ttyACM0`. Locatia adaptorului (antena) Zigbee
- `devices` : ... Lista cu device-urile conectate la retea. In momentul de fata sunt doar 2 device-uri conectate. Se observa numele acestora, reprezentate printr-un numar hexadecimal

5.5.2 Functionalitate

Lista device-urilor

Figura 5.4: FrontPage



La deschiderea paginii web, sistemul realizeaza o cautare in retea locala, cu scopul de a identifica si afisa toate device-urile IoT prezente (Figura 5.4). Utilizatorului ii se ofera lista cu device-urile gasite in urma cautarii, cu anumite informatii generale despre device, precum o poza, tipul, compania ce a creat device-ul, daca este suportat

de Zigbee2MQTT, ce fel de energie consuma (daca primeste energie dintr-o baterie sau trebuie bagat in priza, etc).

Pentru a realiza acest lucru, ne folosim de partea de client a librăriei paho mqtt pentru python.

Figura 5.5: Device Finder Class

```
class DeviceFinder(object):
    def __init__(self):
        try:
            self.data = {}
            self.BROKER_IP = "192.168.100.152"
            self.client = mqtt.Client("P1")
            self.client.on_connect = self.on_connect_finder
            self.client.on_message = self.on_message_finder
            self.client.connect(self.BROKER_IP, 1883, 60)
            self.client.loop_forever()

        except Exception as e:
            print(e)
```

In figura 5.5 observam parametrii clasei ce initializeaza cautarea device-urilor. Primul lucru pe care il face sistemul, este sa se conecteze la server-ul MQTT. Pentru a realiza asta, aplicatia creeaza un client denumit "P1", si apeleaza functia "connect" oferita de librărie, cu adresa IP a server-ului, portul, si timpul maxim de liniste intre client si broker (60 de secunde, in cazul in care nu se realizeaza comunicatii intre client si broker, se deconecteaza clientul).

Datorita naturii asincrone a librăriei, sistemul foloseste o serie de callback-uri pentru a detecta daca conexiunea s-a realizat cu succes, sau daca a primit un mesaj de la broker.

La final, sistemul porneste o bucla infinita, pentru a ocupa thread-ul principal al server-ului web si pentru a avea timp sa gasesca toate device-urile inainte sa incarca pagina principala.

Figura 5.6: Device Finder on_connect callback

```
def on_connect_finder(self, client, userdata, flags, rc):
    self.client.subscribe("zigbee2mqtt/bridge/devices")
    print("connected! finder")
```

Pentru a putea detecta device-urile din rețeaua locala, acesta trebuie in primul rand

sa se aboneze la mesajele trimise de catre broker, in specific, sistemul se aboneaza la topic-ul "zigbee2mqtt/bridge/devices", unde zigbee2mqtt este topic-ul principal, definit in configurarea server-ului MQTT, iar "/bridge/devices" este topic-ul ce ne returneaza toate device-urile conectate in retea. Sistemul realizeaza abonarea la acest topic, cu ajutorul metodei "subscribe()", care este apelata in callback-ul on_connect_finder (Figura 5.6), deoarece in cazul in care conexiunea esueaza la un anumit moment, vrem sa ne reabonam la urmatoarea conexiune.

Figura 5.7: Device Finder on_message callback

```
def on_message_finder(self, client, userdata, message):
    # global messages
    # Start creating the json format with {devices:[]}
    q = Queue()
    first = "{\"devices\":"
    last = "}"
    # Get the payload and save it
    payload = str(message.payload.decode("utf-8"))
    # Build the payload to have a json format so its easier to access
    payload = first + payload + last
    # Save the payload so we can work with it
    q.put(payload)

    while not q.empty():
        message = q.get()
        self.data = json.loads(message)

    print(self.data)
    self.data = self.humanize_device_data(self.data)
    self.client.disconnect()
```

Urmatorul pas pe care il face sistemul, este de a pregati mesajul receptionat de client, cu ajutorul callback-ului `on_message_finder` (Figura 5.7), pentru a fi afisat pe pagina web. Pentru a realiza acest lucru, aplicatia prepara mesaul intr-un format JSON, care trece, la randulul lui, printr-o serie de simplificari, deoarece multe dintre field-urile primite in mesaj, nu sunt de folos.

