UNIVERSITATEA BABEŞ-BOLYAI

Facultatea de Ştiinţe Economice şi Gestiunea Afacerilor

Informatică Economică

Lucrare de licenţă

Absolvent,

Sebastian Călin **Coroian**

Coordonator ştiinţific,

Conf. univ. dr. Ramona **LACUREZEANU**

2022

UNIVERSITATEA BABEŞ-BOLYAI

Facultatea de Ştiinţe Economice şi Gestiunea Afacerilor

Informatică Economică

Lucrare de licenţă

Dispozitiv IoT pentru automatizarea întrerupătoarelor realizat în C++, Java și GO

Absolvent,

Sebastian Călin **Coroian**

Coordonator ştiinţific,

Conf. univ. dr. Ramona **LACUREZEANU**

2022

Rezumat

Conținutul acestei lucrări vizează un dispozitiv IoT care are rolul de a ridica gospodăriile obișnuite la rangul de unități inteligențe. Dispozitivul apasă întrerupătoare care necesită acțiune fizică (întrerupătoare pentru lumină, uși de garaj, centrale termice, etc.), astfel automatizând procesele aferente fără a fi necesară schimbarea întregului sistem electronic. Produsul este destinat să reducă costurile ridicate ale achiziționării sistemelor de tip "unitate inteligentă", acesta având drept țintă gospodării personale, clădiri de birouri și hale industriale. S-a realizat un dispozitiv IoT (cod sursă scris în C++) capabil să acționeze întrerupătoare, un server web (scris în GO) care manageriază o bază de date și mediază traficul dintre dispozitiv și utilizator, dar și o aplicație mobilă (scrisă în Java) pentru ca dispozitivul să fie accesibil publicului. Ca tehnologii s-au utilizat: MQTT, REST, WiFi, SQL, JSON și Android. Obiectivul acestuia este să ajute la tranziția spre digitalizare, aducând un plus de valoare dispozitivelor non-inteligente.

Cuprins

Abrevieri

IoT,	Internet of things
HTTP,	Hypertext Transfer Protocol
MQTT,	Message Queuing Telemetry Transport
TCP,	Transmission Control Protocol
IP,	Internet Protocol
RISC,	Reduced Instruction Set Computer
GPIO,	General-Purpose Input/Output
PWM,	Pulse-Width Modulation
I/O,	Input/Output
GPIO,	General-Purpose Input/Output
SPIFFS,	Serial Peripheral Interface Flash File System
SDFS,	Software Defined File System
FAT,	File Allocation Table
URI,	Uniform Resource Identifier
DC,	Direct Current
IDE,	Integrated Development Environment
SSID,	Service Set IDentifier
JSON,	JavaScript Object Notation
TLS,	Transport Layer Security
SQL,	Structured Query Language
CRUD,	Create, Read, Update, Delete
REST,	Representational State Transfer
PoS,	Point of Service
ICT,	Information and Communications Technology
DII,	Digital Intensity Index
ANL,	Agenția Națională pentru Locuințe
PISA,	Programme for International Student Assessment
INS,	Institutul Național de Statistică
OCDE,	Organizația pentru Cooperare și Dezvoltare Economică

ICT,	Information and Communications Technology
CoBrA,	Context Broker Architecture
OSI,	Open Systems Interconnection
API,	Application Programming Interface

Lista figurilor

Figuri:

Introducere

În această lucrare sunt prezentate tehnologiile cu ajutorul cărora s-a realizat proiectul de licență, cât și particularitățile acestuia. Drept motivație pentru alegerea acestei teme, a servit dinamica deosebită de evoluție a tehnologiilor de tip "locuință inteligentă", acestea devenind mai mult o normalitate decât o comoditate. În timp ce pentru construcțiile noi aceste tehnologii pot fi integrate sisteme inteligente relativ ușor, pentru cele vechi, trecerea la acest tip de tehnologie necesită schimbarea parțială sau totală a întregului sistem electric, tranziția fiind astfel mai costisitoare. Proiectul propune nu doar o alternativă, ci o soluție la acest fenomen prin montarea acestor dispozitive pe întrerupătoarele care pot fi acționate fizic, înlocuirea sistemului nefiind necesară. Acest dispozitiv prezintă utilitate atât pentru locuințele personale, dar mai ales pentru sedii de instituții, clădiri de birouri, hale industriale sau depozite, unde responsabilii cu administrația pot comanda de la distanță diferite aparate/lumini/roboți industriali.

Ca metodologie de lucru am folosit modelul Waterfall (cascadă). Conform *Douglas H., 2009*, modelul în cascadă este o metodologie de dezvoltare software care este structurată în 5 pași urmând un curs logic: cerințe, design, implementare, verificare și mentenanță. Am considerat că acest model este potrivit pentru tematica proiectului, întrucât acesta nu permite trecerea la următorul stadiu decât după ce stadiul actual este complet, ocolirea pașilor intermediari nefiind posibilă. Modelul facilitează bunul demers al lucrurilor, oferind siguranța că există o baza solidă la care se poate reveni, indiferent de stadiul în care se află lucrarea.

Lucrarea este structurată după cum urmează : Capitolul 1 prezintă în detaliu metodologia de lucru utilizată , modul în care aceasta se aplică pe proiect, precum și subcapitolele de tehnologii folosite în realizarea proiectului. Capitolul 2 prezintă analiza de business identificând justificări economice și modele consacrate aplicabile proiectului propus. Capitolul 3 prezintă aspectele de proiectare propriu-zisă a sistemului după care s-a realizat implementarea, utilizând modele consacrate. Capitolul 4 prezintă particularități ale implementării și explicații pe baza acestora. Capitolul 5 prezintă aspecte legate de evaluarea lucrării , cât și prezentarea măsurilor luate pentru testarea software. Capitolul 6 prezintă concluziile trase în urmă realizării proiectului, cât și posibilități de dezvoltare pe viitor.

1. Metodologia

Așa cum a fost specificat anterior, metodologia de dezvoltare software folosită este cea în cascadă (waterfall).

Metodologia waterfall, conform *Douglas, 2009* , a fost definită inițial de către *Winston W. Royce* în anul 1970, aceasta având un demers logic, câștigând susținerea managerilor. Cu toate acestea , există diferite variații ale acestui model, ele adoptând particularități ale proiectelor sau a metodologiei interne de lucru în cadrul companiilor.

Pentru proiectul de față, am decis să utilizez versiunea inițială a acestui model, așa cum a fost dezvoltat de *Winston* și prezentat de *University of Missouri* (Fig. 1).

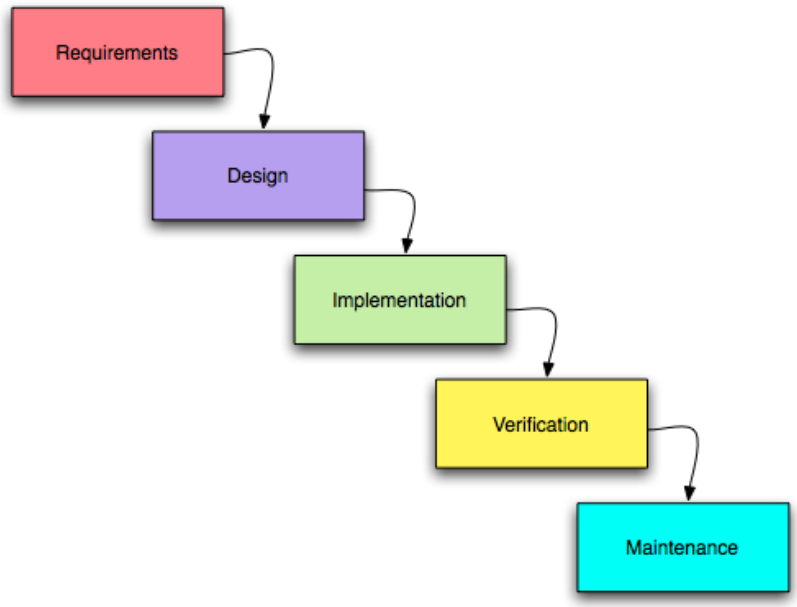

Figura 1. Structura modelului Waterfall
Figura reprezintă structura pe pași a modelului în cascadă așa cum este prezentat de către University of Missouri–St. Louis în hârtiile academice. Sursa: https://www.umsl.edu/~hugheyd/is6840/waterfall.html accesat la data de 22.03.2022

Așa cum se poate observa din fig. 1, primul stadiu este cel de a depista cerințele pentru proiect. Pentru a identifica acest lucru, s-a demarat o analiză pornind de la întrebarea “care este

cel mai apropiat competitor în domeniu?”. S-a ajuns la concluzia că dezvoltatorii de prize inteligente se axează pe rezolvarea aceleiași probleme pe care proiectul nostru dorește să o soluționeze (adaosul de inteligență sistemelor electronice non inteligente).

În urma analizei mai multor dezvoltatori a acestui tip de sistem (TP-Link, Wemo, Xiaomi, Wyze și Amazon), cu toate că sistemele sunt foarte particulare, ele toate au la bază două elemente comune: un cip care permite conectivitatea la rețea (Bluetooth/WiFi), cât și un releu care permite întreruperea fluxului de curent electric. Ca alternativă la aceste componente, pentru sistemul nostru am decis să utilizez o placă de dezvoltare cu modul WiFi integrat, întrucât acesta se poate programa, cât și un mecanism de tip piston, întrucât în cazul dispozitivului nostru, nu e nevoie să oprim fluxul de curent, ci să apăsăm un întrerupător.

Astfel, pornind de la aceste informații, se poate începe primul pas al modelului în cascadă. În continuarea subcapitolului vom încerca să identificăm tehnologiile de care avem nevoie în realizarea dispozitivului, iar în capitolul următor vom identifica probleme de piață. Astfel, vom completa stadiul de “cerințe”.

Stadiul de “design” va structura aceste probleme la nivel conceptual începând cu capitolul 2, demarând un proces de proiectare software în cadrul capitolului 3.

Componenta de “implementare” este prezentată în capitolul 4, unde se descrie în detaliu modul în care componentele dispozitivului au fost puse în funcțiune.

Componentele de “verificare” și “mentenanță” se regăsesc în capitolele 5 și 6, unde se descrie modul în care sistemul a fost testat, respectiv direcțiile de viitor ale proiectului.

Proiectul este alcătuit din 4 componente (atât software cât și hardware) după cum urmează: dispozitiv, server web, aplicație mobilă cât și un broker pentru protocolul de mesaje MQTT.

Fiecare dintre componentele menționate anterior rulează independent una de cealaltă, însă, pentru că întregul proces pe care acestea îl deservesc să aibă loc (de la clientul care apasă un buton din aplicație până la dispozitivul care apasă un întrerupător), este nevoie ca acestea să funcționeze concomitent. Fiecare componentă își îndeplinește funcțiile individual, însă ele comunică unele cu altele în lanț, astfel, o componentă inactivă poate duce la malfuncționarea sistemului. De exemplu, serverul poate să fie activ fară că brokerul MQTT să fie funcțional, însă acesta vă întâmpina o eroare în momentul în care face o cerere către acesta, întrerupând astfel procesul de funcționare al sistemului.

În subcapitolele următoare sunt descrise particularitățile fiecărei componente identificate, cât și tehnologiile pe care acestea le utilizează în mod individual pentru a deservi nivelului “cerințe” din cascadă.

1.1 Tehnologii Specifice dispozitiv

Dispozitivul este componenta fizică din acest sistem informatic. Acesta are rolul de a acționa întrerupătoarele fizice, fără a fi necesară prezența umană în încăpere. Acesta este alcătuit din două componente hardware, cât și o componentă software care le comandă pe acestea.

Pentru partea de hardware, cele două componente sunt un servomotor de 9 grame și o plăcuță de dezvoltare NodeMCU. Aceste două instrumente electronice sunt încorporate într-un corp printat 3D. Acesta are atât rolul de a proteja componentele, cât și întrebuințarea de adaptor pentru servomotor, fiind proiectat cu scopul de a apăsa întrerupătoare. Plăcuța de dezvoltare acționează ca un microprocesor, aceasta controlând servomotorul în conformitate cu codul sursă încărcat pe aceasta.

Pentru partea de software, codul sursă scris în C++ conține instrucțiuni pentru conectarea la rețea, conectarea la brokerul MQTT, procesarea instrucțiunilor și controlul servomotorului.

1.1.1 NodeMCU

NodeMCU este o plăcuță de dezvoltare populară în domeniile electronic și IoT, care are integrat conform website-ului comerciantului *components101.com, 2021*, un modul de WiFi ESP8266. Acest modul dispune de funcții complete TCP/IP , cât și un microprocesor RISC L106 cu configurație 32-bit. Acest modul permite plăcii să demareze operații pe frecvențele WiFi, cât și să stabilească o conexiune la internet.

O altă caracteristică importantă a plăcuței, care a reprezentat și un factor de decizie la alegerea acesteia pentru proiect, este memoria flash. Această particularitate extinde orizonturile de dezvoltare, făcând astfel posibilă stocarea locală de fișiere, cât și găzduirea de portaluri web (datorită componentei TCP/IP).

Placa de dezvoltare NodeMCU se găsește sub diferite variații pe piața de electronice, aceasta aducând după sine multe clone de la diverși producători. Pentru acest proiect, am decis să folosesc un NodeMCU 1.0, așa cum este prezentată pe website-ul referențiat al producătorului.

Acest model standard are o dimensiune redusă (48 milimetri lungime, 32 milimetri lățime și o spațiere între pini de 2.5 milimetri conform *components101.com, 2021*) care îi permite integrarea cu ușurință în diverse proiecte. Aceasta are totodată și un consum redus de energie, având nevoie de doar 15 miliamperi atunci când nu este supraîncărcată. Ea dispune de 13 pini GPIO, dintre care 12 sunt digitali, iar unul analog. Pinii GPIO permit transmiterea diferitelor tipuri de semnale, pinii de la D1 la D8 (conform Fig. 2, unde se prezintă amplasarea pinilor) fiind capabili să transmită semnale de tip PWM, necesare pentru controlul servomotorului. În figură se poate vedea că toți pinii digitali I/O sunt localizați în partea dreaptă, împreună cu 2 pini de voltaj (3.3V), 2 pini pentru împământare (GND), un transmițător (TX) și un receptor (RX).

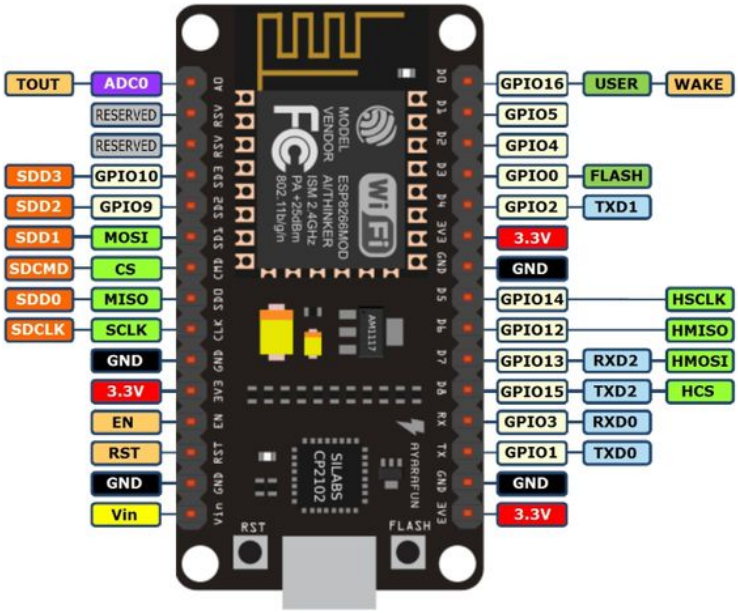

Figura 2. Așezarea pinilor pe placa NodeMCU 1.0
Figura reprezintă dispoziția fiecărui pin pe o placă de dezvoltare NodeMCU 1.0. Sursa: website comerciant - *components101.com*, Accesat la data de 02.04.2022

Partea stângă începe cu pinul A0 folosit pentru conversia de semnal analog în digital, 2 pini rezervați care nu sunt folosiți, urmați de pinii SD3 până la SDCLK care au ca întrebuințare

conectarea unui card de memorie la sistem, un pin pentru împământare, un pin de voltaj 3.3V, pinii "*ENABLE*" și "*RESET*", un alt pin pentru împământare și un input de voltaj care poate fi folosit atât pentru a oferi 5V putere, cât și pentru a asigura energie plăcuței NodeMCU atunci când nu se dorește utilizarea mufei micro-USB.

Pe lângă acești pini, placa mai are încorporate două butoane. Butonul din stanga (RST) va reporni plăcuța atunci când este apăsat, în timp ce butonul din dreapta (FLASH) care este folosit atunci când se instalează firmware nou pe modulul ESP8266. În continuarea acestora, se mai regăsesc două LED-uri montate pe suprafața plăcii. Unul din ele este conectat la GPIO2 (și poate fi folosit de către utilizator), iar celalalt este conectat la GPIO12 (acesta luminând intermitent în momentul în care se încarcă un nou cod sursă)

O altă componentă a acestei plăcuțe pe care doresc să o prezint este memoria flash. În cadrul acestui proiect, este nevoie ca plăcuța să lanseze un portal web de configurare a dispozitivului, configurație pe care să o salvăm ulterior în memoria internă a dispozitivului. Modulul dispune de un spațiu de stocare de 4MB, suficient pentru a executa aceste operații.

Modulul ESP8266 de pe placa de dezvoltare NodeMCU suportă următoarele sisteme de fișiere: SPIFFS, LittleFS, SDFS, sau, dacă se folosește un card SD, chiar și FAT16 sau FAT32 depinzând de formatare. Pentru acest proiect, am ales să folosesc sistemul de fișiere SPIFFS întrucât este cel mai eficient în materie de alocare a memoriei.

Pentru a avea posibilitatea de a citi și a scrie în sistemul de fișiere, vom folosi biblioteca FS.h, care face parte din bibliotecile implicite ale Arduino, și acoperă sistemul SPIFFS pentru C++.

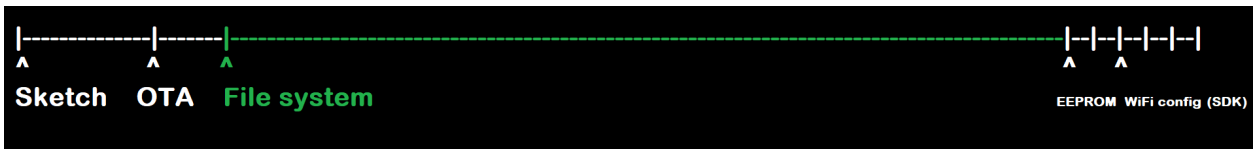

Figura 3. Structura memoriei flash ESP8266
Figura reprezintă partiționarea memoriei interne a modulului ESP8266, spațiul rezervat pentru sistemul de fișiere fiind marcat cu verde

Conform *Grokhotkov, 2017,* SPIFFS poate ocupa până la 70% din memoria flash a modulului (în cazul modulelor de 4MB, procentajul ocupat de sistemul de fișiere în cazul modulelor cu capacitate mai mică reducându-se proporțional), restul memoriei fiind ocupată de

codul sursă, actualizările de firmware, configurările pentru Wi-Fi, cât şi alte utilităţi ale modulului. SPIFFS urmăreşte o structură liniară, fapt pentru care nu suportă directoare de fişiere. În schimb, acesta inserează simboluri "/" în numele fişierelor, concatenând numele iniţial al acestora cu grupul din care fac parte, imitând astfel o cale de acces (de exemplu "spiffs/tmp/file.txt", unde "spiffs" este indicatorul unic pentru secţiunea din memorie asociată sistemului de fişiere, iar "/tmp/file.txt" este numele fişierului). Când dorim să extragem un fişier, o cerere HTTP conţine un URI compus din numele fişierului. SPIFFS încearcă să potrivească URI-ul respectiv cu numele fiecărui fişier stocat. O dată ce o potrivire este găsită, interpretorul SPIFFS citeşte numele fişierului de la coadă la cap până la găsirea primului separator "/", returnând fişierul cu numele original sub formă de răspuns.

1.1.2 Servomotor

Un servomotor este un mecanism pe bază de curent electric compus dintr-un motor DC, o cutie de transmisie şi un mecanism de control al poziţiei.

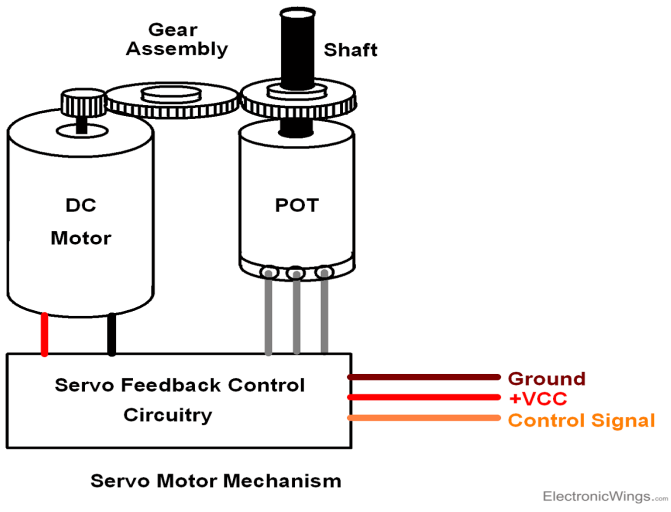

Figura 4. Compoziţia unui servomotor
Figura prezintă elementele care alcătuiesc un servomotor împreuna cu legăturile dintre acestea
Sursa: https://www.electronicwings.com/sensors-modules/servo-motor
Accesat la data de 02.04.2022

În fig. 4 se poate observa că axul motorului este legat de o roată zimţată (parte a unei cutii de transmisie după caz) care este la rândul ei ataşată de un potenţiometru. Se poate observa că atât motorul, cât şi potenţiometrul sunt alimentate de către circuitul de control. În timp ce

motorul este conectat cu doar prin colectorul de voltaj (VCC) și împământare (GND), potențiometrul este conectat și printr-un pin de ieșire. Acest pin transmite circuitului de control un semnal cu voltaj variabil care este proporțional cu poziția fizică a cursorului în jurul axei sale, comunicând în acest fel către circuitul de control poziția sa exactă.