Figura 5.8: Device Finder humanizing data

```
def get_devices_from_data(self, raw_data):
    data = []
    device_list = raw_data['devices']
    for i in range(0, len(device_list)):
        data.append(device_list[i])
    return data

def humanize_device_data(self, data):
    data_list = self.get_devices_from_data(data)
    output_data = []
    for device in range(0, len(data_list)):
        if data_list[device]['definition'] != None:
            output_data.append(data_list[device])

    return output_data
```

Deoarece este posibil ca sistemul sa gasesasca mai multe device-uri, iar acestea sa fie returnate impreuna, vrem sa le despartim pentru a le putea filtra mai usor.

Aceasta lista este trimisa mai departe catre sistemul de filtrare (Figura 5.8), unde cautam dispozitive fara definitii, acestea fiind fie incompatibile cu sistemul, fie dispozitivul este coordonatorul (antena de ZigBee).

Cand sistemul a terminat de filtrat datele receptionate, acesta se deconecteaza de la server-ul MQTT, deblocheaza thread-ul principal pe care ruleaza web server-ul Django, si ramane deconectat pana cand instantiam iara clasa DeviceFinder.

Urmatorul pas, este de a afla starea device-ului pe care sistem-ul l-a detectat. Pentru asta construim o noua clasa:

Figura 5.9: Device Finder humanizing data

```
class DeviceState(object):
    def __init__(self):
        try:
            self.data = DeviceFinder().data
            self.status = None
            self.BROKER_IP = "192.168.100.152"
            self.client = mqtt.Client("P2")

            self.client.on_message = self.on_message_state
            self.client.on_connect = self.on_connect_state

            self.client.connect(self.BROKER_IP, 1883, 60)
            self.client.loop_start()
```

Aplicatia urmareste acelasi principiu de abonare, se creeaza un client, care se conecteaza la server. Insa in plus, aici apelam clasa ce identifica device-urile, si le stocam in dictionarul "data" definit la initializarea clasei. Folosim metoda "loop_start" in loc de "loop_forever", deoarece nu dorim sa mai blocam thread-ul principal al web server-ului.

Figura 5.10: DeviceState

```
def on_connect_state(self, client, userdata, flags, rc):
    # Subscribing in on_connect() means that if we lose the connection and
    # reconnect then subscriptions will be renewed.
    self.subscribe_to_device_status(self.data)
```

(a) DeviceState on_connect callback

```
def subscribe_to_device_status(self, data):
    for device in data:
        device_name = str(device['friendly_name'])
        self.client.subscribe(f"zigbee2mqtt/{device_name}")
```

(b) DeviceState Subscribe mechanism

Mecanismul de abonare este similar, insa de aceasta data, aplicatia se aboneaza la topicuri diferite (Figura 5.10 (a)). Pentru a putea afla detalii despre starea dispozitivelor, vrem sa ascultam mesajele din topicul device-ului particular. Astfel, din datele filtrate in DeviceFinder, ne pregatim numele dispozitiv-ului si ne abonam la fiecare dispozitiv din lista. (Figura 5.10 (b))

Figura 5.11: Device State message example bulb

```
pi@raspberrypi:~$ mosquitto_sub -t "zigbee2mqtt/0x00178801083ba6fd"
{"brightness":139,"color_mode":"color_temp","color_temp":250,"color_temp_startup":153,"linkquality":34,"state":"ON","update":{"state":"available"},"update_available":true}
```

Sistemul primește ca și mesaj, o structură asemănătoare cu cea din figura 5.11. Fiecare dispozitiv poate avea parametrii diferiți, în cazul acesta avem un bec Philips Hue.

Figura 5.12: Device State message example sensor

```
pi@raspberrypi:~$ mosquitto_sub -t "zigbee2mqtt/0x00124b0022659906"
{"battery":78,"battery_low":false,"contact":false,"linkquality":39,"tamper":false,"voltage":2900}
```

Însă dacă ar fi fost un senzor de contact, sistemul ar fi primit mesajul din figura de mai sus (Figura 5.12).

Pentru a reține device-urile găsite în rețea, aplicația folosește un model de Device, cu care se populează baza de date.

Figura 5.13: Device database models

```
class ZigbeeDevice(models.Model):
    ieee_address = models.CharField(primary_key=True, max_length=30)
    type = models.CharField(max_length=30)
    supported = models.BooleanField()
    friendly_name = models.CharField(max_length=30)
    model_name = models.CharField(max_length=30)
    vendor = models.CharField(max_length=30)
    power_source = models.CharField(max_length=30)
```

(a) Standard ZigbeeDevice model

```
class ZigbeeExposes(models.Model):
    device = models.OneToOneField(ZigbeeDevice, primary_key=True, blank=True, on_delete=models.CASCADE)
    exposes = models.JSONField()
    state = models.JSONField()
```

(b) ZigbeeDevice exposes extension

În figura 5.13 (a) se observă modelul unui dispozitiv Zigbee. Câmpurile din model sunt datele importante pe care le primim de la server-ul MQTT. Pe lângă avem modelul ZigbeeExposes (figura 5.13 (b)), care este conectat la fiecare dispozitiv, extinzând modelul principal cu starea dispozitivului și acțiunile/parametrii pe care îi expune.

Figura 5.14: Device State loading models

```
def load_models(self, data):
    for i in data:
        if 'status' in i:
            model_data = ZigbeeDevice(
                friendly_name=i['friendly_name'],
                type=i['type'],
                supported=i['supported'],
                ieee_address=i['ieee_address'],
                model_name=i['definition']['model'],
                vendor=i['definition']['vendor'],
                power_source=i['power_source'],
            )
            model_data.save()
            model_data_exposes = ZigbeeExposes(
                device=model_data,
                exposes=i['definition']['exposes'],
                state=i['status']
            )
            model_data_exposes.save()
```

În figura 5.14, putem observa mecanismul de încărcare a modelelor. Acestea sunt populate cu datele ce au fost obținute până acum, după care sistemul le salvează în baza de date.

Datele prioritizate sunt cele ce pot fi înțelese ușor de către utilizator, precum adresa unică a dispozitivului (`ieee_address`), numele (`friendly_name`), modelul (`model_name`), producătorul (`vendor`) etc. Pentru a putea afla parametrii de control a dispozitivului, extindem modelul cu starea acestuia (`state`), și parametri expusi (`exposes`).

Pentru a putea vizualiza mai bine aceasta baza de date, folosim pagina de admin a sistemului. (figura 5.15)

Figura 5.15: Device State message example sensor

Home · Core · Zigbee devices · ZigbeeDevice object (0x00124b0022cf73f3)

AUTHENTICATION AND AUTHORIZATION

- Groups + Add
- Users + Add

CORE

- Zigbee devices** + Add
- Zigbee exposes + Add
- Zigbee groups + Add

Change zigbee device

ZigbeeDevice object (0x00124b0022cf73f3) HISTORY

ieee address: 0x00124b0022cf73f3

Type: EndDevice

☒ Supported

Friendly name: TemperatureDetector

Model name: SNZB-02

Vendor: SONOFF

Power source: Battery

Delete Save and add another Save and continue editing SAVE

Aceasta parte a website-ului este menita pentru a supraveghea baza de date. Chiar daca sistemul recunoaste schimbarile facute in aceasta pagina, nu le v-a trimite mai departe catre server-ul MQTT, deci urmatoarea data cand se realizeaza o cautare a device-urilor, valorile vor revenii inapoi la cele de dinainte.

Figura 5.16: Front page system

```
def index(request):
    context = {
        'devices': ZigbeeDevice.objects.all(),
        'state': DeviceState().status
    }
    html_template = loader.get_template('index.html')
    return HttpResponse(html_template.render(context, request))
```

(a) Index view

```
{% extends "layouts/base.html" %}

{% block content %}
<div class="container-fluid">
  <h1 class="text-center">Welcome to your home!</h1>
  <h3 class="text-center">Here are the devices that I have found</h3>
  <hr>
</div>
<div class="card-deck" >
  {% for device in devices %}
    <div class="card" >
      
      <div class="card-body">
        <a href="items/{{device.friendly_name}}"><h5 class="card-title">{{device.friendly_name}}</h5></a>
        <p class="card-text">Type: {{device.type}}</p>
        <p class="card-text">Vendor: {{device.vendor}}</p>
        <p class="card-text">Power source: {{device.power_source}}</p>
        <p class="card-text">Supported: {{device.supported}}</p>
      </div>
    </div>
  {% endfor %}
</div>

{% endblock content %}
```

(b) Index template

In figura 5.16 (a) se observa modul prin care aplicatia preia modelele din baza de date, acestea devenind contextul paginii 'index.html'.