Mecanismul de control primește un semnal PWM de la plăcuta de dezvoltare, prin intermediul căruia îi este transmisă poziția țintă a motorului. Acesta va transmite electricitate atât către motorul DC, cât și către potențiometru. În momentul în care motorul începe să se învârtă, acesta va acționa cutia de transmisie, care la rândul ei va acționa potențiometrul. Acesta va transmite în mod constant poziția sa către circuitul de control. Circuitul de control verifică semnalul primit de la potențiometru până când acesta se potrivește cu semnalul PWM, moment în care, acesta se oprește din a alimenta motorul.

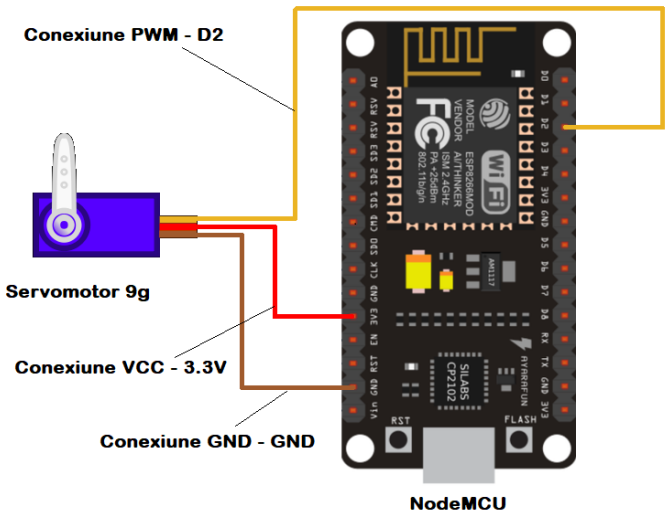

Figura 5. Conectarea NodeMCU la Servomotor
Figura indică legăturile dintre pinii NodeMCU și servomotor

Pentru acest proiect, am decis să folosesc un servomotor standard de 9 grame, întrucât acesta are o dimensiune redusă, fiind ușor de încorporat, totodată având puterea necesară pentru a acționa un întrerupător. Cablarea este făcută conform celei din fig. 5, acesta fiind alimentat cu 3.3 volți, iar semnalele de tip PMW vor fi citite de pe unul dintre pinii digitali (în cazul acesta, D2).

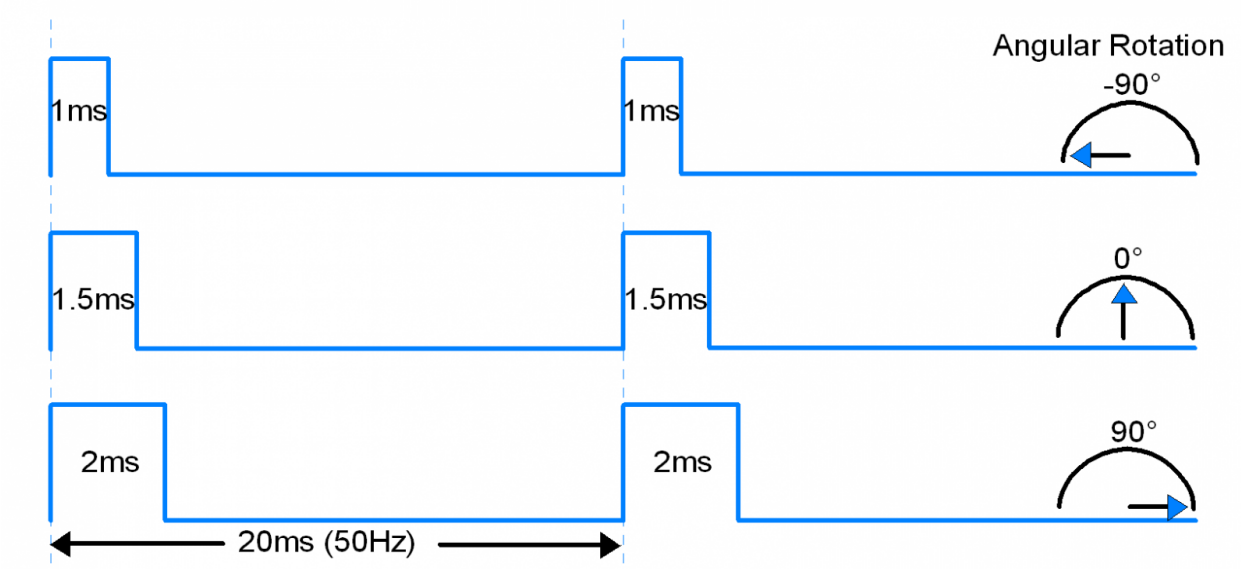

Figura 6. Interpretarea ciclurilor de lucru PWM
Figura afisaza modul în care servomotorul interpretează semnalele PWM indicând corespondentul acestora în grade. Sursa: https://www.electronicwings.com/sensors-modules/servo-motor accesat la data de 02.04.2022

În fig. 6 ne este prezentată interpretarea unui ciclu de lucru PWM de către servomotor. În electronică, un ciclu de lucru este o fracțiune a aparițiilor repetitive pe unitate de timp. În cazul servomotoarelor, acest ciclu are o lungime de 20 de milisecunde. În acest interval, sunt trimise două impulsuri de voltaj, egale în durata. Servomotorul are posibilitatea de a se roti 180 de grade (având o poziție implicită de 0 grade, cu posibilitatea de a se muta la -90 sau +90 de grade), semnalele corespunzând gradațiilor conform *ElectronicWings, 2022,* în felul următor:

- Un puls cu o lungime de 1.5 milisecunde (7.5% din ciclul de lucru) indică servomotorului revenirea la poziția implicită.
- Un puls cu o lungime de 1 milisecunde (5% din ciclul de lucru) indică servomotorului rotirea la poziția de -90 de grade
- Un puls cu o lungime de 2 milisecunde (10% din ciclul de lucru) indică servomotorului rotirea la poziția +90 de grade

Orice unghi din intervalul descris mai sus poate fi atins emițând semnale proporționale din intervalul [1,2] milisecunde.

Biblioteca Servo.h din Arduino se ocupă în fundal de toată această logică, punându-ne la dispoziție un obiect numit “Servo”. Funcțiile adiacente ne permit să atașăm un servomotor pe care să îl controlăm cu ușurință.

1.1.3 PlatformIO

PlatformIO estun IDE cu sursă deschisă, disponibil pe mai multe platforme. Acest IDE a fost dezvoltat cu scopul de a ușura dezvoltarea aplicațiilor de tip IoT. Acesta este compatibil cu un număr mare de microprocesoare, incluzând ESP8266, folosit în realizarea acestui proiect. PlatformIO este disponibil pentru o serie de medii de dezvoltare și editoare de text, incluzând Visual Studio Code (cel utilizat în dezvoltarea proiectului), unde este ușor instalabil sub forma de extensie.

Acesta facilitează accesul la toate librăriile din suita Arduino, cât și o serie de librării externe în cloud, acestea din urmă putând fi integrate cu ușurință utilizând fișierul “platformio.ini”. Acest fișier de configurare este generat automat odată cu crearea unui proiect PlatformIO. În acesta se pot modifica cu ușurință detaliile legate de tipul de microprocesor, frecvența de citire/scriere, dar și căii de acces pentru librăriile de care proiectul depinde.

Cu toate că comunitatea dezvoltatorilor pe surse deschise a creat framework-uri pentru aproape toate limbajele populare de programare, Arduino împreună cu C++, ambele suportate de PlatformIO, rămân în continuare cele mai populare alegeri pentru proiectele IoT, beneficiind de o comunitate mare și un multe resurse online. Pentru partea de hardware a acestui proiect, limbajul utilizat este C++.

1.1.4 WiFiManager.h

Pentru ca placa de dezvoltare să aibă acces la internet, este nevoie ca aceasta să se poată conecta la o rețea Wi-Fi locală. WiFiManager.h este o librărie externă, cu sursă deschisă care, conform documentației oficiale, ajută în realizarea unei astfel de conexiuni. Aceasta are funcții integrate pentru conectivitate, permițând dispozitivului să lanseze un portal web pentru configurarea credențialelor. Pentru ca clientul să poată avea acces la portalul web, WiFiManager are capacitatea de a configura un punct de acces la care utilizatorul se poate conecta. O dată conectat, portalul de configurare web se va lansa în mod automat. Pagina care este randată

dispune de un meniu din care utilizatorul își poate selecta SSID-ul. Modulul ESP8266 suportă doar frecvențe de 2.4GHz, așadar, cele de 5GHz nu vor fi afișate. Mai jos pe pagină există un formular de tip parolă în care utilizatorul își poate introduce credențialele. Configurația va fi salvată în sistemul de fișiere, după care, portalul de configurare va fi închis. WiFiManager are grijă de asemenea de conectarea automată. O dată ce o configurația a fost setată, de fiecare dată când modulul este pornit, acesta va încerca să se conecteze la rețeaua configurată, În cazul în care conexiunea a eșuat, modulul va deschide din nou portalul web și punctul de acces pentru a seta o nouă configurație.

O altă funcție utilă a acestei biblioteci este posibilitatea de a programa parametrii proprii în cadrul portalului web. În cazul de față, aceștia pot fi folosiți pentru a asocia dispozitivul cu unul din clienții existenți în baza de date, și pentru a denumi dispozitivul. Conținutul acestor câmpuri va fi returnat sub forma unui fișier JSON.

1.1.5 ArduinoJson.h

JSON este un format text popular și compact utilizat pentru a structura datele. Acesta este complet independent de limbajul de programare și poate fi citit și interpretat atât de oameni cât și de mașini. Conform *json.org, 2014*, JSON acoperă mai multe tipuri standard de date, cum ar fi șir de caractere, număr, boolean, sau vectori conținând aceste tipuri. Însă pentru acest proiect ne vom focusa pe tipul de dată “object”.

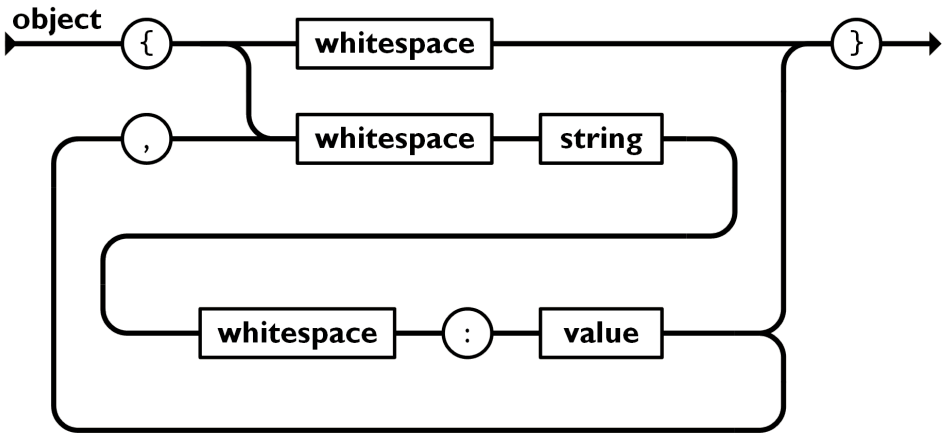

Figura 7. Structura tipului de data JSON object
Figura schiteaza arhitectura unei structuri de date de tip “object” in JSON
Sursa: https://www.json.org/json-en.html accesat la data de 02.04.2022

În fig. 7, este descrisă structura unui astfel de tip de dată. Un obiect JSON este o colecție de atribute cheie/valoare. Cheile sunt atribute de tip șir de caractere în timp ce valorile pot fi de orice tip. Cuvântul rezervat “NULL” ne permite să indicăm că o valoare este goală. Orice obiect care este reprezentat în JSON începe cu o acoladă deschisă. Obiectele pot să fie goale, cu un singur element sau cu elemente multiple. Un element este reprezentat de cheia obiectului (string) urmate de simbolul “:” care îi asociază acestuia o valoare. În cazul elementelor multiple, acestea sunt separate prin virgulă. După ce toate elementele dorite au fost listate, reprezentarea obiectului se încheie cu o acoladă închisă.

ArduinoJson este o librărie externă cu sursă deschisă care, conform documentației oficiale, ușurează lucrul cu formatul JSON. Acesta ne permite să creăm obiecte JSON pe care le putem ulterior concatena într-un fișier. Acest fișier îl vom folosi pentru a stoca parametrii de configurare, salvându-l în memoria SPIFFS. În continuare, ArduinoJson ne permite să citim și să suprascriem aceste date, făcând totodată câmpurile ușor de extractat individual.

1.1.6 PubSubClient.h

Pentru ca dispozitivul să poată fi controlat de la distanță, acesta folosește protocolul de comunicare MQTT. Un server care suportă acest tip de protocol, permite conexiunea mai multor clienți. Același server generează un canal de schimb de mesaje cu numele ID-ului fiecărui client. Fiecare client are un set diferit de privilegii (care este configurat de către administrator). În funcție de aceste privilegii, el poate posta mesaje pe anumite canale. În cazul de față, dispozitivele pot citi și scrie mesaje doar pe propriul canal, în timp ce serverul are acces la toate canalele.

Librăria PubSubClient.h este o altă dependență cu sursă deschisă care, conform documentației, ne permite efectuarea acestor operații pe protocolul MQTT. Aceasta ne permite să configurăm parametrii necesari pentru conexiunea MQTT (adresa IP a serverului, portul, credențialele), dar și să citim și să publicăm mesaje pe canalul propriu al dispozitivului, în acest fel recepționând instrucțiuni și oferind feedback către server.

1.1.7 Proiectare și printare 3D

Pentru ca componentele electronice să fie protejate, a fost necesară proiectarea unei carcase. Această carcasă servește de asemenea drept adaptor pentru scopul în care a fost proiectată, acela de a apăsa butoane. Acest lucru s-a realizat prin proiectarea unei șine zimțate cu transmisie la servomotor. Acest lucru permite mișcarea acesteia pe o singură axă.

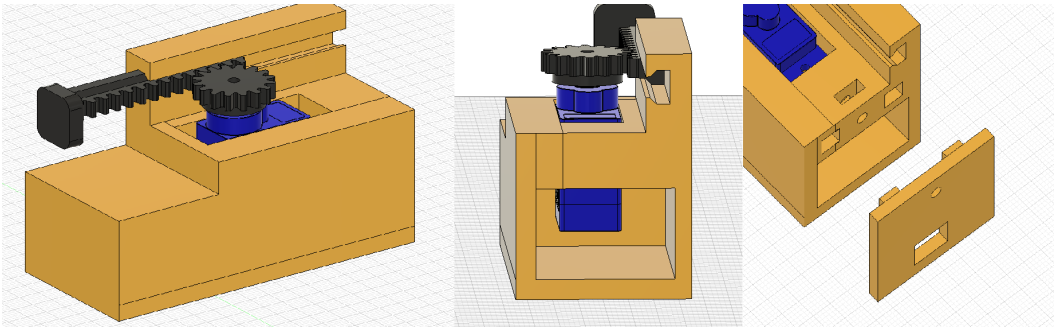

Figura 8. Randiție a carcasei pentru dispozitiv
În figură se regăsește o randiție din mai multe unghiuri a carcasei, proiectată în Fusion360 din diferite unghiuri

Facilitarea accesului la componentele electronice s-a realizat prin proiectarea unui capac în partea din spate a dispozitivului (așa cum se poate observa în ultima randare din fig. 8). Diferențele de lungime dintre partea inferioară și cea superioară a dispozitivului, facilitează montarea acestuia lângă un întrerupător.

Imprimarea acestor componente s-a realizat cu ajutorul unei Imprimante 3D Creality CR-10 V2, aceasta având parte de îmbunătățiri pentru senzorul de nivelare (s-a achiziționat un senzor BlTouch) și placă de baza (Creality V.2.4.7 Ultra Silent). Acesta de asemenea folosește motoare pas-cu-pas (stepper) pentru care este necesară instalarea de drivere TMC 2208.

Ca material de printare s-a utilizat un filament de la Fiberlogy (Nylon PA12). Alegerea acestui material a fost determinată de rezistența lui la forțele mecanice (conform testelor realizate de producator, *Fiberlogy, 2022*), șina carcasei fiind astfel supusă la riscul de a se curba in timp. Materialul de asemenea are rezistență mare la mediile acide, acesta fiind totodată și foarte ușor.

1.2 Tehnologii Specifice MQTT

MQTT este un protocol de internet care facilitează transmiterea de mesaje între dispozitivele care se află la distanță. Acesta are o arhitectură simplă și este adesea utilizat în proiectele IoT.

Protocolul MQTT se folosește de o serie de canale la care un un client se poate abona. Aceste canale pot fi deschise, sau, din motive de securitate după caz, închise, necesitând credențiale pentru acces. O dată abonat, un client poate vedea mesajele care intră pe canalul respectiv, dar și să posteze la rândul său mesaje pe acesta.

Un broker MQTT este un program care rulează pe un sistem folosind protocolul MQTT pentru a găzdui clienți și pentru a facilita transferul de mesaje între aceștia. În acest fel, nu există conexiuni între clienți (de tipul relațiilor many-to-many), ci doar câte o conexiune între fiecare client și broker. În acest fel, toți clienții rămân indirect conectați prin intermediul brokerului, acesta având grijă să redirecționeze mesajele corespunzător.

1.2.1 HiveMQ

Drept broker MQTT, am decis să folosesc serviciul HiveMQ, întrucât acesta dispune de un număr mare de unelte bine puse la punct (conexiuni securizate, generare de rapoarte, interfață grafică pentru a putea urmări cu ușurință canalele și dispozitivele, etc.).

HiveMQ oferă atât posibilitatea de a întreține un broker în cloud, cât și de a rula unul pe propriul dispozitiv/server. Eu am ales să folosesc brokerul pentru desktop, întrucât acesta este mai permisiv în materie de configurări.

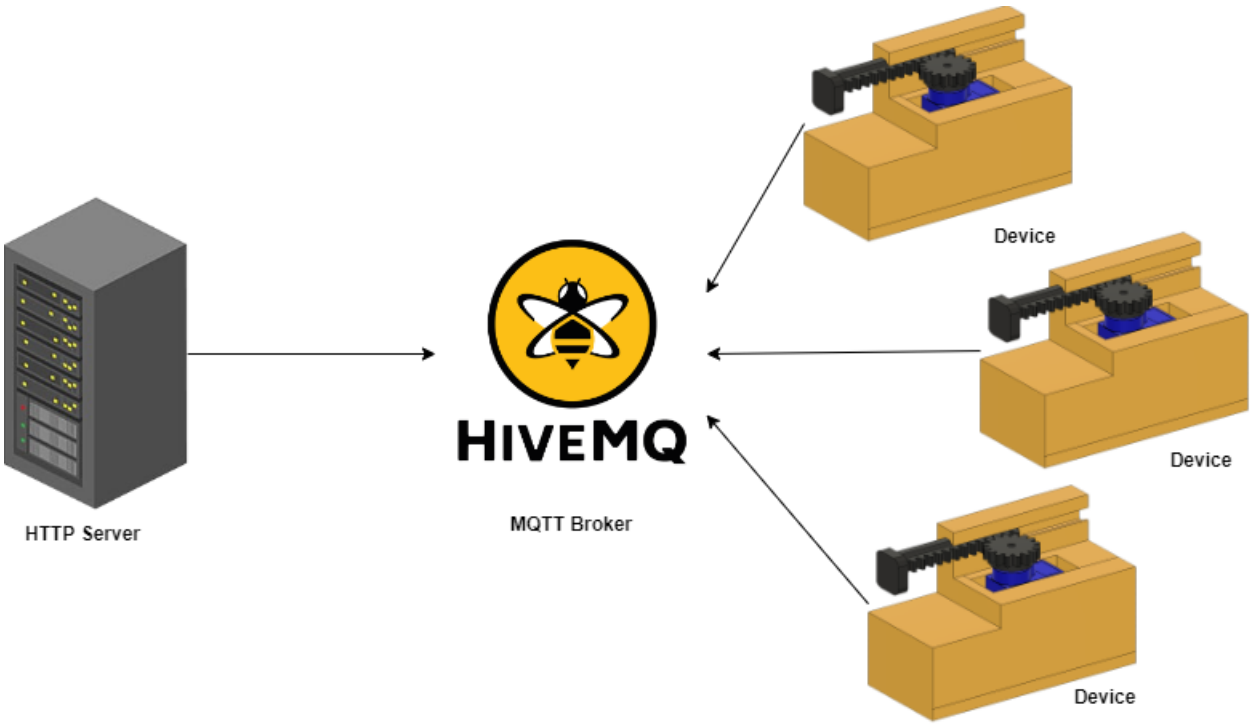

Figura 9. Model de conectivitate MQTT
Figura prezintă un model de relații aferent proiectului între clienți și brokerul MQTT

Așa cum se poate observa din fig. 9, atât serverul cât și dispozitivele au toate rolul de clienți. HiveMQ facilitează indirect conexiunea dintre acestea.

Fiecare dispozitiv este abonat la propriul canal de mesagerie, serverul fiind abonat la toate canalele. La configurarea inițială a unui dispozitiv, acestuia i se crează un canal la care se abonează, serverul se abonează de asemenea la canal, după care dispozitivul publică pe canal ID-ul și denumirea setată de proprietar, pentru ca serverul să înregistreze aceste informații în baza de date.

Brokerul urmărește în mod constant statusul clienților, astfel serverul putând știi dacă un dispozitiv este activ sau inactiv. În continuare, serverul va publica (în funcție de cererile utilizatorului) instrucțiuni pentru dispozitiv (extindere/restrângere braț servo).

1.2.2 MQTTBox

MQTTBox este o unealtă pentru dezvoltatori care facilitează testarea conectivității la protocolul MQTT. Acesta permite dezvoltatorului să creeze clienți virtuali care să se conecteze la un broker MQTT cu scopul de a observa dacă există erori. Din interfața acestuia se pot configura setări preferențiale pentru fiecare client în parte (tipul de conexiune TCP sau TLȘ, reconectare automată, deconectare după un anumit număr de secunde, etc.) ajutând astfel utilizatorul să verifice scenariile posibile, rezolvând anticipat cazurile de eroare, aducând un plus de inteligență pentru sistem.