Figura 5.16 (b) prezinta limbajul de template a framework-ului Django, cu care construim pagina principala a aplicatiei.

Pentru a putea identifica mai usor dispozitivul, se incarca o imagine cu ajutorul numelui modelului. Imagina exista insa, doar daca acest model este unul suportat de Zigbee2MQTT. In continuare sistemul preia datele din modelul ZigbeeDevice, si le afiseaza intr-un divizor de tip card, cu ajutorul lui bootstrap. Acest lucru se intampla pentru fiecare device din baza de date.

Detalii despre device

Pentru a putea afla mai multe detalii despre un device, sau chiar pentru a putea controla acest device, utilizatorul poate apasa pe numele device-ului respectiv, si v-a fi redirectionat catre o pagina unica a dispozitivului. Aici el poate afla mai multe detalii si poate sa modifice anumiti parametri, daca ii permite dispozitivul.

In acest pas, sistemul trece prin lista de parametrii pe care ii expune un dispozitiv, aflati anterior, si decide daca acestia pot fi modificati sau nu. Structura expunerii poate fi de doua feluri, una generica(de tip numeric, binary, text, enum, composite) sau una specifica(de tip light, switch, lock, climate etc.). Pentru a putea decide daca unul dintre parametri poate fi schimbat, ne folosim de proprietatea "access". Aceasta proprietate poate avea urmatoarele valori:

- "1". Acesta valoare semnifica faptul ca nu poate fi receptionata de comanda "/get", deci nu putem forta device-ul sa ne ofere informatii.
- "2". Aceasta valoare semnifica faptul ca dispozitivul raspunde doar la comanda "/set", deci putem doar sa modificam starea dispozitivului
- "5". Aceasta valoare semnifica faptul ca raspunde doar la comanda "/get" si ca se afla in starea publicata de dispozitiv.
- "7". Aceasta valoare inseamna ca device-ul raspunde atat la "/get" cat si la "/set", si se afla in starea publica de dispozitiv.

Pentru a decide ce fel de interfata de modificare ii se ofera utilizatorului, sistemul trece prin toti parametri si verifica tipul acestora. Acestea pot fi cele generice:

- "numeric". Daca tipul parametrului este unul numeric, inseamna ca device-ul expune o valoare numerica, ce are in mod optional, o valoare maxima (value_max), o valoare minima (value_min), valoarea cu care se poate incrementa la orice moment (value_step), unitatea cu care este masurata valoarea (unit), si o lista de valori presetate(presets)
- "binary". Acest tip indica faptul ca dispozitivul expune o valoare binara, ce are intotdeauna optiunile "value_on" si "value_off". De cele mai multe ori acest parametru este folosit pentru a controla daca un dispozitiv este pornit sau nu.
- "enum". Acest tip are intotdeauna o lista "values", unde se afla optiunile posibile pe care le poate alege un utilizator.

- "text". Acest tip semnifica faptul ca dispozitivul expune text.
- "composite". Acesta nu este tocmai un parametru, insa, reprezinta o combinatie a tipurilor generice prezentate mai sus, intr-un array numit features.

Sau pot fi cele specifice:

- "Light". Acest tip indica faptul ca dispozitivul expune o lumina, si poate avea urmatoarele caracteristici:
 - state
 - brightness
 - color_temp
 - color_xy
 - color_hs
- "Switch". Acest tip indica faptul ca dispozitivul este unul ce poate fi doar pornit sau oprit.
- "Fan". Acest tip indica faptul ca dispozitivul este un ventilator.


Figura 5.17: Device page view

```
def render_items(request, item_name):
    item = get_object_or_404(ZigbeeDevice, friendly_name=item_name)
    print(list(request.POST))
    if request.method == 'POST':
        if request.POST.get('rename', None) is not None:
            message = request.POST.get('rename', None)
            DeviceSetter().publishRename(item_name, message)
        elif request.POST.get(list(request.POST)[1], None) is not None:
            message = request.POST.get(list(request.POST)[1], None)
            DeviceSetter().publishCustom(item_name, list(request.POST)[1], message)
    return render(request, 'device.html', {'device': item, 'exposes': ZigbeeExposes.objects.all()})
```

Urmatorul pas pe care il realizeaza sistemul, este de a genera o pagina pentru fiecare device. Acest lucru este posibil cu metoda `get_object_or_404`, unde specificam numele device-ului. Pregatim contextul, unde avem device-ul in sine, si tot ce expune. Tot aici aplicatia se ocupa de controlarea form-urilor html, pe care user-ul le completeaza cu scopul de a modifica parametri device-ului.

Figura 5.18: Device page view

Philips



Change Device Name

Change

brightness=200

color_mode=color_temp

color_temp=153

color_temp_startup=153

linkquality=65

state=OFF

update={'state': 'available'}

update_available=True

state -> On/off state of this light

On

Off

brightness -> Brightness of this light

Default input

Change

color_temp -> Color temperature of this light

Default input

Change

color_temp_startup -> Color temperature after cold power on of this light

Default input

Change

effect -> Triggers an effect on the light (e.g. make light blink for a few seconds)

blink

breathe

okay

channel_change

finish_effect

stop_effect

In figura 5.18 se observa pagina unui device. Aceasta pagina este dinamica, fiecare device avand stari diferite, si parametri de modificat diferiti. In cazul acesta avem un bec Philips, pe care il putem controla.

Putem modifica luminozitatea, "temperatura culorii", "temperatura culorii la pornirea device-ului" si un efect pe care dispozitivul il v-a realiza.

Aceste lucruri sunt posibile datorita logicii din template-ul paginii.

Figura 5.19: Device page exposed

```
{% for exposed in exposes %}
  {% if device.friendly_name == exposed.device.friendly_name%}
    {% for states, values in exposed.state.items%}
      <ul class="list-group">
        <li class="list-group-item">{{states}}={{values}}</li>
      </ul>
    {% endfor %}
  {% endfor %}
```

Secventa de cod din figura 5.19 afiseaza starea actuala a dispozitivului. Sistemul preia datele din baza de date, si verifica ca datele expuse sa fie intradevar ale dispozitivului pe care il vizualizeaza utilizatorul. Acestea fiind in format JSON, preluam atat cheia cat si valoarea si le afisam pe toate in format de lista.

Figura 5.20: Device page logic

```
{% if item.features != None %}
  {% for feature in item.features %}
    {% if feature.type == 'binary' and feature.access != 1 %}
      <p>{{feature.name}} -> {{feature.description}}</p>
      <form class="form-inline" method="POST" action="/items/{{device.friendly_name}}/">
        {% csrf_token %}
        <div class="form-group mb-2">
          <button type="submit" class="btn btn-primary" name="binary/{{feature.name}}" value="On">On</button>
          <button type="submit" class="btn btn-primary" name="binary/{{feature.name}}" value="Off">Off</button>
        </div>
      </form>
    {% endif %}
    {% if feature.type == 'numeric' and feature.access != 1 %}
      <p>{{feature.name}} -> {{feature.description}}</p>
      <form class="form-inline" method="POST" action="/items/{{device.friendly_name}}/">
        {% csrf_token %}
        <div class="form-group mb-2">
          <input class="form-control" type="text" placeholder="Default input" name="numeric/{{feature.name}}">
          <button type="submit" class="btn btn-primary">Change</button>
        </div>
      </form>
    {% endif %}
    {% if feature.type == 'enum' and feature.access != 1 %}
      <p>{{feature.name}} -> {{feature.description}}</p>
      <form class="form-inline" method="POST" action="/items/{{device.friendly_name}}/">
        {% csrf_token %}
        <div class="form-group mb-2">
          {% for val in feature.values %}
            <button type="submit" class="btn btn-primary" name="enum/{{feature.name}}" value="{{feature.name}}-{{val}}">{{val}}</button>
          {% endfor %}
        </div>
      </form>
    {% endif %}
  {% endfor %}
{% endif %}
```

Figura 5.20 prezinta logica folosita de sistem pentru a decide daca device-ul expune parametri de tip composite (verifica daca contine un array 'features'). In cazul in care exista, afiseaza numele si descrierea fiecarui feature, in functie de tipul acesteia, in cazul in care acestea se pot modifica.