MQTTBox permite atât abonarea cât și publicarea de mesaje. Acesta dispune de un panou în care se poate observa istoricul mesajelor citite/interceptate. Clienții MQTTBox au și posibilitatea de a se conecta la rubrici private dacă le sunt conferite credențiale.

1.3 Tehnologii Specifice Server HTTP

Serverul HTTP permite schimbul de mesaje între client și brokerul MQTT. Acesta are în administrare de asemenea și o bază de date MySQL.

Serverul este scris și configurat în limbajul GO, un limbaj cu sursă deschisă dezvoltat la Google care are o sintaxă similară cu cea a limbajului C. Am ales acest limbaj pentru dezvoltarea acestei componente a proiectului pentru că este rapid (fiind un limbaj compilat), ușor de învățat (atât ca sintaxă cât și ca semantică) și dispune de librării și unelte dezvoltate de comunitate care sunt foarte bine documentate. Dintre aceste instrumente externe, în realizarea serverului s-au folosit Goqu, Paho, Gin și MySQL-Driver.

1.3.1 Goqu

Goqu este o librărie externă cu sursă deschisă care, conform documentației oficiale, generează și execută interogări SQL. Am decis să utilizez acest framework întrucât facilitează o experiență mult mai plăcută atunci când se lucrează cu baze de date, făcând generarea de interogări ușoară, convertind valorile returnate în valori native, cu care compilatorul poate lucra. Acesta de asemenea poate genera operații CRUD. O caracteristică aparte a acestei librării este capacitatea

de a defini o structură de date care să capteze înregistrările din tabele în formatul de date în care acestea au fost definite.

În cazul în care utilizatorul dorește să implementeze un script în limbaj nativ SQL, Goqu permite acest lucru. Goqu vine totodată cu un set de dialecte care permit construcția corectă a interogărilor SQL pentru diferitele baze de date disponibile. Dialectele suportate la momentul actual sunt Postgres, SQLite3, SQL server și MySQL, care este folosit și în proiect.

1.3.2 Paho

Întrucât serverul are nevoie să comunice atât cu clientul, cât și cu dispozitivele într-un mod cât mai eficient, acesta trebuie să folosească protocoalele optime. Paho este o librărie cu sursă deschisă care permite conectarea la un broker MQTT, abonarea la canalele de topic, postarea de mesaje pe topic și primirea de mesaje.

Paho nu este o librărie dezvoltată implicit pentru limbajul Go, aceasta fiind parte a proiectului Eclipse Paho, având versiuni pentru 10 limbaje de programare. De asemenea, Paho este librăria care se regăsește în suportul pentru dezvoltatori oferit de HiveMQ.

1.3.3 Gin

Pentru a crea routerul web, am ales să folosesc Gin. Gin este o librărie HTTP cu sursă deschisă pentru limbajul Go, fiind cea mai rapidă librărie de acest fel și având o documentație foarte bine structurată.

1.3.4 MySQL

MySQL este unul dintre cele mai utilizate sisteme pentru manageriat baze de date. Pentru a ține evidența dispozitivelor astfel încât acestea să poată fi utilizate în procese, este nevoie de o bază de date. Bază de date MySQL rulează pe același dispozitiv împreună cu serverul web și brokerul MQTT, utilizând portul 3306.

MySQL folosește un model de tipul client/server, unicul client fiind serverul web, bază de date fiind dependentă de acesta în arhitectura sistemului.

Motivul alegerii MySQL în favoarea altor sisteme de acest tip este popularitatea, dispunând de o multitudine de resurse informaționale și fiind compatibilă cu diverse sisteme.

1.4 Tehnologii Specifice Aplicație Mobilă

Aplicația mobilă este interfața principală la care clientul are acces. Aceasta este dezvoltată în limbajul Java, mai precis în mediul de dezvoltare Android Studtio.

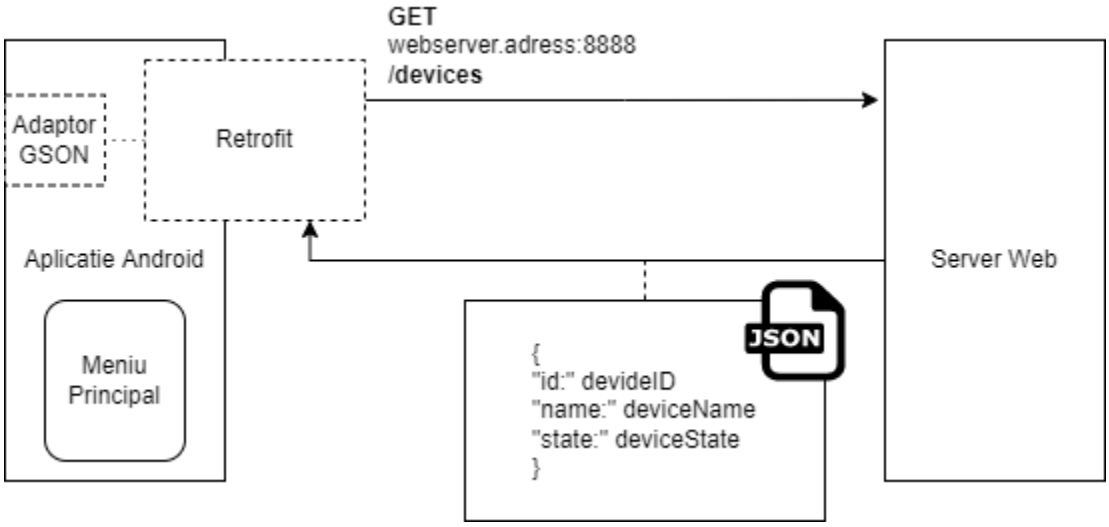

Figura 10. Schema aplicație mobilă
Figura prezintă modul de funcționare a aplicației mobile

Așa cum reiese din fig. 10, aplicația comunică cu serverul web pentru a transmite informații cu privire la dispozitive. În meniul principal se regăsește un obiect de tip list view în care se înșiră lista dispozitivelor disponibile în funcție de răspunsurile primite de la server. De aici, utilizatorul are posibilitatea de a seta statutul dispozitivelor în modul PUSH sau PULL.

1.4.1 GSON

GSON este o librărie cu sursă deschisă dezvoltată de Google, care ajută atât cu serializarea obiectelor Java în format JSON, dar și viceversa.

Aplicația mobilă schimbă în permanență cu serverul web informații referitoare la statusul dispozitivelor. Acest lucru este făcut prin intermediul cererilor HTML care transportă date în format JSON. Pentru a ușura procesul de lucru cu aceste tipuri de date, se utilizează GSON, acesta fiind capabil să serializeze clase complexe Java cu ușurință.

1.4.2 Retrofit

Retrofit este un client REST dezvoltat de Square (companie care are drept domeniu de activitate producția de cititoare de carduri de credit pentru terminalele PoS care sunt compatibile și cu dispozitivele Apple).

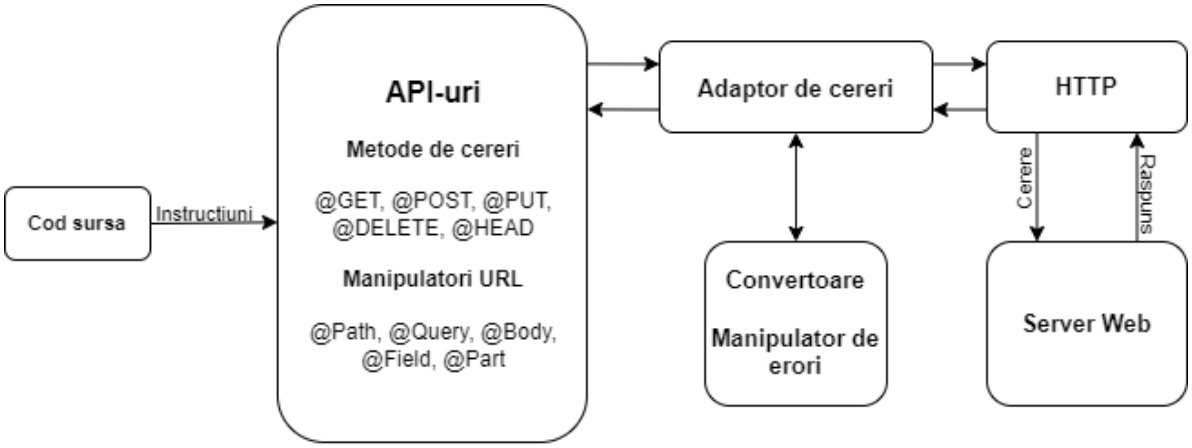

Figura 11. Model de funcționare Retrofit

Modul de procesare al unei cereri HTTP prin utilizarea Retrofit

Retrofit ușurează comunicarea cu serverele web, returnând datele cerute de la acestea formatate sub formă de obiecte Java. Această bibliotecă folosește intensiv adnotări Java (de exeplu “@GET” în același fel în care se folosesc adnotări de tipul “@Override”).

Adnotările se folosesc pentru a oferi metadate sintetice codului sursă. Aceste metadate pot consta în clase, metode, parametrii și nivele de acces sau variabile.

Retrofit poate încorpora o serie de convertoare (așa cum se observă în figura X). În cazul de față, răspunsul returnat de server este un fișier JSON. Astfel, pentru conversie, îi vom atașa un convertor din cadrul librăriei GSON.

2. Analiza Economică

Cu scopul de a valida justificarea economică a proiectului de licență, este necesară demararea unei analize în acest sens.

Pentru început, se dorește ca produsul să ajute la soluționarea unor probleme reale din piață.

Publicul țintă al produsului lucrării de licență este alcătuit atât din companii din mediul privat care doresc automatizarea clădirilor pe care acestea le au în administrație din considerente economice, cât și din persoane individuale care doresc să își automatizeze locuințele pentru un plus de confort.

Una din țintele Uniunii Europene pentru transformare digitală conform *comunicatului 118 din 2021 al Uniunii Europene* care cuprinde busola pentru dimensiunea digitală 2030, este ca cel puțin 90% din întreprinderile medii și mici, să ajungă la un nivel de bază al indicelui de intensitate digitală (DII).

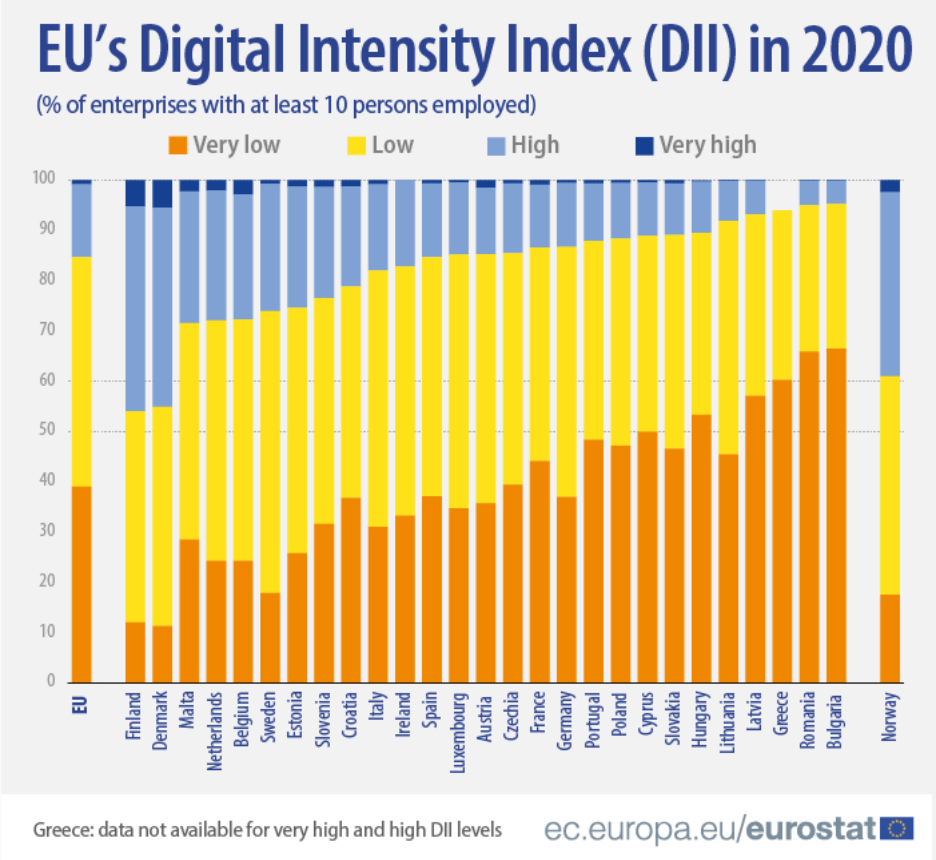

Figura 12. Statistica DII în țările Uniunii Europene
Figura prezintă nivelul indicelui de intensitate digitală în fiecare din țările membre UE
Sursa: https://ec.europa.eu/eurostat/web/products-eurostat-news/-/ddn-20211029-1 accesat la 04.04.2022

Indicele de intensitate digitală, este un indicator compus care măsoară nivelul de dezvoltare digitală a unei companii. Acesta se bazează pe un număr de 12 variabile din sfera ICT. Între aceste variabile se includ și automatizarea proceselor, utilizarea de tehnologii cloud și utilizarea de roboți, toate acestea fiind direct asociate produsului acestui proiect.

O evaluare a biroului european de statistică (*Eurostat, 2020*) care vizează întreprinderile cu cel puțin 10 angajați, arată că România împreună cu Bulgaria se află pe ultimele locuri la nivelul de digitalizare în Uniunea Europeană.

Așa cum se poate observa în fig. 12, 66% dintre companiile din România au un nivel foarte scăzut de digitalizare, acest procent fiind cu mult mai mare față de media europeană de 38%.

Produsul lucrării de licență vine ca o alternativă la soluționarea acestor probleme din companii, aducând plus valoare nivelului de digitalizare a companiilor acoperind trei dintre indicii utilizați în calculul acestui nivel.

Un studiu efectuat de compania germană *Statista, 2022* care centralizează datele din peste 120 de țări cu privire la piața de locuințe inteligente, arată că din 2014 până în 2021, sistemele inteligente pentru control și conectivitate (în care se încadrează și proiectul de licență) au ajuns de la 8.2 la 24 de milioane de utilizatori(fig. 13), acesta fiind cel mai popular segment de piață. Compania a demarat în cadrul studiului și o estimare, conform căreia, până în anul 2026, acest număr va ajunge la peste 86 de milioane.

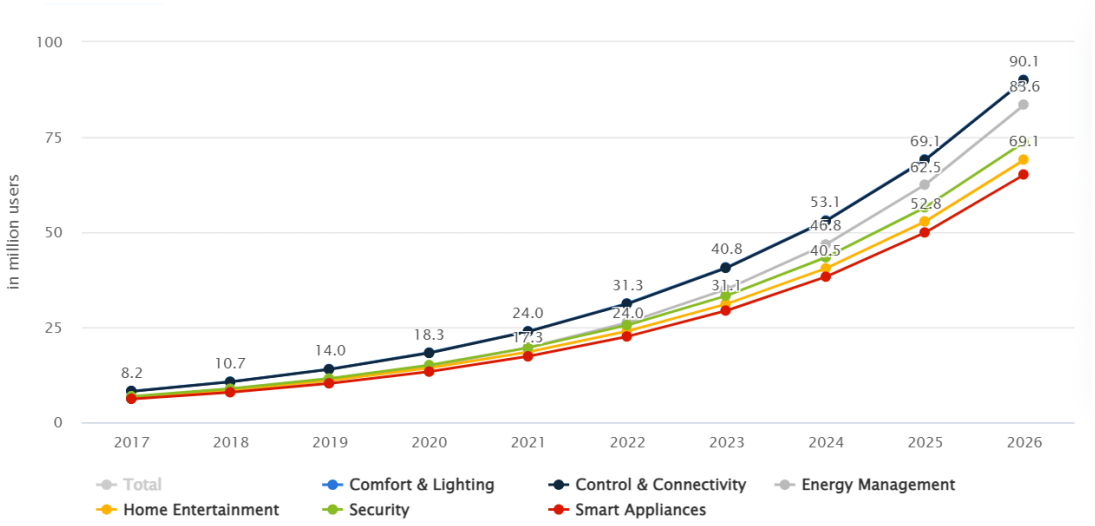

Figura 13. Evoluția numărului de utilizatori a sistemelor de tip Smart Home
Figura reprezintă evoluția pe ani a numărului de utilizatori a sistemelor de tip Smart Home
Sursa: https://www.statista.com/outlook/dmo/smart-home/europe#global-comparison accesat la 04.04.2022

Cu toate acestea, din datele puse la dispoziție de *Statista, 2022,* reiese că în România, în anul 2021 s-au cheltuit doar 259 de milioane de dolari pentru sisteme de tip locuință inteligentă (comparativ Germania unde s-au cheltuit 6.5 miliarde de dolari).

Pentru a putea identifica factorii care cauzează această discrepanță, s-a realizat următoarea diagramă Fishbone:

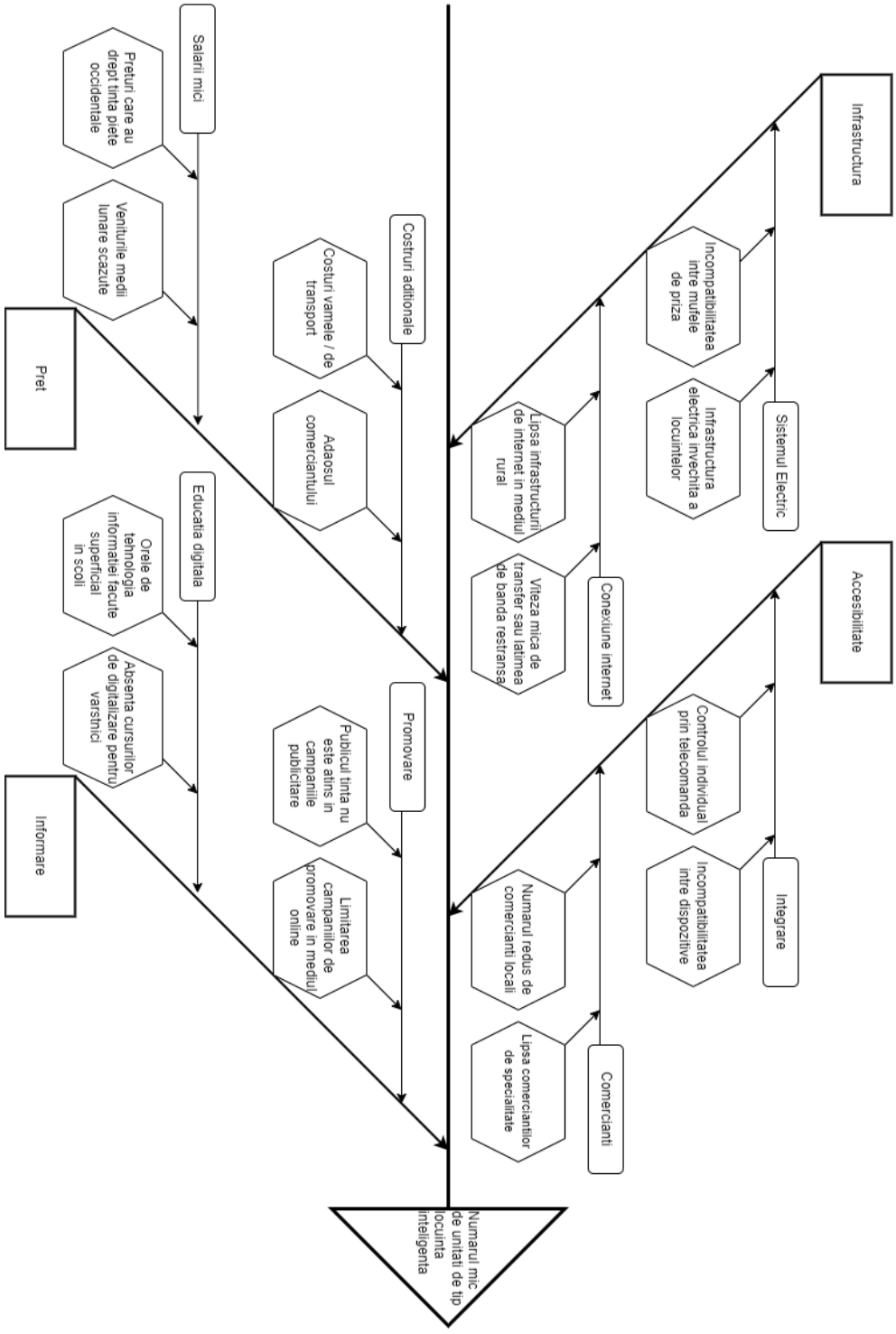

Figura 14. Diagrama Fishbone
Diagrama Fishbone care vizează problema centrală a numărului redus de locuințe inteligente

Diagrama Fishbone (de asemenea găsită și sub denumirea diagramă cauză și efect) este adesea folosită atunci când se dorește identificarea cauzelor care determină problematica principală (așa cum reiese din figura 14, problema principală identificată este "numărul mic de unități de tip locuință inteligență", făcându-se referire la piața din România).

Plecând de la problema principală, se determină categoriile principale de probleme, urmând ca ulterior, prin adresarea repetată a întrebării "De ce?" vizavi de acestea, cauzele să se ramifice și mai mult, diagrama având forma unui schelet de pește.

În diagrama din fig. 14 s-au identificat 4 cauze principale: Infrastructura, Prețul, Accesibilitatea și Informarea. Acestea la rândul lor au o serie de cauze particulare după cum urmează: pentru infrastructură s-au identificat drept categorii de probleme sistemul electric și conexiunea la internet, pentru accesibilitate avem categoriile de integrare și comercianți, pentru preț avem categoriile de salarii mici și costuri adiționale, iar pentru informar e avem categoriile de promovare și educație digitală.

▪ Incompatibilitatea între mufele de priză. Marii producători din domeniu se regăsesc în Statele Unite și China. Foarte multe dintre produsele de pe piață nu sunt adaptate pentru prizele de tip UE, acestea adesea necesitând achiziționarea adițională a unui adaptor. Acesta reprezintă un cost adițional și ocupă un volum mai mare. Între instalațiile electrice din Europa și cele din Statele Unite/China, există diferențe de voltaj în curentul alternativ. Acest lucru poate reprezenta o provocare pentru sistemele cu consum mai ridicat. Proiectul vine în sprijinul acestei problematici, putând fi alimentat cu microUSB.