Figura 5.21: Front page system

```
{% else %}
{% if item.type == 'binary' and item.access != 1 %}
<p>{{item.name}} -> {{item.description}}</p>
<form class="form-inline" method="POST" action="/items/{{device.friendly_name}}/">
  {% csrf.token %}
  <div class="form-group mb-2">
    <button type="button" class="btn btn-primary" name="binary/{{item.name}}">On</button>
    <button type="button" class="btn btn-primary" name="binary/{{item.name}}">Off</button>
    <button type="submit" class="btn btn-primary">Change</button>
  </div>
</form>
{% endif %}
{% if item.type == 'numeric' and item.access != 1 %}
<p>{{item.name}} -> {{item.description}}</p>
<form class="form-inline" method="POST" action="/items/{{device.friendly_name}}/">
  {% csrf.token %}
  <div class="form-group mb-2">
    <input class="form-control" type="text" placeholder="Default input" name="numeric/{{item.name}}">
    <button type="submit" class="btn btn-primary">Change</button>
  </div>
</form>
{% endif %}
{% if item.type == 'enum' and item.access != 1 %}
<p>{{item.name}} -> {{item.description}}</p>
<form class="form-inline" method="POST" action="/items/{{device.friendly_name}}/">
  {% csrf.token %}
  <div class="form-group mb-2">
    {% for val in item.values %}
    <button type="submit" class="btn btn-primary" name="enum/{{item.name}}" value="{{val}}">{{val}}</button>
    {% endfor %}
  </div>
</form>
{% endif %}
```

(a) Index view

```
class DeviceSetter(object):
    def __init__(self):
        try:
            self.data = DeviceFinder().data
            self.status = None
            self.BROKER_IP = "192.168.100.152"
            self.client = mqtt.Client("P3")

            self.client.connect(self.BROKER_IP, 1883, 60)
            self.client.loop_start()

        except Exception as e:
            print(e)

    def on_publish_setter(self, client, userdata, mid):
        print("Message has been published")
        self.client.disconnect()

    def publishCustom(self, device, topic_raw, message):
        topic = topic_raw.split("/")[1]
        print(topic)
        print(message)
        self.client.publish(f"zigbee2mqtt/{device}/set/{topic}", message)
```

(b) Index template

Figura 5.21 (a) prezinta cazul in care nu exista parametru de tip composite, in-seamna ca este un tip generic, astfel iteram o singura data asupra datelor, cu aceeasi logica mentionata mai sus.

Pentru a putea controla dispozitivele, sistemul trebuie sa trimita instructiuni catre server-ul MQTT. Acest lucru este realizat cu ajutorul unei clase suplimentare, destinate publicarii mesajelor cu instructiuni.

Figura 5.21 (b) contine clasa destinata publicarii acestor mesaje. Se creeaza un nou client ce se conecteaza la server-ul MQTT si se descrie o functie care se ocupa de publicarea mesajelor cu ajutorul metodei "publish" incluse de librerie. Functia primeste ca parametru device-ul pe care se v-a realiza modificarea, topic-ul (unul dintre parametri expusi de dispozitiv) si mesajul cu instructiunea pe care o v-a trimite (e.g 'state': 'ON')

Gruparea dispozitivelor

Figura 5.22: Group page

Home Groups

Here are your groups

Group Name [Create a new group!](#)

Add

ID: 1

Members:

0x00178801083ba111

0x00178801083ba6fd

Figura 5.22 prezinta pagina ce permite utilizatorului sa creeze grupuri noi, si sa acceseze grupuri deja create.

Utilizatorul poate decide daca doreste sa grupeze dispozitive impreuna. Acestea se realizeaza prin publicarea unui request catre broker, de catre clientul de publish. Mesajul trimis poate fi trimis catre urmatoarele topic-uri:

- zigbee2mqtt/bridge/request/group/members/add
- zigbee2mqtt/bridge/request/group/members/remove
- zigbee2mqtt/bridge/request/group/members/remove_all

Impreuna cu topicul, trebuie sa existe si un payload, acesta fiind de tipul {"group": GROUPNAME, "device": DEVICENAME}, unde GROUPNAME reprezinta numele grupului unde se v-a adauga sau se v-a scoate un device, si DEVICENAME reprezinta numele device-ului.

Pentru a realiza acest lucru, sistemul foloseste classa DeviceSetter (figura 5.21 (b)), unde sunt adaugate metode aditionale.

Figura 5.23: Group page

```
def publishCustomGroupDevice(self, topic, group, device):
    message_sent = {
        'group': group,
        'device': device
    }
    #MQTT only acknowledges strings, ints, floats, or bytes as messages, so we have to convert the dictionary
    #to a string
    message_sent_string = json.dumps(message_sent)
    self.client.publish(f"zigbee2mqtt/bridge/request/group/members/{topic}", message_sent_string)

def publishCustomNewGroup(self, topic_raw, message):
    message_sent = {
        'friendly_name': message,
    }
    #MQTT only acknowledges strings, ints, floats, or bytes as messages, so we have to convert the dictionary
    #to a string
    message_sent_string = json.dumps(message_sent)
    self.client.publish(f"zigbee2mqtt/bridge/request/group/{topic_raw}", message_sent_string)
```

Figura 5.23 contine codul utilizat de sistem pentru a publica instructiuni catre server-ul MQTT. Functia publishCutomGroupDevice, permite adaugarea unui nou device intr-un grup existent iar functia publishCustomNewGroup permite crearea unui grup nou.

Capitolul 6

Concluzii

Sisteme inteligente pentru case, sau sistemele de automatizare a casei, devin din ce in ce mai importante in viata de zi cu zi, intrucat oamenii doresc sa isi usureze viata, mai ales in aceste perioada grea. Acest tip de aplicatie devine din ce in ce mai comun datorita companiilor mari din domeniul IT (precum Google, Apple, Amazon etc), insa majoritatea persoanelor interesate in astfel de sisteme doresc control complet asupra sistemului, proiectele open-source destinate controlului casei devin mai populare cu zi ce trece. Insa multe dintre aceste proiecte sunt lente, sau chiar abandonate, cu putin spatiu de extindere, fie limbajului de programare ales, fie arhitecturi alese.

Asadar, proiectul discutat in acest document, decide sa rezolve aceste probleme cu ajutorul tehnologiilor si arhitecturi folosite. Avantajele oferite de Django acopera problemele discutate mai sus, intrucat acest cadru ofera posibilitatea de a extinde foarte usor aplicatia web fara a afecta viteza aplicatiei, iar utilizarea libreriei `paho-mqtt` pentru Python, ofera o solutie pentru problema modularitatii altor proiecte de acest tip. Nu mai este nevoie de module aditionale pentru a putea controla device-urile IoT din retea, intrucat conectivitatea client-server este servita cu ajutorul libreriei mentionate.

Pe viitor, aceasta aplicatie va extinde functionalitatea prin crearea unor pagini aditionale unde se pot vedea tiparele utilizatorului, cu ajutorul machine learning-ului, lucru usor de realizat datorita limbajului Python. Se v-a realiza o revizie a interfetei pentru a imbunatati experienta utilizatorului.

Bibliografie

<https://www.asw.ro/ce-inseamna-internet-of-things-si-care-sunt-schimbarile-aduse-de-acesta/>
<https://www.mobiup.ro/blog/internet-of-things-ce-inseamna-si-cum-iti-poti-transforma-casa-intr-o-locuinta-inteligenta.html>
<https://www.electromaker.io/blog/article/9-best-raspberry-pi-smart-home-software-options>
<https://www.elprocus.com/home-automation-systems-applications/>
<https://www.techradar.com/best/best-home-automation-systems>
<https://iot.mozilla.org/about/>
<https://pypi.org/project/paho-mqtt/>
<https://www.hivemq.com/blog/how-to-get-started-with-mqtt/>
<https://docs.djangoproject.com/en/3.2/>