▪ Infrastructura electrică învechită a locuințelor. Sistemele de tip locuință inteligentă au început să apară relativ recent. Multe dintre acestea se integrează direct în infrastructura electrică a locuinței în momentul construcției, făcând astfel parte din instalația electrică a locuinței. În cazul clădirilor mai vechi, trecerea la un altfel de sistem poate reprezenta o inconvenientă majoră întrucât acesta presupune schimbarea întregii instalații electrice și posibile modificări adiționale în structura acesteia.

O centralizare a datelor despre blocurile din România realizată de *teoalida.ro, 2020*, oferă o mostră gratuită a bazei de date, în care se pot găsi și informații referitoare la anul de construcție al acestora. Dintr-un total de 19058 de blocuri analizate, doar 1470 au fost ridicate

după anul 1990. În cazul acestora, trecerea la sisteme de tip locuință inteligentă este în special dificilă, întrucât, în multe dintre cazuri este nevoie de acordul asociației de proprietari.

Proiectul vine în sprijinul tuturor acestor locuințe, fiind proiectat pentru a fi montat pe întrerupătoare, automatizând astfel sistemele electronice tradiționale, eliminând astfel necesitatea de a schimba instalația în întregime.

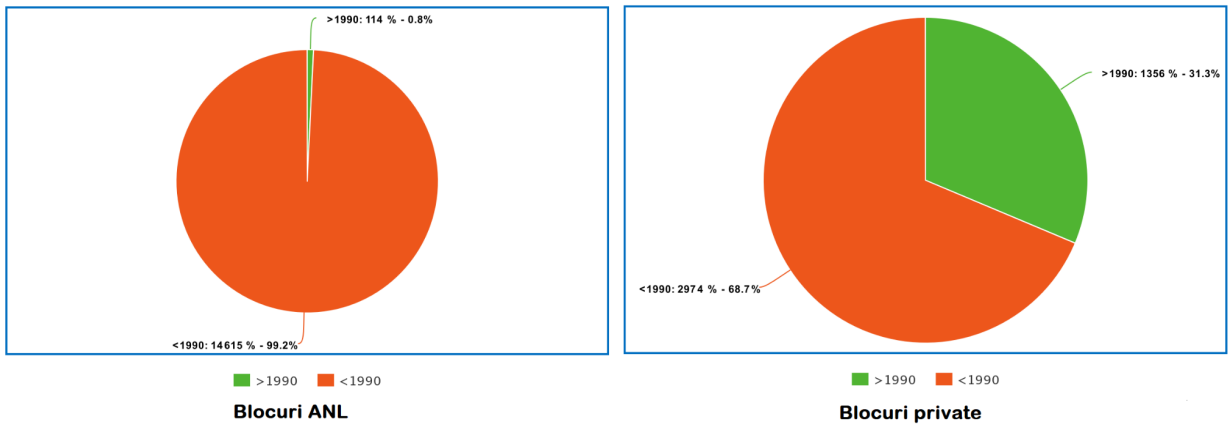

Figura 15. Statistică an de construcție blocuri
În partea stangă este prezentată statistica anului de constructie pentru blocurile ANL iar în partea dreaptă pentru cele private.

▪ Lipsa infrastructurii de internet în mediul rural. O publicație a institutului național de statistică arată că doar 73.1% din gospodăriile din mediul rural aveau acces la internet în anul 2021. Pentru gospodăriile care nu au acces la rețeaua de internet, integrarea dispozitivelor IoT devine dificilă.

O posibilă soluție la această problemă o reprezintă crearea unei rețele de intranet în aceste gospodării.

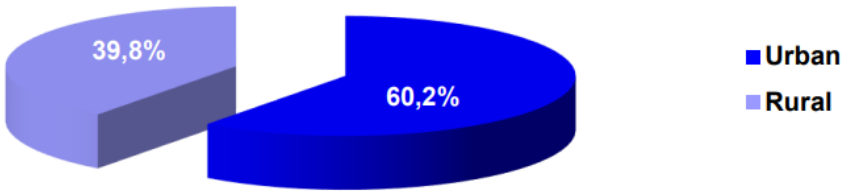

Figura 16. Statistică acces la internet după mediul de proveniență
Repartizarea gospodăriilor cu acces la internet din România după mediul de proveniență în 2021
Sursa:https://insse.ro/cms/ro/tags/accesul-populatiei-la-tehnologia-informatiei-si-comunicatiilor

▪ Viteză mică de transfer sau lățimea de bandă restrânsă. Din diverse motive, rețelele de internet pot fi foarte încete adeseori. Acest lucru poate îngreuna comunicarea pentru dispozitivele IoT, lucru care duce la malfuncționarea acestora. Dispozitivul IoT din proiect folosește protocolul MQTT pentru comunicare, acesta transferând pachete mici de date, funcționând chiar și pe lățimi de bandă restrânse.

▪ Controlul individual prin telecomandă. Un număr mare de dispozitive din gama Smart Home se folosesc de tehnologii infraroșu sau semnal analog pentru a transmite instrucțiuni. Acestea sunt de cele mai multe ori controlate prin telecomenzi. Semnalele pot fi receprionate de mai multe dispozitive în același timp, ducând la rezultate nedorite. Ca soluție pentru această problemă, proiectul se folosește de o aplicație mobilă drept controler.

▪ Incompatibilitatea între dispozitive. Mulți producători de sisteme Smart nu includ un mod pentru a putea integra dispozitivele cu unele străine (de exemplu, asistenți inteligenți). Soluționarea acestei probleme se poate face prin cereri HTTP către server, funcție care poate fi utilizată cu ușurință de către dezvoltatori.

▪ Numărul de comercianți locali. Dacă ar exista un producător de dispozitive IoT în România, numărul comercianților ar crește implicit, ușurând sistemul de adopție al acestor produse.

▪ Lipsa comercianților de specialitate. Este firesc să apară constrângeri în momentul în care se ia în considerare trecerea la o locuință inteligentă dacă nu există comercianți de specialitate în proximitate. În caz de defecțiuni de sistem, trimiterea în garanție poate fi un proces de lungă durată dacă nu există reprezentanțe în apropiere. De asemenea, comercianții locali pot oferi suport personalizat, în conformitate cu țara/regiunea în care se găsesc (limba, condiții meteorologice, etc.)

▪ Costuri vamale / de transport. Dat fiind faptul că pentru mărfurile importate care depășesc 10 euro (acesta fiind cazul pentru majoritatea produselor din această nișă), la care se adaugă și costurile provenite din transport, prețul produselor crește. În cazul unui producător cu sediul în Uniunea Europeană, aceste costuri sunt semnificativ mai mici.

▪ Adaosul comerciantului. Conform publicației *Institutului Național de Statistică, 2017*, care a măsurat rezultatele și performanțele întreprinderilor din comerț și servicii în anul 2017, în

cadrul întreprinderilor care specializate în sectorul de comerț al echipamentului informatic și de telecomunicații, procentul înregistrat de adaos comercial din cifra de afaceri este de 25.45%. Cifra de afaceri medie pe o întreprindere de acest fel este de 1353331 lei. Astfel, se poate deduce ca în anul 2017, o companie specializată în comerțul produselor informatice și de telecomunicații, a generat în medie un venit de 344422.73 lei din adaosul comercial.

În cazul unui producător care facilitează vânzarea produselor direct către client, acest adaos dispare, produsele fiind vândute la prețul lor original.

▪ Prețuri care au drept țintă piețe occidentale. Pentru producătorii de produse IoT, cea mai mare piață de desfacere este occidentul. Acest lucru influențează costul produselor, întrucât producătorii au parte de costuri mai mari în ceea ce privește chirii, salariați și alte cheltuieli terțe.

▪ Veniturile medii lunare scăzute. Conform datelor furnizate de platforma Eurostat, România este pe penultimul loc la venitul salarial din Uniunea Europeană. Acest lucru reprezintă unul dintre factorii care determină numărul mic de locuințe inteligente din România, produsele de acest tip fiind adesea mai scumpe.

▪ Publicul țintă nu este atins în campaniile publicitare. Datele cu privire la randamentul campaniilor de publicitate sunt păstrate în arhivele private ale companiilor, această problematică fiind doar speculativă. Această ipoteză se poate confirma prin realizarea unui chestionar la scară largă.

▪ Limitarea campaniilor de promovare la mediul online. Publicitatea la produsele IoT este rar întâlnită sub alte forme decât cea de reclamă digitală. O altă ipoteză este aceea că clienții sunt sceptici cu privire la aceste produse, campaniile de promovare în magazin (demonstrative) reprezentând o posibilă soluție.

▪ Orele de tehnologia informației predate superficial în școli. Testele PISA reprezintă o serie de studii ale OCDE pentru evaluarea sistemelor de învățământ. Acest lucru este realizat prin supunerea copiilor de 15 ani din țările membre la o serie de testări pentru măsurarea performanței școlare. Pentru testările din 2021 (care au fost amânate pentru anul 2022 datorită contextului pandemic), PISA a inclus un modul ICT. Aceste teste includ matematică, citire, științe exacte, rezolvare colaborativă de probleme. România a înregistrat la testele *PISA, 2018* o medie de 428 de puncte, media generală a țărilor fiind de 489 de puncte. Conform acestor date,

confirmate prin intermediul unei publicații de *Ministerul Educației, 2018*, România s-a situat pe locul 47 dintr-un total de 79 de țări evaluate. O îmbunătățire a abilităților digitale ale elevilor din ciclul pre-universitate va crește implicit și indicele de tehnologizare.

▪ Absența cursurilor de digitalizare pentru vârstnici. Conform *Statista, 2021*, aproximativ 19.23% din populatia Romaniei are vârsta de peste 65 de ani. Aceștia reprezintă un număr mare de potențiali utilizatori. Dintre aceștia, mai puțin de 15% folosesc internetul zilnic. Pentru a crește numărul acestora, este nevoie de familiarizarea lor cu tehnologiile ICT.

Pentru a putea valida motivele indicate, s-a realizat un chestionar cu ajutorul aplicației web Google Forms, destinat potențialilor utilizatori. Chestionarul este alcătuit dintr-o singură întrebare, și anume "Care credeți că sunt principalele motive pentru care românii nu cumpără produse IoT?".

Fiecare dintre persoanele chestionate are posibilitatea de a alege până la 3 variante de răspuns. Fiecare dintre aceste variante corespunde unei cauze de nivel 1 a diagramei Fishbone (fig. 14). În cazul în care utilizatorul nu consideră variantele suficient de relevante, acesta poate completa câmpul "other". În cazul în care vor exista mai multe răspunsuri de tip "other" care să facă referire la aceeași problemă, acestea vor fi ulterior luate în considerare și introduse drept cauze de nivel 1.

Variantele sunt următoarele:

- 1. Taxele vamale / adaosurile comerciale prea mari
- 2. Prețurile produselor sunt prea mari pentru veniturile din România
- 3. Campaniile de promovare nu sunt bine structurate
- 4. Lipsa educației digitale adecvate
- 5. Dispozitivele nu pot fi controlate din aceeași aplicație
- 6. Numărul redus de comercianti/service-uri/asistență
- 7. Incompatibilitatea cu instalația electrică
- 8. Lipsa conexiunii la internet / viteza redusă a internetului

În urma centralizarii răspunsurilor a 33 de persoane, rezultatele au fost următoarele:

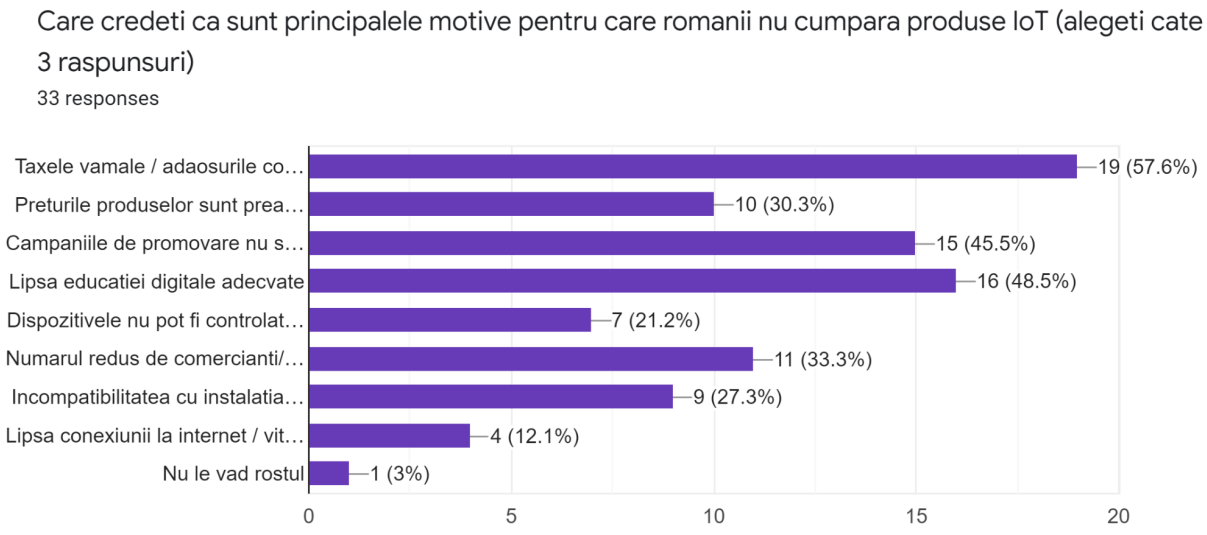

Figura 17. Statistica principalelor cauze a pieței restrânse
Statistica după centralizarea datelor formularului referitor la cauzele numărului restrâns de utilizatori ai produselor IoT în România

Procentele prezente în figura 13 (partea dreaptă) reprezintă rata persoanelor chestionate care au ales varianta respectivă drept răspuns la întrebarea adresată. De reținut este faptul că o persoană poate alege de la 1 până la 3 variante, astfel procentul total poate ajunge până la 300%.

În urma centralizării, o singură persoană a selectat varianta “other”, motivul inserat fiind “nu le văd rostul”.

La o primă vedere asupra graficului se poate observa că fiecare dintre cele 8 motive inițiale este relevant, având un procent de selecție de peste 10%. Cel mai votat motiv a fost cel referitor la creșterea prețului, provenită din taxe vamale și adaosuri comerciale, acesta având corespondentul “costuri adiționale” în diagrama Fishbone. Este de asemenea singurul motiv care a fost ales de peste 50% (mai precis 57.6%) din totalul persoanelor chestionate. Pe pozițiile următoare se clasează motivele legate de educația digitală și promovare cu procente de 48.5% respectiv 45.5%. Educația digitală este de asemenea motivul pe care eu l-am considerat cel mai relevant. În continuare avem numărul redus de comercianți, service-uri și servicii de asistență cu 33.3%, prețurile produselor care sunt prea ridicate pentru plafonul din România cu un procent de 30.3%, incompatibilitatea cu instalația electrică cu 27.3% și motivul referitor la accesibilitate și

incompatibilitatea între dispozitive cu 21.2%. Cea mai puțin votată dintre variante, dar totuși relevantă, este conexiunea slabă la internet cu un procentaj de 12.1%.

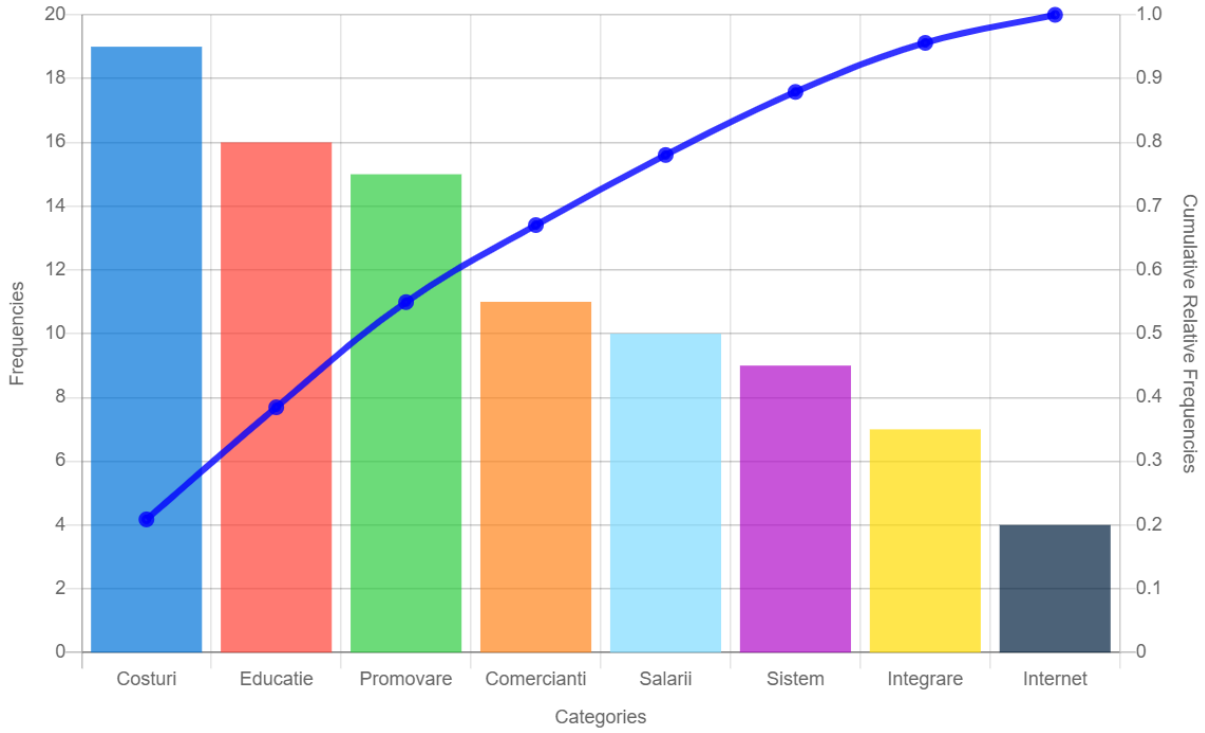

Figura 18. Diagrama Pareto

Diagrama Pareto a principalelor cauze a numărului redus de utilizatori de produse IoT în România

Pentru a putea vedea în ansamblu amplitudinea cauzelor așa cum au fost votate în chestionar, s-a realizat o diagrama Pareto (fig. 18).

Diagrama Pareto se bazează pe principiul lui Pareto (des întâlnit sub numele de regula 80/20) care spune că 80% din consecințe sunt realizate de 20% din motive (de exemplu, 20% din clienți generează 80% din vânzări, 80% din accidentele rutiere sunt cauzate de 20% dintre șoferi, 80% din performanță este cauzată de 20% din eforturi). Cu toate că acest principiu nu coincide mereu cu datele reale de intrare/ieșire, el oferă o premisă din perspectiva căreia se poate lansa o analiză.

Pe axa verticală a figurii 14 sunt reprezentate două variabile: frecvențele cauzelor, aferente numărului de voturi obținut după centralizarea datelor chestionarului (fiind reprezentate grafic prin benzi verticale de diferite culori), și frecvența cumulativ-relativă, care reprezintă

ponderea din întreg a cauzelor cumulate la un moment dat (reprezentată printr-o linie care crește pe axa y). În diagramă, datele sunt ordonate de la cele cu frecvența cea mai mare, la cele cu frecvența cea mai mică.

În acest fel, cu toate că principiul lui Pareto nu se poate identifica, diagrama ne ajută să identificăm cauzele care au impactul cel mai mare, pentru a le putea prioritiza. Astfel, se observă că primele 4 cauze cumulează 70% din totalul voturilor, în timp ce următoarele 4 cumulează restul de 30%. Aceste rezultate pot fi interpretate în diferite moduri, de exemplu, ca o analogie la principiul lui Pareto, primele 3 cauze cumulează 60% din totalul de voturi, ceea ce înseamnă că 37.5% din cauze realizează 60% din efecte.

Unele dintre aceste cauze nu pot fi soluționate în mod direct de către proiect (de exemplu educația sau absența internetului) însă majoritatea acestora pot fi rezolvate parțial sau total, validând astfel ideea.

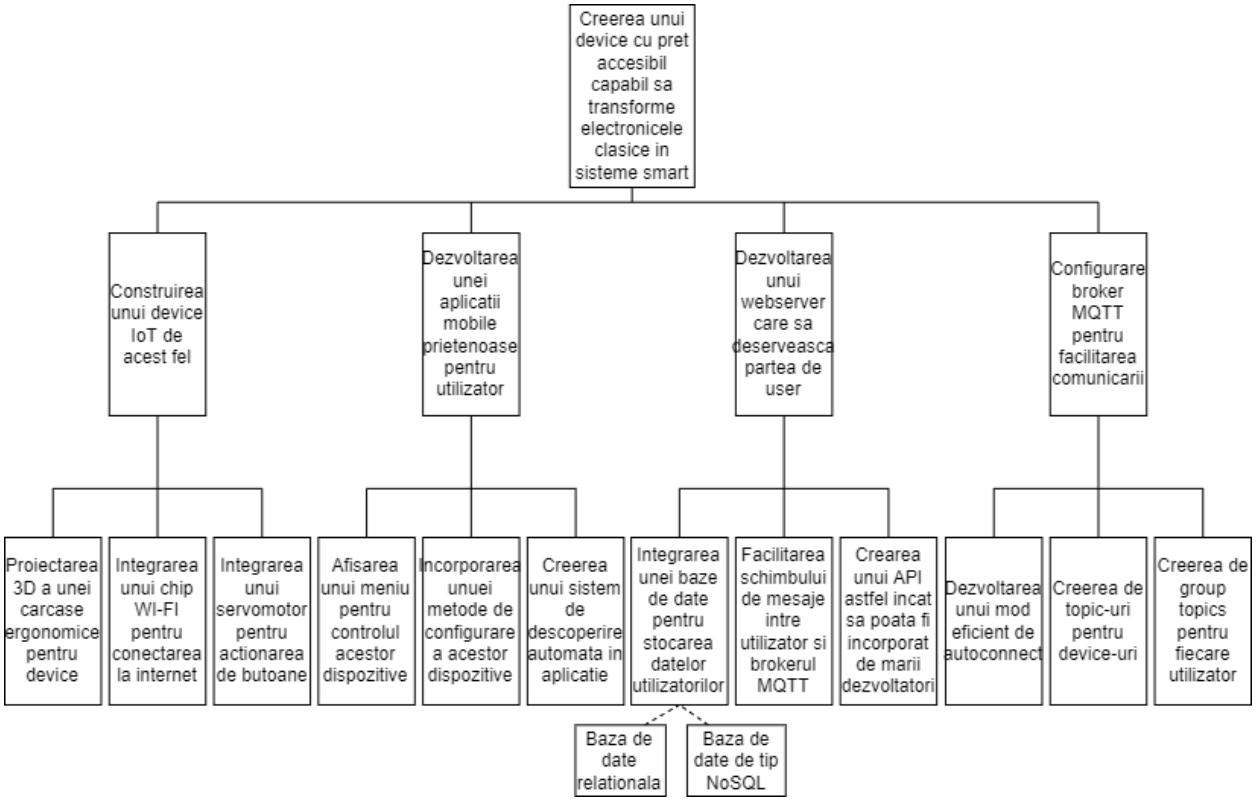

Figura 19. Diagrama descompunerii obiectivelor

Diagrama descompunerii obiectivelor în relație cu cauzele identificate

Astfel, pornind de la problemele care pot fi soluționate, s-a creat o diagramă de obiective (fig. 19). Aceasta este utilizată pentru a corela problematicile identificate cu soluții concrete care pot fi implementate pe parcursul dezvoltării proiectului.

Obiectivul principal este dezvoltarea unui sistem cu preț accesibil care este capabil să transforme unitățile tradiționale (clădiri de birouri, hale industriale, locuințe personale, instituții) în unități smart.

Obiectivul principal este mai departe descompus în 4 componente identificate ca fiind necesare pentru funcționarea unui astfel de sistem. Aceste componente constau în: dispozitiv, aplicație mobilă, server web și broker MQTT. Fiecare dintre acestea la rândul lor sunt descompuse în câte 3 obiective de dezvoltare particulare, astfel:

- Pentru construirea unui dispozitiv IoT de acest fel în primul rând avem nevoie de o carcasă. Carcasa are atât rolul de a proteja dispozitivele electronice, de a putea fi cu ușurință instalată pe întrerupătoare datorită formei accesibile, dar și rol în design, având un estetic plăcut și ergonomic. Pentru a atinge aceste ținte, s-a ales utilizarea tehnologiilor 3D, atât de proiectare, cât și de printare. Pentru ca dispozitivul să poată fi controlat de oriunde există conectivitate, acesta trebuie conectat la rețeaua globală de internet. Pentru a realiza acest lucru, obiectivul este integrarea unui microcip Wi-Fi care să poată fi configurat pentru rețeaua locală de internet. Nu în ultimul rând, pentru partea de acțiune fizică a întrerupătoarelor, este necesar un sistem mecanic de tip piston. Pentru îndeplinirea acestui obiectiv, soluția sugerată este integrarea unui servomotor cu transmisie către un corp rigid care se deplasează pe o șină.

- În cazul dezvoltării unei aplicații mobile cu ajutorul căreia utilizatorul să poată utiliza dispozitivele cu ușurință, în primul rând este nevoie de un meniu de configurare în care utilizatorii să poată inițializa dispozitive, oferind date despre acestea. În continuare, este necesară dezvoltarea unui meniu cu interfața intuitivă de unde utilizatorul să poată trimite instrucțiuni către aceste dispozitive, iar, pentru ca dispozitivele să poată figura ca entități în aplicație, este nevoie de o componentă adițională care comunică cu acestea, descoperindu-le automat în meniul utilizatorului.

- Se dorește dezvoltarea unui web server pentru comunicarea între client și dispozitiv. Cu toate că dispozitivele au propria adresă IP, se dorește crearea unui web server întrucât acesta oferă o mai mare gamă de posibilități. Dorim ca serverul web să funcționeze ca un API de tip

REST pentru ca acesta să poată accepta cereri HTTP atât de la aplicația mobilă, cât și de la aplicații/servicii auxiliare (de tipul Asistenților inteligenți Amazon Alexa, Google Assistant, Apple Siri sau Microsoft Cortana), făcând întregul sistem foarte integrabil. Serverul va avea în administrare o bază de date pentru a ține evidența dispozitivelor, aducând utilitate atât administratorilor de sistem cât și utilizatorilor. Nu în ultimul rând, ca obiectiv se dorește dezvoltarea unei componente care să monitorizeze schimburile de mesaje pe protocolul MQTT între dispozitive și aplicație, procesându-le pe acestea urmând să le redirecționeze.

▪ Pentru a evita relațiile de tip many-to-many, se dorește configurarea unui broker MQTT care să le gestioneze pe acestea mai eficient. În cadrul acestuia se dorește dezvoltarea unor canale individuale pentru fiecare dispozitiv, la care serverul web să fie abonat, primind și publicând mesaje pe acestea. Atunci când un dispozitiv este repornit, se dorește reconectarea automată la sistem. Acest lucru se face prin compunerea unui mesaj care să conțină detaliile acestuia, publicat pe canalul MQTT.

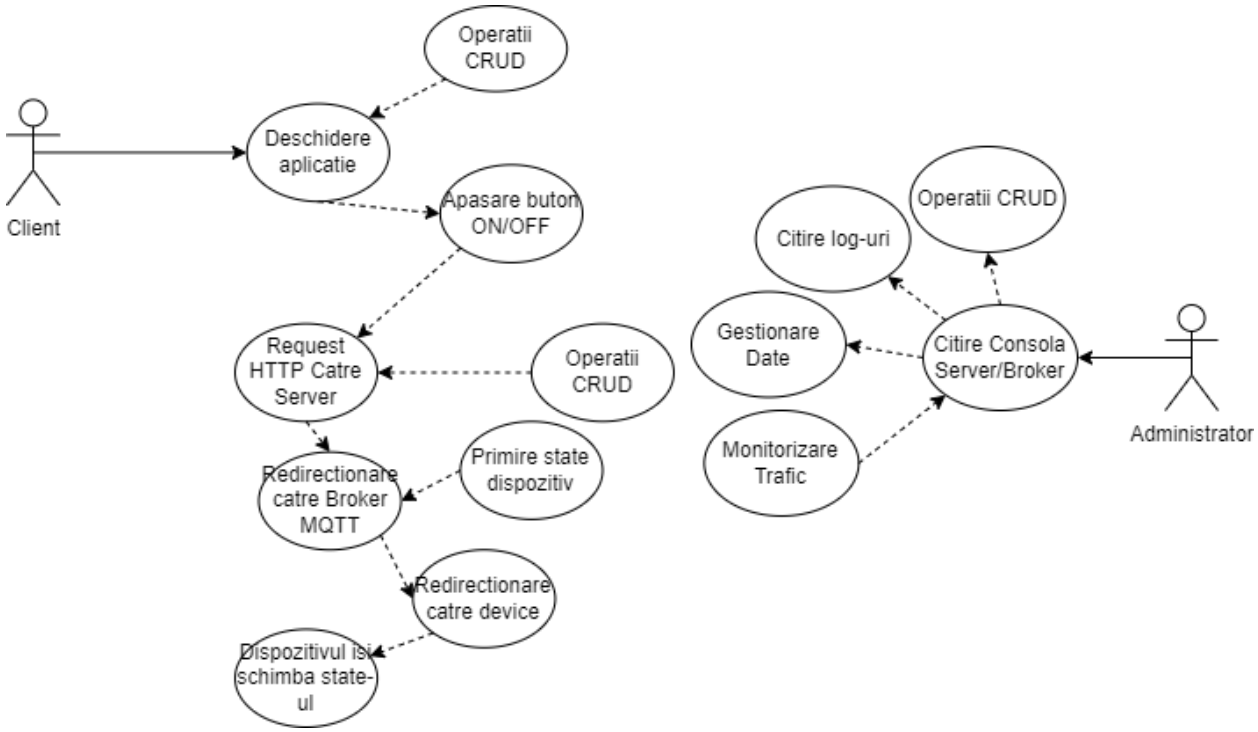

Figura 20. Diagrama de funcționalitate

Diagrama de funcționalitate a sistemului care evidențiază aportul agenților în sistem

După clarificarea obiectivelor de dezvoltare a sistemului, este nevoie sa vizualizăm aportul agenților externi, motiv pentru care s-a realizat diagrama de funcționalitate (fig. 20).

În aceasta se pot observa clientul, care interacționează cu aplicația mobilă, și administratorul de sistem, care interacționează cu sistemul.

Clientul dorește să își automatizeze sistemul electric, motiv pentru care îi este pus la dispoziție un meniu. În momentul în care acesta deschide meniul, prin intermediul bazei de date, acestuia îi sunt listate dispozitivele disponibile. Clientul are posibilitatea de a seta un dispozitiv în mod ON sau OFF. Odată ce clientul a selectat o opțiune, aplicația declanșează un lanț de acțiuni, trimițând o cerere HTTP către server. Acesta procesează cererea și publică un mesaj pe un canal din cadrul brokerului MQTT. O dată postat acest mesaj, dispozitivul abonat la respectivul canal va citi cererea, urmând să își schimbe statusul.

Administratorul de sistem nu interacționează în mod direct cu clientul. Acesta are rolul de a monitoriza terminalele sistemelor aflate în administrarea companiei, scopul lui fiind să depisteze erori sau viitoare oportunități de dezvoltare, asigurându-se că totul funcționează bine, asigurând astfel satisfacția în utilizare pentru client.

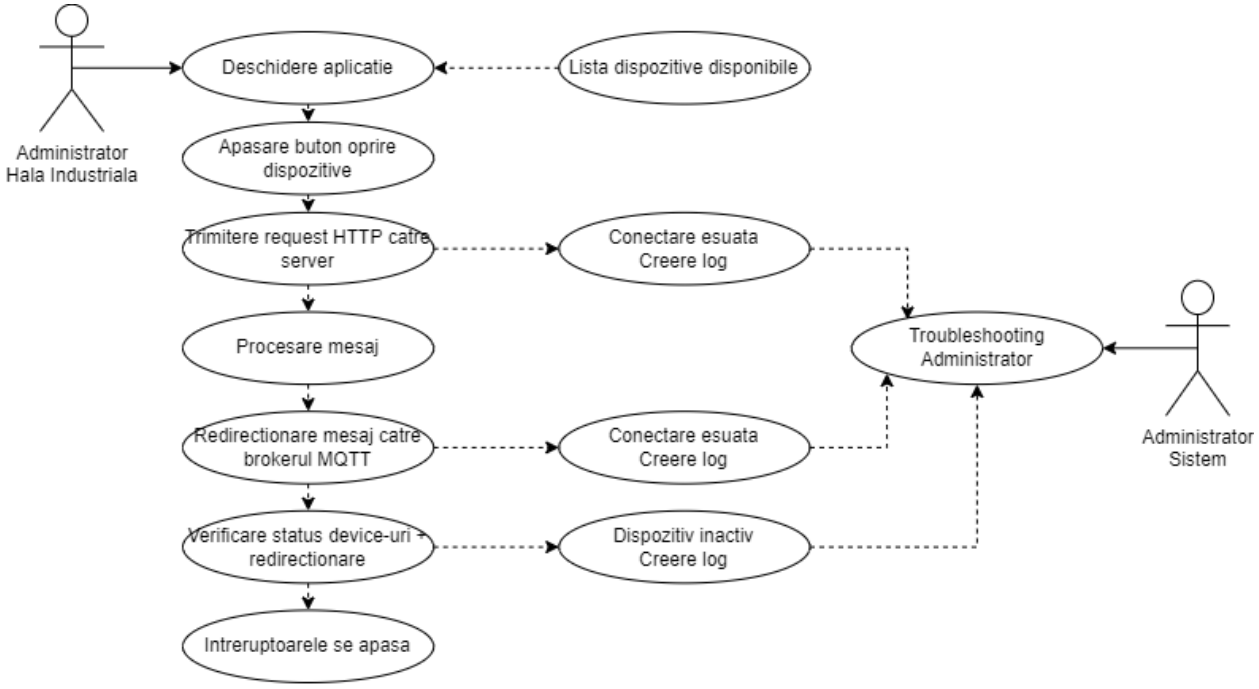

Figura 21. Diagrama caz de utilizare

Diagrama prezintă un caz concret de utilizare al sistemului dezvoltat

Pentru ca funcționalitatea aplicației să poată fi simulată într-un cadru real în care se poate vedea întregul proces în raport cu actorii, s-a realizat o diagramă a cazurilor de utilizare.

În cazul prezentat, este vorba de un administrator de hală industrială care dorește să stingă luminile la sfârșitul zilei de lucru. Acesta are atașate dispozitive pe întrerupătoarele de lumini. El își deschide aplicația de control, moment în care pe ecran este încărcată o listă cu toate dispozitivele disponibile. Acesta acționează butonul de oprire, moment în care aplicația va trimite o cerere HTTP serverului web. În cazul în care cererea nu a putut fi primită, se va genera un log de eroare în consola serverului care va conține codul de eroare și metadatele acesteia (oră, expeditor, etc.)

În cazul în care cererea a fost primită cu succes, serverul o va procesa, iar, în funcție de rezultatul procesării, acesta o va posta pe topicul potrivit de MQTT cu detaliile corespunzătoare. În cazul în care postarea nu a reușit, se va genera un log de eroare în terminalul brokerului MQTT.

În cazul în care mesajul a fost postat cu succes, dispozitivul abonat trebuie la rândul lui să posteze un mesaj de confirmare. Dacă acesta nu postează mesajul de primire înapoi pe topic, se va genera de asemenea un log de eroare.

Administratorul de sistem va avea acces la toate aceste console și va încerca sa găsească soluții și să ofere asistență pentru a remedia eventualele erori.

De cele mai multe ori nu va fi nevoie de intervenția administratorului de sistem, cazul de față luând în considerare toate posibilitățile. Cazul poate fi adaptat în aceeași manieră atât la alte sisteme din hale industriale, cât și la locuințe, birouri sau instituții.

Un atu al dispozitivului dezvoltat este că administratorul de hală industrială poate performa această acțiune atât din interiorul fabricii, cât și din exteriorul ei, și chiar și din confortul casei sale.

Luând în considerare că scenariile pot fi diferite, s-a realizat următoarea diagramă de activități (fig. 22) în care sunt prezentate toate evenimentele posibile care pot avea loc în cadrul sistemului, aceasta putând fi adaptată oricărui tip de situație.

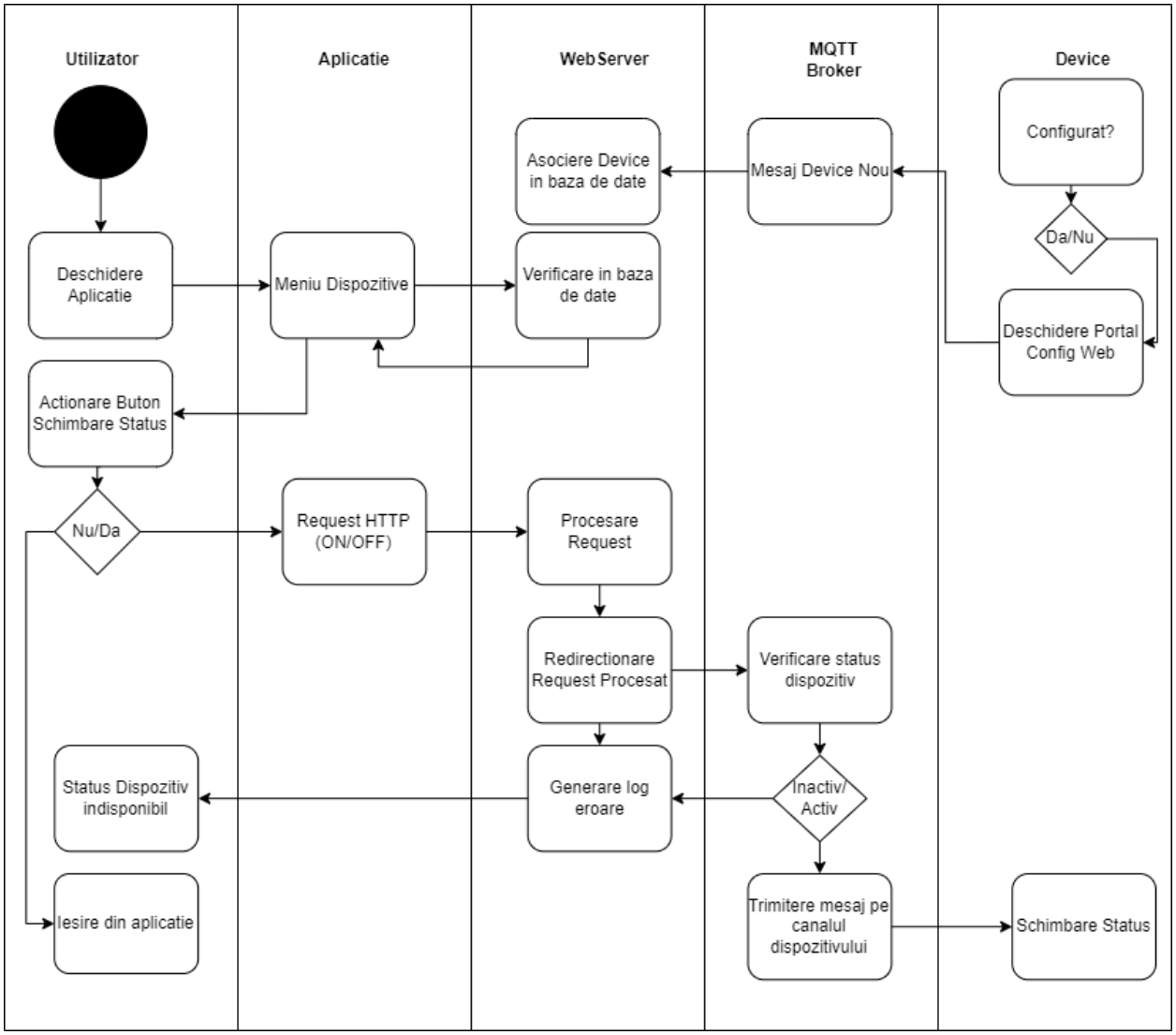

Figura 22. Diagrama de activități

Diagrama de activități prezintă fiecare eveniment care poate avea loc în cadrul sistemului

Întrucât sistemul este alcătuit din mai multe entități individuale care comunică unele cu altele, diagrama a fost segmentată în 5 coloane, aferente fiecărei entități.

Fiecare dintre aceste entități are posibilitatea de a comunica cu entitățile adiacente. Astfel, prima entitate este reprezentată de utilizator, care poate performa acțiuni în aplicație.

Utilizatorul are rolul de a da instrucțiuni, aplicația redirecționându-le pe acestea drept cereri HTTP către server. Serverul are în administrare o bază de dată asupra căreia se execută

operații CRUD în funcție de cererile primite. Aceste operații fac parte din sarcina de procesare a datelor. El comunică totodată și cu brokerul MQTT, care administrează schimbul de opțiuni între dispozitiv și server, așadar și dispozitivul comunicând cu serverul MQTT.

În cazul în care dispozitivul nu a fost configurat încă, acesta va deschide un portal de configurare a datelor, urmând ca acestea să fie transmise prin intermediul brokerului MQTT către web servere, fiind ulterior introduse în baza de date.

Ca o anexă a acestei diagrame, s-a creat diagrama de stări (fig. 23), care indică modul de funcționare al aplicațiilor în funcție de stare.

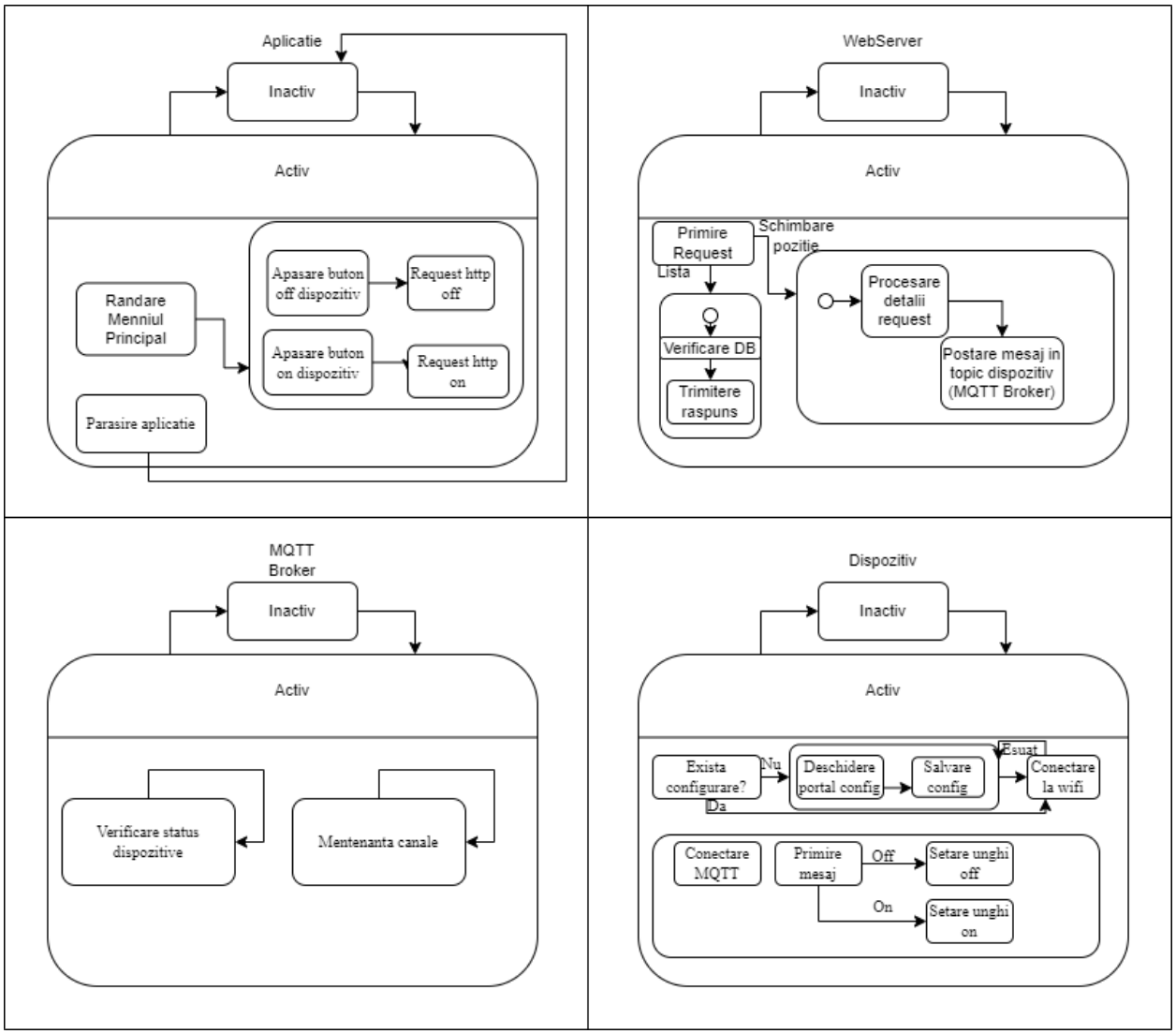

Figura 23. Diagrama de stari

Diagrama de stări prezintă procesele individuale ale aplicațiilor în funcție de starea lor de activitate

Întrucât există 4 aplicații care pot fi în paralel active și inactive, fiecare dintre acestea contribuind la produsul final, diagrama este împărțită în 4 cadrane, fiecare dintre acestea fiind aferent unei aplicații.

Fiecare dintre cele 4 diagrame are o stare de “Activ” și una de “Inactiv”. Singura dintre cele 4 care poate fi oprită de către utilizator fiind aplicația mobilă.

Serverul Web și brokerul MQTT find componente backend, ele în teorie rulează constant, singurele momente de inactivitate fiind când acestea sunt oprite de administrator. Dispozitivul poate fi oprit doar atunci când se deconectează sursa de energie electrică.

3. Proiectarea Sistemului

În cadrul acestui capitol, voi prezenta procesele de proiectare care au rezultat în realizarea sistemului IoT de acționare a întrerupătoarelor.

Pentru a putea eficientiza bunul mers al lucrurilor, evitând pierderea haotică a timpului în contextul în care nu ar fi existat un plan de urmat, s-a realizat un proces de brainstorming în urma căruia s-au luat decizii cu privire la componentele sistemului, arhitectura acestuia precum și ciclul fluxului de date, toate acestea fiind reprezentate grafic prin diagramele aferente în conformitate cu standardele de modelare consacrate.

3.1 Arhitectura sistemului

Arhitectura sistemului software (conform *Ciaca, 2018*) are rolul de a separa structura unui sistem (care este alcătuit din subsisteme și interfețe) de detaliile interne ale subsitemelor individuale.

Astfel, în cazul sistemului de față, s-au identificat mai multe componente (subsisteme) care pot fi încadrate în anumite arhitecturi.

▪ Arhitectura Client-Server "poate fi definită drept o arhitectură software compusă atât din client, cât și din server, în cadrul careia întotdeauna clienții trimit cereri în timp ce serverul răspunde cererilor trimise" (*Haroon Shakirat, 2014*). Acest tip de arhitectura se poate identifica între aplicația mobilă și serverul web, aplicația mobilă fiind utilizată de către client pentru a trimite cereri prin protocolul HTTP către server, acesta trecând cererile printr-o serie de procese interne, în funcție de rezultatul acestora returnând un răspuns către client.

▪ Arhitectura Context Broker sau CoBrA este, conform abordării descrise de *Harry Chen, Tim Finin și Anupam Joshi (2004)*, cuprinde o entitate de tip server care poartă numele de "context broker" care are rolul de a oferi un model centralizat de contexte. Aceste contexte pot fi împărțite între dispozitive și servicii. Un astfel de model de arhitectură se regăsește în sistemul dat, întrucât serverul comunică cu dispozitivul prin intermediul unui broker de mesagerie care utilizează protocolul MQTT.

▪ Arhitectura Multilayered este (conform *Ciaca, 2018*) o arhitectură de tip client-server în care prezentarea, aplicarea, procesul și data managementul funcțiilor sunt separate fizic. Acest tip de arhitectură este de asemenea prezent în sistem prin intermediul portalului de configurare. Portalul de configurare este lansat pe un server găzduit pe dispozitiv, acesta fiind interfața disponibilă utilizatorului, astfel reprezentând componenta de prezentare. Datele de configurare introduse pe acest portal sunt ulterior stocate în memoria SPIFFS a dispozitivului pentru a putea fi procesate local, dar și transmise către bază de date pentru un management care permite utilizarea acestora și de către alte componente de sistem.

Aceste arhitecturi se pot identifica și vizualiza în diagrama următoare realizată pentru arhitectura sistemului (fig. 25).

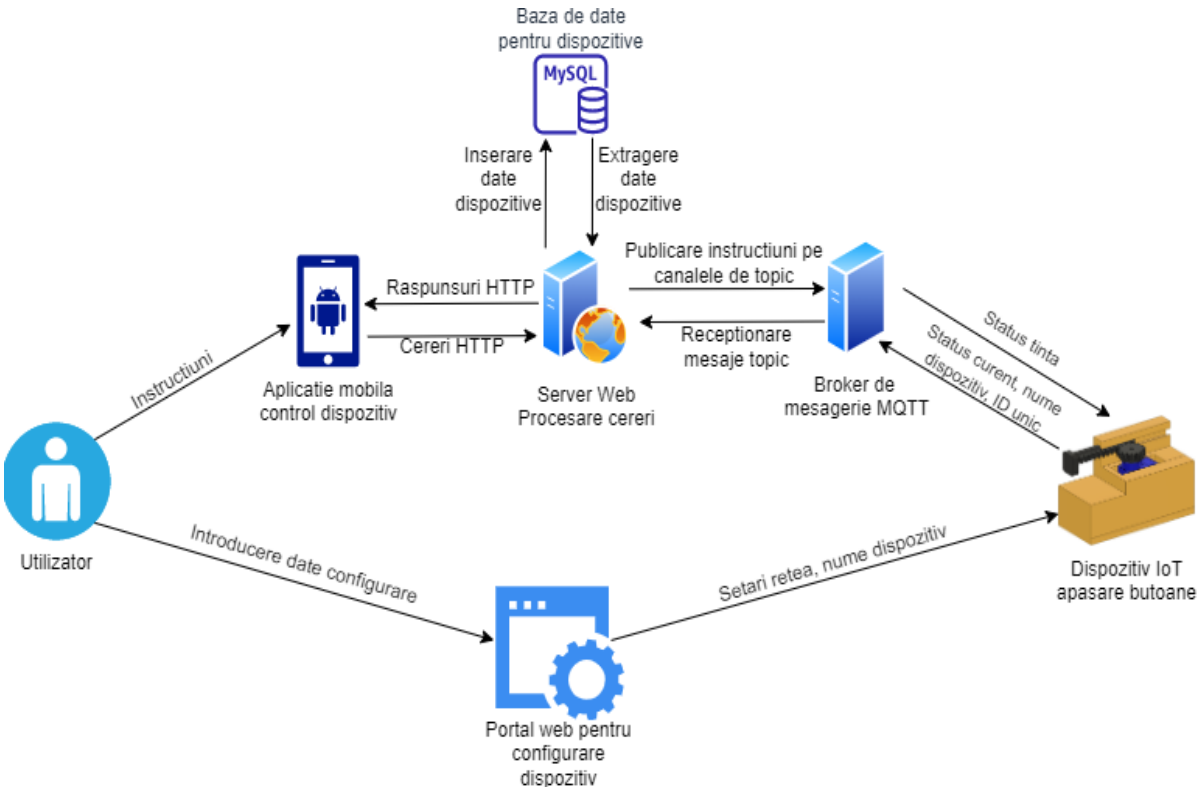

Figura 24. Arhitectura sistemului

Diagrama alăturată reprezintă arhitectura sistemului, conținând componente precum și relațiile dintre acestea

Ca primă componentă a sistemului, îl avem pe utilizator. Acesta, în primul rând, are posibilitatea de a configura dispozitivul prin intermediul portalului web. Detaliile de configurare vor fi salvate în dispozitiv.

Totodată, utilizatorul are acces la aplicația mobilă. În funcție de acțiunile pe care le execută în cadrul aplicației, aceasta va transmite cereri HTTP către server, fiindu-i returnate răspunsuri.

Serverul web se ocupă de procesarea datelor care intră în acesta, el comunicând totodată cu baza de date MySQL.

Acesta, mai departe pentru a cere / transmite informații de la / către dispozitiv, comunică cu brokerul de mesagerie MQTT. Serverul poate posta mesaje pe canalele de comunicare, ascultând totodată răspunsurile dispozitivelor.

Brokerul MQTT creează un canal de comunicare pentru fiecare dintre dispozitive astfel încât schimbul de date să se poată realiza individual.

Dispozitivul IoT, în funcție de informațiile primite pe canalul MQTT, va executa comenzile primite, returnând ulterior răspunsuri pe canal.

Aceste schimburi de informații pot fi identificate și vizualizate mai clar pe diagrama fluxului de date.

3.2 Flux de date

Diagrama fluxului de date prezintă modul în care datele călătoresc în cadrul sistemului, de la o componentă la alta.

Cu toate că în cadrul sistemului există o singură bază de date relațională, care cuprinde un singur tabel de înregistrări, între componentele sistemului se transmit și se procesează date sub diverse alte formate.

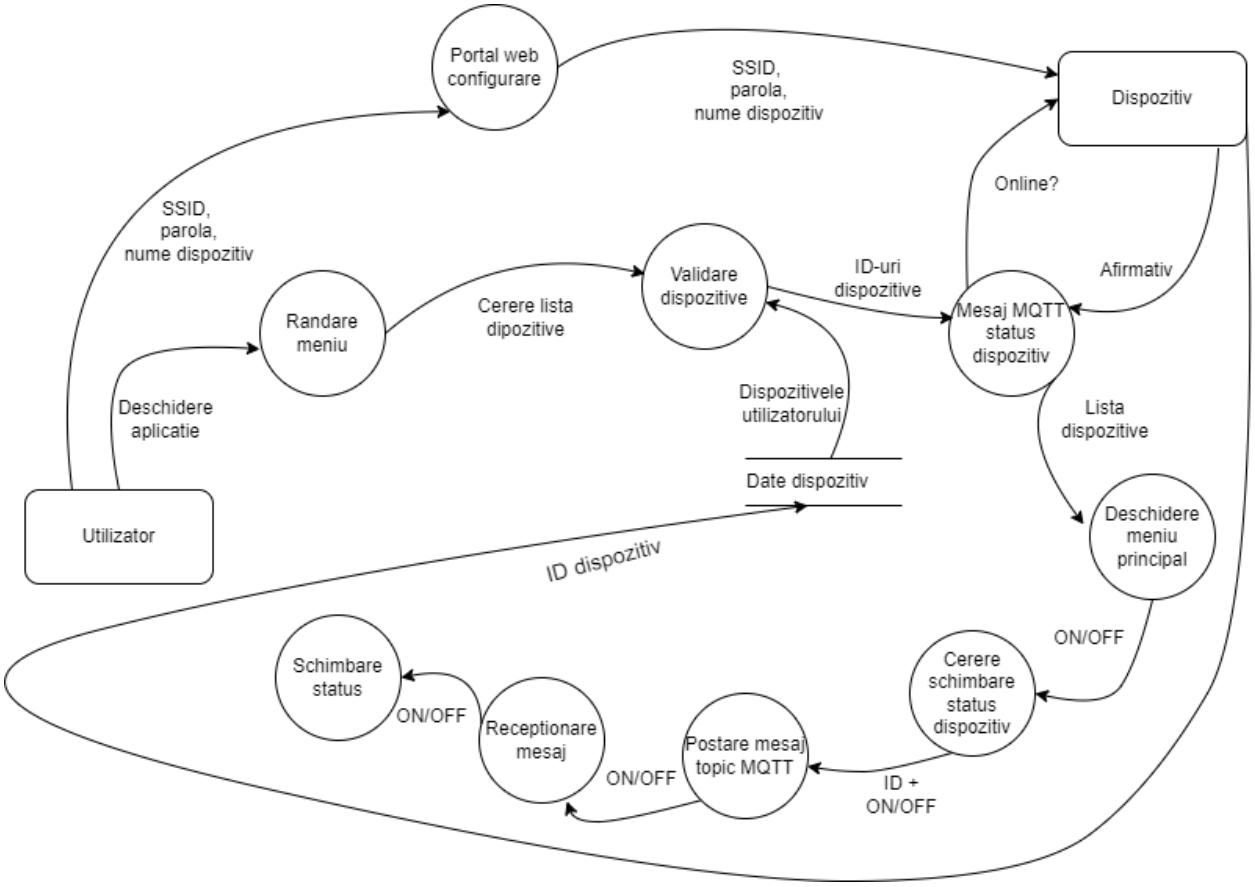

Figura 25. Digrama flux de date

Diagrama prezintă traseul datelor în interiorul sistemului informatic

Fluxul de date începe de la utilizator. Așa cum a fost prezentat și în arhitectura sistemului, în acest punct apare o bifurcație, în funcție de acțiunea pe care o execută utilizatorul. Acesta poate accesa fie portalul de configurare, unde va introduce datele legate de rețea (SSID-ul și parola rețelei wireless, cât și un nume pentru dispozitiv).

O dată ce datele din portalul web au fost completate, acestea sunt mai departe transmise către memoria internă a dispozitivului, unde urmează a fi salvate în format JSON, urmând ca acesta să își transmită id-ul către bază de date, prin intermediul protocolului MQTT.

Utilizatorul poate genera un flux de date și prin intermediul aplicației mobile, astfel că, în momentul lansării acesteia se va deschide un meniu pe care urmează a fi afișată lista dispozitivelor. Va porni un proces de validare în care, prin intermediul brokerului MQTT se vor

lansa mesaje către dispozitive, urmând ca acestea să își returneze ID-ul către baza de date în cazul în care sunt active. După aceea, meniul principal al dispozitivelor active va fi randat.

În acest meniu, utilizatorul are posibilitatea de a trimite mesaje de schimbare a statusului (ON/OFF) pentru fiecare dintre acestea. Aceste mesaje sunt transmise către server sub formă de cerere HTTP, conținând statusul țintă, cât și ID-ul dispozitivului în cauză.

Serverul HTTP va procesa acest mesaj, urmând să îl încarce pe canalul MQTT al dispozitivului, de unde acesta poate fi preluat de dispozitivul care este abonat la acel canal. Acest ciclu de date are ca final schimbarea statusului dispozitivului.

3.3 Componente

Fiecare dintre elementele individuale care alcătuiesc sistemul (baza de date MySQL, aplicația mobilă, serverul web, brokerul MQTT HiveMQ și dispozitivul) sunt alcătuite la rândul lor din mai multe subcomponente care fac posibil întregul proces la nivel de subsistem.

Astfel, pentru a înțelege pe deplin sistemul, s-a realizat diagrama de componente (fig. 26) care ne ajută să vizualizăm compoziția fiecărui subsistem.

Aplicația mobilă, la care utilizatorul are acces, cuprinde o componentă ascunsă destinată autodescoperirii dispozitivelor și 3 componente vizuale cu care utilizatorul poate interacționa (lista dispozitivelor, butoanele ON și butoanele OFF), acestea fiind interdependente. Aceasta comunică cu serverul web care cuprinde 3 componente: un procesator de cereri HTTP (pentru intrare și ieșire), un monitor pentru canalele MQTT, cât și un manager pentru baza de date,

Bază de date comunică doar cu serverul web, având o componentă unică, și anume tabelul cu datele dispozitivelor.

Serverul comunică totodată și cu brokerul MQTT, acesta având unicul rol (precum și componenta aferentă) de a manageria canalele de comunicare dintre server și dispozitive. Unul sau mai multe dispozitive sunt conectate la broker, care la rândul lor includ o componentă pentru managerierea și monitorizarea canalelor, dar și pentru menținerea conectivității, controlerul pentru servomotor și portalul de configurare web.

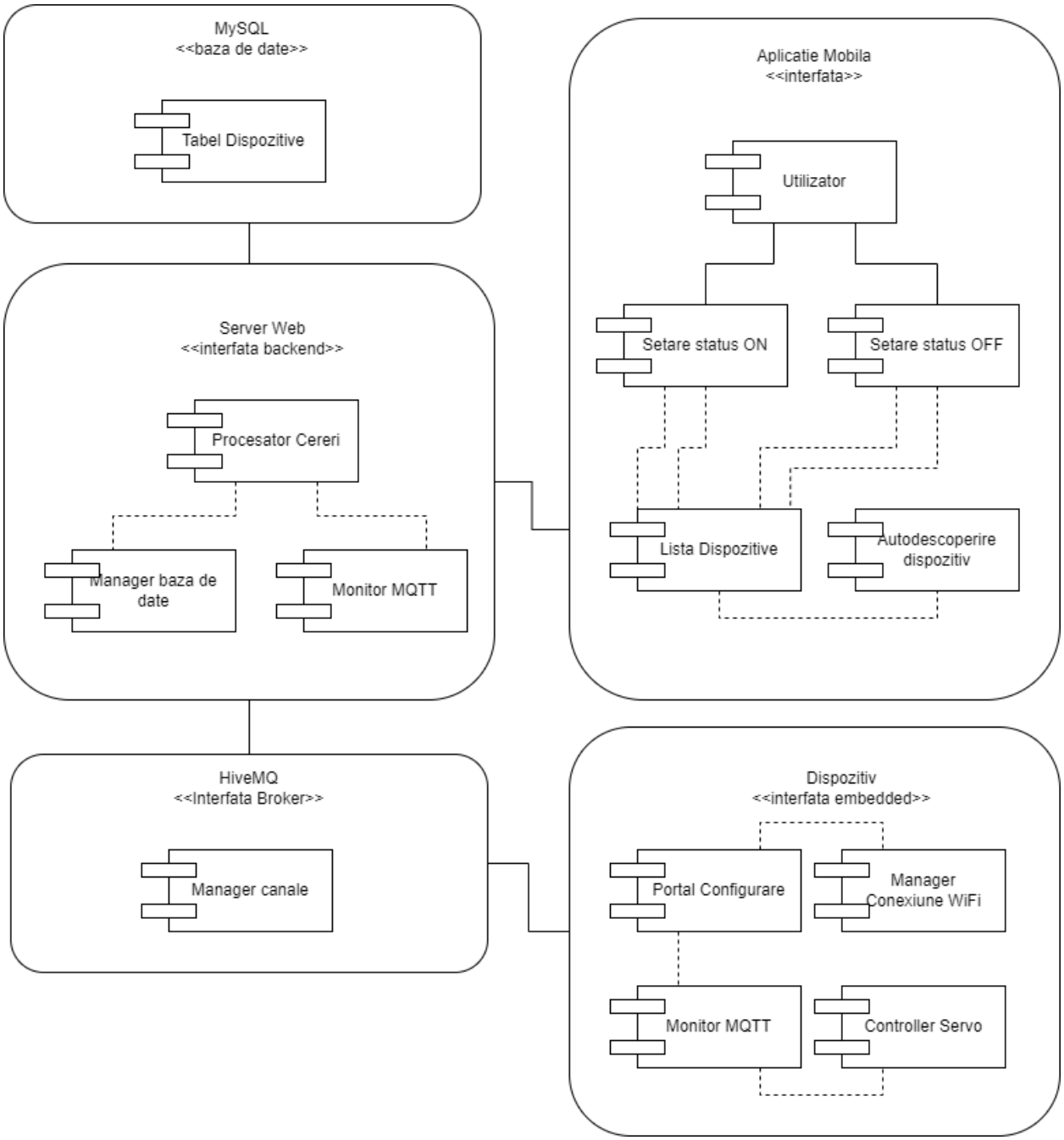

Figura 26. Diagrama pe componente

Diagrama prezintă componentele sistemului informatic

4. Implementare

În acest capitol voi prezenta în detaliu procesul de implementare al codurilor sursă, pentru fiecare dintre componentele care alcătuiesc sistemul informatic în cauză (aplicația mobilă, dispozitivul, precum și serverul web) prin intermediul capturilor de ecran cuprinzând fragmente de cod sursă și al explicațiilor adiționale pe baza acestora.

4.1 Implementare dispozitiv

Pentru partea de dispozitiv, IDE-ul folosit a fost Visual Studio Code, pe care s-a instalat pachetul PlatformIO. Acesta ne-a permis să configurăm cu ușurință mediul de lucru pentru dezvoltarea unui dispozitiv IoT, oferind acces la bibliotecile de specialitate, permițându-ne să încărcăm codul pe plăcuță de dezvoltare prin intermediul porturilor USB. Orice proiect PlatformIO cuprinde un fișier de inițializare numit platform.ini.

```
[env:nodemcuv2]
platform = espressif8266
board = nodemcuv2
framework = arduino
monitor_speed = 115200
upload_port = COM3
lib_deps =
    bblanchon/ArduinoJson@5.13.4
    knolleary/PubSubClient@^2.8
    tzapu/WiFiManager@^0.16.0
```

Pe prima linie, se definește mediul de lucru pentru care se va aplica protocolul definit. Se utilizează o placă de dezvoltare NodeMCU, corespondentul acesteia în PlatformIO fiind nodemcuv2 (în conformitate cu standardele platformio.org, 2014). Pe liniile următoare

specificăm ce platformă folosim (pentru modulul ESP8266 aceasta fiind espressif8266), modelul de placă folosită, framework-ul de dezvoltare care se folosește pentru placa respectivă (în cazul de față, framework-ul Arduino fiind compatibil, rata cu care se citește/scrie informația de pe/pe porturile USB. Cu toate că recomandarea producătorului este 9600 biți / secundă, am decis să utilizez 115200 biți / secundă, întrucât această rată este singura care returna text descifrabil.

Mai departe se specifică portul USB pentru încărcarea codului, acesta putând fi aflat din Device Manager. Nu în ultimul rând, se specifică bibliotecile utilizate în codul sursă, PlatformIO incluzându-le automat conform versiunii specificate. În cazul ne față, aceste biblioteci sunt ArduinoJSON pentru operarea fișierelor JSON, PubSubClient pentru utilizarea funcționalităților MQTT, respectiv WiFiManager pentru portalul web și setările de rețea WiFi.

Pentru a fi mai ușor de operat, codul este structurat în 4 componente (mqtt, log, controls, autodiscovery), fiecare compuse din câte un fișier sursă (.cpp) și un fișier header (.hpp). În fișierele header sunt definite anumite variabile și funcții care se utilizează pentru componentele respective în cadrul fișierelor sursă.

Placă de dezvoltare rulează în mod implicit fișierul main.cpp. Acesta va conține o funcție pentru inițializare (“setup”) și o funcție pentru rulare continuă (“loop”). Pe parcursul rulării fișierului main.cpp, se inițializează fiecare componentă.

4.1.1 Componenta log

Componența log are rolul de a afișa mesaje în Serial Monitor. Conform *platformio.org 2014*, Serial Monitor este o aplicație de tip terminal care comunică cu un anume port USB. Astfel, de pe portul USB se pot citii informații care vin de la placa de dezvoltare în cazul în care aceasta este conectată la computer, componenta log fiind utilizată strict pentru generarea de informații care ajută în procesul de debug.

```
16  void log_init() {
17      Serial.begin(SERIAL_BAUD_RATE);
18      pinMode(LOG_ENABLE_DEBUG_PIN, INPUT);
19      int enable_debug = digitalRead(LOG_ENABLE_DEBUG_PIN);
20      if (enable_debug) {
21          _log_debug = log_debug_impl;
22      } else {
23          _log_debug = log_debug_dummy;
24      }
25  }
26
```

```
27  void log_debug(const char * msg) {
28      _log_debug(msg);
29  }
30  void log_info(const char * msg) {
31      log(LOG_LEVEL_INFO, msg);
32  }
33  void log_debug_impl(const char * msg) {
34      log(LOG_LEVEL_DEBUG, msg);
35  }
36  void log_debug_dummy(const char * msg) {
37  }
38  void log(const char * level, const char * msg) {
39      static char buf[1024];
40      sprintf(buf, "[%s] %s\n", level, msg);
41      Serial.print(buf);
42  }
```

În cadrul funcției log_init, pe linia 17 se lansează conexiunea la serial. Pe linia 18 se inițializează pin-ul pentru debug, pe linia 19 se citește valoarea de pe acest pin iar în următoarea structură de tip “if” se decide în funcție de valoarea acestuia dacă se va porni sau nu modul de debug.

Funcția log_debug vă va apela funcția _log_debug după ce i s-a setat valoarea în cadrul log_init. Funcția log_info are rolul de a afișa un log în Serial Monitor.

Funcțiile log_debig_impl și log_debug_dummy sunt valorile pe care le poate lua funcția _log_debug în funcție de valoarea citită de pe pin, acestea afișând sau nu mesaje de debug.

Funcția log are rolul de a salva mesajul care trebuie afișat într-un buffer pentru a putea fi formatat. După formatare, acesta este afișat sub formă de log în Serial Monitor.

4.1.2 Componenta mqtt

Componeta mqtt encapsulează clientul de MQTT și pune la dispoziție câteva funcții pentru a facilita interacțiunea cu acesta. De asemenea această componentă permite altor componente să se cupleze în procesul de procesare al mesajelor primite.

```
#define MQTT_SERVER_IP ((const char *)("192.168.1.193"))
#define MQTT_SERVER_PORT ((uint16_t)(1883))
#define MQTT_MESSAGE_CONSUMED ((uint8_t)(0))
#define MQTT_MESSAGE_NOT_CONSUMED ((uint8_t)(1))

typedef int (*mqtt_consume_t)(const char *topic, const byte *payload, unsigned int len);

void mqtt_init(const char * client_id);
void mqtt_register_consumer(mqtt_consume_t consumer);
void mqtt_loop();
void mqtt_publish(const char * topic, const char * payload);
void mqtt_subcribe(const char * topic);
void mqtt_unsubscribe(const char * topic);
```

Pe primele 4 linii din cadrul fișierului header “mqtt.hpp” se regăsesc definite câmpurile necesare pentru conectarea la un server MQTT, și anume IP-ul, portul precum două definiții folosite de consumatori pentru ca această componentă să își dea seama dacă mesajul a ajuns la consumatorul destinație sau nu.

După aceea, se definește un tip de dată care conține un pointer la o funcție cu semnătura unui consumator, făcând astfel posibilă declararea cu ușurință a unui consumator.

Pe liniile următoare sunt definite antetele funcțiilor care operează procesele în cadrul componentei, și anume mqtt_init, mqtt_register_consumer, mqtt_loop, mqtt_publish, mqtt_subscribe și mqtt_unsubscribe.

```
WiFiClient wifi_client;
PubSubClient client(wifi_client);

#define CONSUMER_LIST_CAPACITY ((uint8_t)(4))
mqtt_consume_t consumer_list[CONSUMER_LIST_CAPACITY];
uint8_t consumer_list_len = 0;

void mqtt_callback(char * topic, byte *payload, unsigned int len);
```

La începutul fișierului mqtt.cpp declarăm un WiFiClient, care este un obiect din cadrul librăriei WiFiClient.h care facilitează comunicarea în rețea. În continuare se definește clientul de mqtt care ia ca parametru de intrare clientul de rețea wifi_client definit anterior. În cadrul WiFiClient se regăsesc pachetele care au ajuns de la rețeaua de internet la dispozitiv. Prin pasarea

acestuia la componenta PubSubClient, componenta MQTT a dispozitivului primește acces la resursele de internet.

Acest proces se poate asocia cu modelul OSI care, conform Costa, 1998 este un model standard pentru rețelele de telecomunicație, care este independent de sistem și de protocolul de comunicare, structurat sub formă de stivă, pe 7 nivele. Fiecare nivel reprezintă o sub-sarcină care este manageriată în procesul de telecomunicație. Astfel, în cazul de față componentă WiFiClient se regăsește pe nivelul de rețea (Network Layer) acesta având rolul să preia informațiile din rețea. Componenta PubSubClient poate fi identificată drept nivelul de aplicație (Application Layer) aceasta fiind interfața căreia i se permite accesul și se folosește de resursele din rețea. Nivelele intermediare dintre acestea sunt reprezentate de componentele interne ale librăriilor WiFiClient.h respectiv PubSubClient.h.

Pentru a eficientiza sistemul de mesagerie, pe liniile următoare se definesc elementele listei de consumatori, și anume vectorul de consumatori, capacitatea acestuia și lungimea lui.

Pe linia de după este declararea funcției MQTT callback care primește mesaje pe mqtt. Acesta nu a fost declarat în header întrucât nu este folosit și în exterior.

```
void mqtt_init(const char * client_id) {
    client.setServer(MQTT_SERVER_IP, MQTT_SERVER_PORT);
    client.setCallback(mqtt_callback);
    if (!client.connect(client_id)) {
        log_info("mqtt connection failed, please reset the board");
        while (true) {}
    }
    log_debug("initialized mqtt");
}
void mqtt_register_consumer(mqtt_consume_t consumer) {
    if (consumer_list_len + 1 > CONSUMER_LIST_CAPACITY) {
        log_info("mqtt consumer list overflowed");
        while (true) {}
    }
    consumer_list[consumer_list_len] = consumer;
    consumer_list_len += 1;
}
```

Funcția mqtt_init are rolul de a realiza conexiunea la serverul MQTT. Această primește ca parametru id-ul clientului, după care setează serverul precum și callback-ul. În continuare, funcția încearcă să conecteze clientul la serverul mqtt, țar dacă conexiunea eșuează, se va afișa un log în monitorul de serial, cerând utilizatorului să reseteze placa. Dacă conexiunea reușește, va fi afișat un log care să ateste acest lucru.

Funcția mqtt_register_consumer primește ca parametru un consumator pentru a fi adăugat în listă. Dacă lista se afla la capacitate maximă va fi afișat un log care să anunțe acest lucru, iar în caz contrar, consumatorul va fi adăugat în vector iar lungimea liste se va incrementa cu o unitate.

```
void mqtt_loop() {
    client.loop();
}
void mqtt_publish(const char * topic, const char * payload) {
    client.publish(topic, payload);
}
void mqtt_callback(char * topic, byte *payload, unsigned int len) {
    for (uint8_t i = 0; i < consumer_list_len; i++) {
        mqtt_consume_t consumer = consumer_list[i];
        if (consumer(topic, payload, len) == MQTT_MESSAGE_CONSUMED) {
            return;
        }
    }
    log_debug("no consumers consumed this message");
}
void mqtt_subcribe(const char * topic) {
    client.subscribe(topic);
}
void mqtt_unsubscribe(const char * topic) {
    client.unsubscribe(topic);
}
```

Funcția MQTT loop are rolul de a verifica dacă există pachete de date în rețea care au ajuns de la clientul MQTT și de a le procesa pe acestea. mqtt_publish are unicul rol de a posta un mesaj pe un topic.

Funcția mqtt_calback are rolul de a verifica lista de consumatori pentru a vedea cui i se adresează mesajul din topic, iar în momentul în care s-a găsit o potrivire, mesajul este consumat de către consumator. Dacă nu s-a găsit nici o potrivire, se va afișa un log menționând acest lucru.

Funcțiile MQTT subscribe și unsubscribe au rolul de a abona/dezabona clientul la/de la anumite topicuri.

4.1.3 Componenta controls

Componenta controls are rolul de a controla servomotorul.

```
#define CONTROLS_SERVO_PIN (D6)
#define CONTROLS_ANGLE_MIN 0
#define CONTROLS_ANGLE_MAX 180

void controls_init();
void controls_set_control_mqtt_topic(const char * topic);
void controls_set_angle(int angle);
```

În fișierul header controls.hpp se definește pinul că care este conectat servomotorul, precum și unghiul minim și cel maxim. Se definesc funcțiile controls_init care are rolul de a inițializa componentele de control, controls_set_control_mqtt_topic care setează topicul pe care se primesc controale pentru servomotor (unghiuri), respectiv controls_set_angle care schimbă unghiul servomotorului.

```
void controls_init() {
    servo.attach(CONTROLS_SERVO_PIN);
    mqtt_register_consumer(controls_mqtt_consume);
    log_info("initialized controls");
}
void controls_set_control_mqtt_topic(const char * topic) {
    mqtt_subcribe(topic);
    if (controls_mqtt_topic != NULL) {
        mqtt_unsubscribe(controls_mqtt_topic);
    }
    controls_mqtt_topic = topic;
}
```

În fișierul controls.cpp regăsim funcția controls_init. Aceasta are rolul de a atașa servomotorul la pinul destinat pentru transmiterea semnalelor pwm care acesta, după care îl înregistrează drept consumator în lista de consumatori. Când procesul este gata, un log este afișat.

Funcția controls_set_control_mqtt_topic are rolul de a abona dispozitivul la topicul MQTT pentru transmiterea mesajelor de control.

```
int controls_mqtt_consume(const char *topic, const byte *payload, unsigned int len) {
    static char buf[12];
    if (strcmp(topic, controls_mqtt_topic)) {
        log_debug("not a control message");
        return MQTT_MESSAGE_NOT_CONSUMED;
    }
    if (len == 0) {
        log_debug("empty control message");
        return MQTT_MESSAGE_CONSUMED;
    }
    memcpy(buf, payload, len * sizeof(char));
    int angle = atoi(buf);
    if (angle < CONTROLS_ANGLE_MIN || angle > CONTROLS_ANGLE_MAX) {
        log_debug("illegal control message");
        return MQTT_MESSAGE_CONSUMED;
    }
    servo.write(angle);
    return MQTT_MESSAGE_CONSUMED;
}
```

Funcția controls_mqtt_consume are rolul de a verifica dacă o funcție este destinată controlului pentru servomotor. Aceasta verifică în primul rând dacă topicul ținta este diferit de topicul de control, caz în care afișează un log. În continuare verifică dacă conținutul este gol, afișând de asemenea un log. Dacă mesajul a trecut cu succes de aceste două condiții, înseamnă că este un mesaj de control valid, caz în care este copiat într-un buffer. Buffer-ul este convertit într-un întreg, întrucât el reprezintă unghiul ținta la care servomotorul trebuie să ajungă. Dacă acesta se află în afara parametrilor minim și maxim definiți anterior, atunci unghiul țintă este invalid, caz în care se va afișa un loc atestând acest lucru, mesajul fiind consumat. Dacă unghiul este valid, acesta se va scrie în servo, dispozitivul modificându-și poziția, mesajul consumându-se.

4.1.4 Componenta autodiscovery

Componența autodiscovery are rolul de a posta un mesaj pe MQTT care conține ID-ul și numele dispozitivului. Acest mesaj este folosit de aplicația android pentru a genera lista dispozitivelor disponibile, și de server pentru a actualiza baza de date.

```
#define AUTODISCOVERY_TOPIC ((const char *)("connected-device"))
#define AUTODISCOVERY_PAYLOAD_SEPARATOR ((const char *)(";"))

void autodiscovery_init();
void autodiscovery_send_notice(const char * device_id, const char * device_name);
```

În fișierul header se definește topicul utilizat pentru autodiscovery, împreună cu un separator care este folosit pentru diferențierea câmpurilor din mesajul postat. Se definesc funcțiile autodiscovery_init pentru inițializare și autodiscovery_send_notice pentru transmiterea mesajului pe topic.

```
void autodiscovery_init() {
    log_debug("initialized autodiscovery");
}
void autodiscovery_send_notice(const char * device_id, const char * device_name) {
    static char buf[256];
    sprintf(buf, "%s%s%s", device_id, AUTODISCOVERY_PAYLOAD_SEPARATOR, device_name);
    mqtt_publish(AUTODISCOVERY_TOPIC, buf);
    log_debug("sent autodiscovery payload");
}
```

Funcția autodiscoery_init este utilizată în procesul de debug, aceasta lăsând o notă în consolă care marchează începerea procesului.

Funcția autodiscovery_send notice creează un buffer în care se concatenează id-ul dispozitivului, separatorul definit și numele dispozitivului. Acest mesaj este publicat pe topic-ul de autodiscovery, afișându-se la urmă un log.

4.1.5 Fisierul main.cpp

Fisierul main este fisierul care se rulează implicit atunci cand placa este conectata la curent electric. Acesta are doua funcții principale, și anume funcția setup și funcția loop.

```
String device_id = ("pusher-" + String(random(0x13af), HEX));
const char * device_ap_password = "Password1234.";
void setup() {
    pinMode(LED_BUILTIN, OUTPUT);
    digitalWrite(LED_BUILTIN, HIGH);
    log_init();
    log_info("setup started");
    WiFiManager wifiManager;
    WiFiManagerParameter custom_name("name", "device name", "Wi-Fi Pusher", 30);
    wifiManager.addParameter(&custom_name);
    if (false) {
        if (!wifiManager.startConfigPortal("Configure wi-fi pusher", device_ap_password)) {
            log_info("failed to connect and hit timeout, resetting in 3 seconds..");
            delay(3000);
            ESP.reset();
        }
    } else if (!wifiManager.autoConnect("Configure wi-fi pusher", device_ap_password)) {
        log_info("failed to connect and hit timeout, resetting in 3 seconds..");
        delay(3000);
        ESP.reset();
    }
    mqtt_init(device_id.c_str());
    controls_init();
    controls_set_control_mqtt_topic(device_id.c_str());
    autodiscovery_init();
    autodiscovery_send_notice(device_id.c_str(), custom_name.getValue());
    log_info("setup complete");
    digitalWrite(LED_BUILTIN, LOW);
}
void loop() {
    mqtt_loop();
}
```

Înaintea funcției de setup, se creează un id unic pentru dispozitiv, acesta fiind construit prin concatenarea expresiei “pusher-” și a unui șir aleator de caractere hexadecimal. După aceea, este definită parola pentru punctul de acces al portalului web care se lansează pentru configurare.

În funcția setup, pentru a marca inițierea procesului de setup, led-ul de pe suprafața dispozitivului este inițializat și aprins. După aceea, se inițializează componența de log și se afișează mesajul “setup started”.

Se inițializează un obiect de tip WiFiManager. Acesta pornește portalul de configurare web, stochează credențialele conexiunii wi-fi și manageriază conexiunea. Acestuia i se mai adaugă un parametru pentru a putea introduce numele dispozitivului. După aceea, se va încerca pornirea portalului web. Dacă acest se pornește, utilizatorul trebuie să îl acceseze și să introducă datele de configurare. Dispozitivul va încerca implicit să folosească ultimele detalii salvate

pentru a se conecta automat. Dacă conectarea automată nu reușește, se va porni portalul web. Dacă acesta eșuează să se pornească, se va afișa un log în consolă și se va reseta plăcuta.

În continuare, se inițializează conexiunea mqtt, controale, topicul pentru controale și autodescoperirea. Se trimite un mesaj pe topicul de autodiscovery pentru a identifica dispozitivul, după care procesul de setup se încheie iar led-ul se oprește.

În funcția loop se apelează pe durata funcționării dispozitivului funcția mqtt_loop pentru a monitoriza canalele.

4.2 Implementare server

Serverul web are rolul de a primi și procesa cereri HTTP, acesta redirecționându-le către brokerul MQTT, având în administrație și o bază de date. Pentru scrierea acestuia s-a utilizat limbajul GO.

```
package main
import (
    "context"
    "database/sql"
    "errors"
    "fmt"
    "net"
    "net/http"
    "strings"
    "time"

    "github.com/doug-martin/goqu/v9"
    _ "github.com/doug-martin/goqu/v9/dialect/mysql"
    mqtt "github.com/eclipse/paho.mqtt.golang"
    "github.com/gin-gonic/gin"
    "github.com/go-sql-driver/mysql"
)
```

Se definește package-ul main, după care se importă o serie de librării. Context este un pachet care se folosește pentru transmiterea de valori contextuale, database/sql permite conectarea la o bază de date sql, errors un pachet pentru gestiunea de erori, fmt pachetul de formatare, net și net/http sunt utilități pentru conectivitate la rețea, respectiv la o rețea http, strings este pachetul pentru operarea șirurilor de caractere, iar time este pachetul pentru măsurarea și afișarea timpului.

Ca librării externe, am importat Goqu (pentru baze de date), specific dialectul MySQL. Librăria Paho se ocupă cu stabilirea și managerierea conexiunii mqtt, acesteia atribuindui-se un alias. Gin este unealtă web care ne permite să manageriem cereri și să returnăm răspunsuri. Ultima librărie este un manager specific pentru bazele de date MySQL.

```
const (
    MQTT_BROKER = "tcp://localhost:1883"
    MQTT_USERNAME = ""
    MQTT_PASSWORD = ""
    MQTT_CLIENT_ID = "graduation-thesis-webserver"
    MYSQL_USER     = "root"
    MYSQL_PASSWORD = "Password1234."
    MYSQL_HOST     = "127.0.0.1"
    MYSQL_PORT     = "3306"
    MYSQL_DB       = "graduation"
)
```

În continuare se definesc constantele care sunt utilizate pentru conectarea la serverul MQTT respectiv la bază de date MySQL.

```
var (
    PUSH_ANGLE = fmt.Sprintf("%d", 130)
    PULL_ANGLE = fmt.Sprintf("%d", 0)
    MYSQL_URL = &mysql.Config{
        User:   MYSQL_USER,
        Passwd: MYSQL_PASSWORD,
        Net:    "tcp",
        Addr:   net.JoinHostPort(MYSQL_HOST, MYSQL_PORT),
        DBName: MYSQL_DB,
    }
)
type Device struct {
    ID       uint64 `json:"-" db:"id,skipinsert"`
    DeviceID string `json:"id" db:"device_id"`
    Name     string `json:"name" db:"name"`
    State    string `json:"state" db:"state"`
}
```

Se definesc variabilele care reprezintă gradele ținta la care trebuie să ajungă servomotorul în pozițiile de apăsare, respectiv de retragere, după care se compune URL-ul de conectare la MySQL alcătuit din constantele definite anterior.

Pentru a spori operabilitatea cu bază de date, aplicația android și dispozitivul, se definește un nou tip de data care încorporează detaliile unui dispozitiv (ID-ul în baza de date, ID-ul dispozitivului, numele dispozitivului și statul în care acesta se regăsește)

```go
func main() {
    opts := mqtt.NewClientOptions()
    opts.AddBroker(MQTT_BROKER)
    opts.SetClientID(MQTT_CLIENT_ID)
    opts.SetUsername(MQTT_USERNAME)
    opts.SetPassword(MQTT_PASSWORD)
    opts.SetAutoReconnect(true)
    opts.SetOrderMatters(false)
    opts.SetCleanSession(false)
    opts.SetKeepAlive(2 * time.Second)
    opts.SetPingTimeout(1 * time.Second)

    mqttClient := mqtt.NewClient(opts)
    defer mqttClient.Disconnect(5000)
    fmt.Println("connecting to mqtt")
    if token := mqttClient.Connect(); token.Wait() && token.Error() != nil {
        panic(token.Error())
    }
    fmt.Println("connected to mqtt")
```

Funcția main începe cu definirea opțiunilor pentru clientul MQTT. Sq inserează credențialele, după care se stabilesc reglulile de conectivitate. După aceea, se creează clientul de drept MQTT, căruia i se pasează opțiunile pentru conexiune. Se începe procesul de conectare, țar, dacă conexiunea eșuează, vom ieși din funcție returnând o eroare. În cazul în care conexiunea reușește, în consolă va fi afișat un log care să ateste acest lucru.

```go
mysqlDb, err := sql.Open("mysql", MYSQL_URL.FormatDSN())
if err != nil {
    panic(err)
}
defer mysqlDb.Close()
if err := mysqlDb.Ping(); err != nil {
    panic(err)
}
db := goqu.New("mysql", mysqlDb)
```

Același lucru se întâmplă și în cazul serverului MySQL, serverul încercând să se conecteze la aceasta, iar în cazul în care nu reușește, va părăsi funcția returnând o eroare.

```
t := mqttClient.Subscribe("connected-device", 2, func(c mqtt.Client, m mqtt.Message) {
    fmt.Println("received autodiscovery message:", string(m.Payload()))
    payload := strings.Split(string(m.Payload()), ";")
    deviceID := payload[0]
    deviceName := payload[1]
    go func() {
        _, err := db.Insert(goqu.I("devices")).Rows(Device{
            DeviceID: deviceID,
            Name:     deviceName,
            State:    "default",
        }).Executor().ExecContext(context.Background())
        if err != nil {
            fmt.Printf("failed to add new device: %v\n", err)
        }
    }()
})
go func() {
    <-t.Done()
    if t.Error() != nil {
        panic(t.Error())
    }
}()
```

Vom abona serverul la topicul MQTT în care dispozitivele își fac autodiscovery, iar în momentul în care acesta va recepționa un astfel de mesaj pe topic, va împărți șirul de caractere care reprezintă mesajul în funcție de separator, salvând rezultatele (ID-ul și numele dispozitivului) în două variabile diferite. Aceste date vor fi inserate în baza de date, iar în cazul în care inserția eșuează, o eroare va fi afișată în consolă.

```
router := gin.Default()
router.GET("/push/:device_id", func(c *gin.Context) {
    fmt.Printf("device %s -- push\n", c.Param("device_id"))
    token := mqttClient.Publish(c.Param("device_id"), 0, false, PUSH_ANGLE)
    if token.Wait() && token.Error() != nil {
        c.Error(token.Error())
        c.Status(http.StatusInternalServerError)
        return
    }
    result, err := db.
        Update(goqu.I("devices")).
        Set(goqu.Record{"state": "push"}).
        Where(goqu.I("device_id").Eq(c.Param("device_id"))).
        Executor().
        ExecContext(c.Request.Context())
    if err != nil {
        c.Error(err)
        c.Status(http.StatusInternalServerError)
        return
    }
    rowsAffected, err := result.RowsAffected()
    if err != nil {
        c.Error(err)
        c.Status(http.StatusInternalServerError)
        return
    }
    if rowsAffected != 1 {
        c.Error(errors.New("no such device"))
        c.Status(http.StatusInternalServerError)
        return
    }
    c.Status(200)
})
```

Mai departe, se va genera router-ul web. Vom înregistra o cale pentru trimiterea comenzilor de apăsare sub formă de cerere. Aceasta este formată din numele comenzii (push) și din ID-ul dispozitivului căruia îi este adresată comandă. În momentul în care o astfel de cerere ajunge pe server, acesta va publica mesajul pe topicul MQTT aferent dispozitivului țintă. Dacă acest mesaj nu reușește să fie postat, vom returna emițătorului un cod de eroare. Dacă mesajul reușește să fie postat, vom actualiza statusul dispozitivului în baza de date. Dacă actualizarea nu reușește, vom returna de asemenea un cod de eroare. În cazul în care actualizarea a reușit, însă nici o coloană nu a fost afectată, înseamnă că cererea a fost eronată și vom returna de asemenea o

eroare. Dacă până la acest punct nu am întâmpinat erori, vom returna un cod care să ateste succesul operației.

```
router.GET("/pull/:device_id", func(c *gin.Context) {
    fmt.Printf("device %s -- pull\n", c.Param("device_id"))
    token := mqttClient.Publish(c.Param("device_id"), 0, false, PULL_ANGLE)
    if token.Wait() && token.Error() != nil {
        c.Error(token.Error())
        c.Status(http.StatusInternalServerError)
        return
    }
    result, err := db.
        Update(goqu.I("devices")).
        Set(goqu.Record{"state": "pull"}).
        Where(goqu.I("device_id").Eq(c.Param("device_id"))).
        Executor().
        ExecContext(c.Request.Context())
    if err != nil {
        c.Error(err)
        c.Status(http.StatusInternalServerError)
        return
    }
    rowsAffected, err := result.RowsAffected()
    if err != nil {
        c.Error(err)
        c.Status(http.StatusInternalServerError)
        return
    }
    if rowsAffected != 1 {
        c.Error(errors.New("no such device"))
        c.Status(http.StatusInternalServerError)
        return
    }
    c.Status(200)
})
```

Același proces are loc și pentru operația de “retragere”, acesta fiind configurat pe o cale diferită.

```
router.GET("/devices", func(c *gin.Context) {
    devices := make([]Device, 0)
    err := db.
        From(goqu.I("devices")).
        ScanStructsContext(c.Request.Context(), &devices)
    if err != nil {
        c.Error(err)
        c.Status(http.StatusInternalServerError)
        return
    }
    c.JSON(http.StatusOK, devices)
})
```

Nu în ultimul rând, aplicația mobilă va cere o lisă de dispozitive pentru afișarea meniului. Vom scâna lista dispozitivelor disponibile în baza de date. În cazul în care această operație eșuează, vom returna un cod de eroare, iar, în caz contrar, vom returna un cod de succes împreună cu lista dispozitivelor în format JSON.

```
    fmt.Println("running")
    http.ListenAndServe(":8888", router)
}
```

După ce toate aceste rute au fost configurate, serverul poate începe să ruleze, așteptând cereri.

4.3 Implementare aplicație mobilă

Aplicația mobilă este destinată utilizatorului și are rolul de a oferi acestuia posibilitatea de a acționa dispozitivele de la distanță. Aceasta a fost scrisă în mediul Android Studio utilizând limbajul de programare Java.

Aceasta este structurată în: activități, adaptoare, modele și servicii.

4.3.1 Servicii

Folderul “servicies” se ocupă de logica de business. aici se regaseste fișierul DeviceService.java.

```
public interface DeviceService {
    @GET("push/{device_name}")
    Call<Void> push(@Path("device_name") String deviceName);
    @GET("pull/{device_name}")
    Call<Void> pull(@Path("device_name") String deviceName);
    @GET("devices")
    Call<List<Device>> getDevices();
}
```

În interfața DeviceService, cu ajutorul librariei Retrofit, se definește clientul pentru API. Fiecare adnotare HTTP de tip GET este urmată de o cale relativă, care are în componență metoda (push/pull/devices), cât și un bloc înlocuibil, delimitat de acolade. Blocul înlocuibil constă in numele dispozitivului. Pe linia următoare, se specifică că la apelul cererii, prin adnotarea Path, că blocul înlocuibil ia valoarea variabilei referențiate. În cazul cererii catre “devices”, se va cere o listă a tuturor dispozitivelor.

4.3.2 Modele

Folderul “models” cuprinde definiții pentru modele și structuri de date. Aici se regaseste fisierul “Device.java” în care se regăsește structura logică/obiectuală a unui dispozitiv.

```
public class Device {
    @SerializedName(value = "id")
    public String deviceID;
    @SerializedName(value = "name")
    public String name;
    @SerializedName(value = "state")
    public String state;
}
```

Adnotarea “SerializedName” din cadrul GSON permite maparea numelui obiectului din clasa model cu numele cu care acesta trebuie serializat în JSON în cazul în care acestea diferă. Există 3 câmpruei de acest fel: “deviceID” cu corespondentul in JSON “id”, “name” care are aceeași denumire și în JSON și “state” care de asemenea va fi serializat la fel în JSON.

4.3.3 Adaptoare

În folderul “adapters” regăsim fișierul “DeviceListAdapter.java”, care are rolul de a adapta o listă de dispozitive în așa fel încât să fie prezentabilă într-o vizualizare de tip “ListView”.

```
public class DeviceListAdapter extends BaseAdapter {

    private static final String TAG = "DeviceListAdapter";

    private final List<Device> items;
    private final DeviceService deviceService;

    public DeviceListAdapter(DeviceService deviceService) {
        this.items = new ArrayList<>();
        this.deviceService = deviceService;
        reloadItems();
    }
```

String-ul “TAG” se folosește pentru scrierea de log-uri, lista “items” este lista propriu-zisă de dispozitive, deviceService este serviciul cu care interacționăm cu dispozitivele propriu-zise. Pe linia următoare avem constructorul obiectului, care inițializează obiectele și apelează funcția reloadItems.

```java
public void reloadItems() {
    this.deviceService.getDevices().enqueue(new Callback<List<Device>>() {
        @Override
        public void onResponse(Call<List<Device>> call, Response<List<Device>> response) {
            items.clear();
            if (response.body() == null) {
                Log.e(TAG, msg: "received empty device list body");
                return;
            }
            items.addAll(response.body());
            notifyDataSetChanged();
        }

        @Override
        public void onFailure(Call<List<Device>> call, Throwable t) {
            Log.e(TAG, msg: "onFailure: failed to get device list", t);
            items.clear();
            notifyDataSetChanged();
        }
    });
}
```

Funcția reloadItems folosește membrul “deviceService” care a fost initialized în constructor pentru a primi o listă de dispozitive. Funcția “onResponse” golește lista inițială și verifica dacă după efectuarea cererii HTTP exista o lista cu dispozitive. În cazul în care răspunsul nu este nul, dispozitivele vor fi adaugate în listă, după care schimbarea va fi anunțată. Funcția onFailiure se apelează atunci când există o eroare la trimiterea/primirea cererii/răspunsului HTTP, caz în care lista se va goli și acest lucru va fi notificat.

```java
@Override
public int getCount() { return items.size(); }

@Override
public Object getItem(int i) { return items.get(i); }

@Override
public long getItemId(int i) { return getItem(i).hashCode(); }
```

Pe liniile următoare, se implementează metodele de bază din adaptorul de bază, și anume numărul de dispozitive din listă, returnarea unui anume dispozitiv, respectiv un hash code al id-ului obiectului.

```
@Override
public View getView(int i, View view, ViewGroup viewGroup) {
    if (view == null) {
        LayoutInflater inflater = LayoutInflater.from(viewGroup.getContext());
        view = inflater.inflate(R.layout.device_list_item, viewGroup,  attachToRoot: false);
    }
    final Device device = items.get(i);
    final TextView deviceNameTextView = (TextView) view.findViewById(R.id.device_name_text_view);
    if (device.name.isEmpty()) {
        deviceNameTextView.setText(device.deviceID);
    } else {
        deviceNameTextView.setText(device.name);
    }
    final TextView deviceStatusTextView = (TextView) view.findViewById(R.id.device_status_text_view);
    deviceStatusTextView.setText(device.state);
```

Funcția getView este funcția care are rolul de a randa fiecare obiect din listă. Daca view-ul este nul, atunci acesta va trebui create. În continuare se extrage un dispozitiv aferent acelei poziții din lista care urmează a fi randată, după care se initializeaza un TextView pentru nume, unul pentru status și apoi cele doua butoane care controlează dispozitivul. Dacă dispozitivului nu i s-a atribuit un nume, acesta va fi înlocuit cu id-ul sau.

```
final Button devicePushButton = (Button) view.findViewById(R.id.device_push_button);
devicePushButton.setOnClickListener(devicePushButtonView -> {
    Call<Void> call = deviceService.push(device.deviceID);
    call.enqueue(new Callback<Void>() {
        @Override
        public void onResponse(Call<Void> call, Response<Void> response) {
            reloadItems();
        }
        @Override
        public void onFailure(Call<Void> call, Throwable t) {
            Log.e(TAG,  msg: "onFailure: device push failed", t);
            t.printStackTrace();
            reloadItems();
        }
    });
});

final Button devicePullButton = (Button) view.findViewById(R.id.device_pull_button);
devicePullButton.setOnClickListener(devicePullButtonView -> {
    Call<Void> call = deviceService.pull(device.deviceID);
    call.enqueue(new Callback<Void>() {
        @Override
        public void onResponse(Call<Void> call, Response<Void> response) {
            reloadItems();
        }
        @Override
        public void onFailure(Call<Void> call, Throwable t) {
            Log.e(TAG,  msg: "onFailure: device pull failed", t);
            t.printStackTrace();
            reloadItems();
        }
    });
});
```

Butoanelor de push, respectiv pull li se atribuie capacitatea de a lansa cereri HTTP aferente. La final, se returnează vederea creată.

4.3.4 Activități

În folderul “activities” se regasesc activitățile aplicației. Aici avem ecranul “MainActivity.java”, care este meniul principal al aplicației.

```java
public class MainActivity extends AppCompatActivity {

    private static final String TAG = "MainActivity";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Log.d(TAG, msg: "creating http service utilities");
        final Gson gson = new Gson();
        final Retrofit retrofit = new Retrofit.Builder()
                .baseUrl("http://192.168.1.193:8888")
                .addConverterFactory(GsonConverterFactory.create(gson))
                .build();

        Log.d(TAG, msg: "creating device service");
        final DeviceService deviceService = retrofit.create(DeviceService.class);

        Log.d(TAG, msg: "creating device list adapter");
        final DeviceListAdapter deviceListAdapter = new DeviceListAdapter(deviceService);

        Log.d(TAG, msg: "creating device list");
        final ListView deviceList = findViewById(R.id.device_list);
        deviceList.setAdapter(deviceListAdapter);

        Log.i(TAG, msg: "initialized main activity");
    }
}
```

La crearea activității se alege layout-ul, după care se inițializează Gson petru JSON, Retrofit pentru constructorul de servicii, după care se inițializează serviciul de dispozitive, adaptorul, și lista.

5. Evaluare

Dispozitivele IoT sunt predispuse la o serie de erori pe parcursul implementării, întrucât acestea pot genera erori atât la nivel software, cât și hardware, motiv pentru care acestea necesită o testare intensivă.

În primul rând, înainte de a intra în partea de software, am dorit să ne asigurăm că componentele sunt funcționale. Pentru început, am dorit să testăm atât dacă servomotorul este funcțional, cât și unghiurile de rotație ale acestuia și mai ales să verificăm dacă acesta are puterea necesară pentru a acționa un întrerupător, motiv pentru care s-a achiziționat un modul de testare servo *REV SRS Programmer*.

Conform website-ului producătorului *revrobotics.com, 2021,* modulul SRS Programmer este utilizat pentru a testa atât servomotoarele obușnuite, cât și servomotoarele cu rotație continuă (care nu sunt limitate unghiular) fără a fi nevoie de scrierea de cod pentru controlul acestora.

Modulul are un mod de funcționare simplu. Acesta permite conectarea cu ușurința servomotorului, testarea lui automată (rotește servomotorul înainte și inapiu până ajunge la limitele acestuia) precum și testarea manuală (oferă posibilitatea de a mișca servomotorul la poziția minimă, poziția maximă cât și poziția 0, de centru). Pentru testarea servomotorului dispozitivului, s-au folosit atât testarea automată, care a permis observarea unghiurilor, cât și testarea manuală.

Testarea manuală a fost utilă pentru a detecta poziția de centru. Este nevoie ca această poziție să fie cunoscută pentru încorporarea în dispozitiv. Servomotorului i s-a atașat o extensie de plastic care a permis testarea acestuia pentru întrerupătoare. În urma testelor, s-a constatat că servomotorul de 9 grame este capabil să apese atât întrerupătoare obișnuite, de lumină, cât și întrerupătoare mai dure, de tip switch.

Testarea manuală a fost utilizată și după printarea carcasei. Aceasta a permis testarea și montarea sistemului de apăsare pe șina zimțată, contribuind la verificarea poziției 0 în sistem, cât și a limitelor.

Testarea modulului NodeMCU s-a realizat cu ajutorul librăriei Arduino ESP8266WiFi.h, dar și a monitorului de serial încorporat în suita PlatformIO, care permite transmiterea de informații între dispozitiv și computer. Librăria oferă o serie de funcții pentru conectivitatea la WiFi, iar, prin utilizarea anumitor funcții puse la dispoziție de către aceasta, a fost posibilă testarea.

S-a scris un program simplu de test care scana rețelele WiFi din perimetru, după care le afișa în monitorul de serial. După aceea, tor prin intermediul monitorului de serial, se putea selecta una din aceste rețele și introducerea credențialelor pentru a realiza conexiunea WiFi, un log care să ateste succesul fiind afișat.

Pentru testarea brokerului MQTT, s-a utilizat aplicația MQTTBox. Aceasta permite emularea de clienți digitali care se pot conecta la serverul MQTT. Acești clienți se pot abona la topicuri și pot posta mesaje sub diferite formate.

Brokerul MQTT HiveMQ dispune de o interfață de administrator sub formă de pagină web, care este rulată implicit pe portul 8080 al sistemului gazdă. Această interfață permite monitorizarea clienților care sunt conectați la acesta, gestionarea acestora cât și vizualizarea și transmiterea mesajelor pe topicuri. Acest panou a fost utilizat atât pentru testarea conexiunii MQTT cu clienții MQTTBox, cât și pentru testarea conexiunii cu serverul, respectiv dispozitivul.

Pentru testarea serverului web, s-a folosit un browser web (de pe aceeași mașină gazda) care să trimită manual cererile de tip GET programate pe căile URL, răspunsurile primite în urma acestor cereri putând fi vizualizate în consola browserului.

Pentru testarea bazei de date MySQL, s-au inserat mai mulți clienți fictivi prin intermediul serverului web, acestora aplicându-se diverse operații CRUD.

Aplicația mobilă s-a testat prin intermediul unui emulator de android. Acest emulator se numește Nox App Player și permite instalarea de aplicații Android cât și testarea funcțiilor acestora în mod dezvoltator, având posibilitatea de a replica mai multe modele de dispozitive Android, cu rezoluții și setări de sistem diferite.

Aplicația a fost testată totodată și cu un telefon Android fizic, care are activat modul dezvoltator și depanarea USB. Aceste metode fac posibilă testarea integrală a tuturor posibilităților.

Fiecare dintre componentele programabile (aplicație mobilă, dispozitiv, server web) are încorporat în cod un sistem de log-uri. După fiecare funcție realizată de unul dintre sisteme, un log va fi afișat în consola acestuia. Aceste opțiuni sunt vizibile doar dezvoltatorului.

Concluzii

Tehnologiile IoT au o dinamică de dezvoltare impresionantă, numărul și diversitatea acestora crescând în mod vizibil de la o zi la alta. Lumea migrează către tehnologii de tip smart într-un ritm atât de alert încât infrastructura nu poate face față.

Lumea așa cum o știm noi nu poate fi reconstruită de la 0 pentru a se alinia în rând cu cele mai noi tehnologii , așadar este nevoie de soluții alternative, care să vină în sprijinul acestei tranziții, ajutând lumea să țină pasul.

În urma realizării proiectului de licență am ajuns la concluzia că astfel de soluții sunt viabile. Nu este nevoie de o avere pentru a beneficia de tehnologiile smart. S-a realizat un produs IoT care are un pret redus de producție, care s-a dezvoltat prin utilizarea de tehnologii consacrate, care este capabil să ducă sistemele non inteligente la următorul nivel, reducând astfel risipa de aparatură și crescând eficiența economică și ecologică a planetei.

Pe viitor se dorește ca acest produs să dezvolte un branding propriu, care să fie asociat cu inovare, acoperint mai multe segmente de soluționare IoT. Se dorește implementarea unei funcții de control pentru dispozitive multiple care pot fi grupate (de exemplu pe camere), implementarea unui sistem de înregistrare/logare cu baza de date centralizată, cât și separarea pachetelor pentru întreprinderi față de cele pentru persoane fizice.

De asemena, în cadrul aceleiași aplicații (conectată implicit la același server, baza de date, broker MQTT) să poată fi integrate mai multe dispozitive IoT de tipuri diferite (priză inteligență, bec inteligent, ușa inteligentă, etc.)

Pe parcursul realizării acestui proiect, am avut ocazia să îmi dezvolt atât cunoștințele despre electronică, cât și să deprind abilități de backend, MQTT și Cloud. Consider totodată că am deprins capacități de a face aplicații diferite, scrise în limbaje diferite, care rulează pe sisteme diferite să poată opera unele cu altele. Toate aceste abilități deprinse s-au materializat în dezvoltarea unui dispozitiv care ajută lumea în care trăim să țină pasul cu dinamica cu care se dezvoltă noi tehnologii smart.

Glosar termeni

(dispozitiv) smart - obiect de zi cu zi care utilizează tehnologii computaționale, inteligență artificială sau internet pentru a performa funcții inteligente

Waterfall - model de dezvoltare software care nu permite trecerea la următorul pas până pasul curent nu a fost completat

Broker (MQTT) - program de tip server care intermediază comunicarea între două entități

Servomotor - motor care permite controlul poziției unghiulare

NodeMCU - placă de dezvoltare cu modul WiFi inegrat

ESP8266 - modul WiFi

Memorie Flash - tip de memorie digitală care se poate suprascrie

Pin digital - pin care citește semnale digitale, de tip binar(0/1), separate de un interval de timp

Pin analog - pin care citește semnale analoage, continue, care reprezintă măsurări de voltaj

Semnal PWM - tip de semnal digital

Firmware - software care este direct scris pe hardware pentru controlul anumitor dispozitive

Potențiometru - component electronic care măsoară diferența de potențial

Cutie de transmisie - ansamblu care facilitează transmiterea mișcării

Serializare - transcrierea unei informații dintr-un format în altul

API - interfață care permite aplicațiilor să comunice între ele

Bibliografie

Anita Gehlot, Rajesh Singh, Praveen Kumar Malik, Lovi Raj Gupta, Bhupendra Singh (2020), Internet of Things with 8051 and ESP8266, Editura Taylor & Francis Ltd

Ciaca Monica (2018), Suport de curs - Analiza şi proiectarea sistemelor informatice

Douglas Hughey (2009), The Traditional Waterfall Approach, https://www.umsl.edu/~hugheyd/is6840/waterfall.html

Harry Chen, Tim Finin, Anupam Joshi (2004), Semantic Web in the Context Broker Architecture https://mdsoar.org/bitstream/handle/11603/12230/76.pd.pdf?sequence=1&isAllowed=y

Louis Costa (1998), Open Systems Interconnect (OSI) Model https://journals.sagepub.com/doi/full/10.1177/221106829800300108

Mohsen Hallaj Asghar (2016), Internet of Things Architecture and Research App with MQTT Protocol, Editura LAP LAMBERT Academic Publishing

Shakirat Haroon- Sulyman (2014), Client-Server Model https://www.researchgate.net/profile/Shakirat-Sulyman/publication/271295146_Client-Server_Model/links/5864e11308ae8fce490c1b01/Client-Server-Model.pdf

*** Biroul European de Statistică Eurostat (2021), How digitalised are EU's enterprises? https://ec.europa.eu/eurostat/web/products-eurostat-news/-/ddn-20211029-1

*** Comisia Europeană (2021), Comunicatul 118, Busola pentru dimensiunea digitală 2030 https://eur-lex.europa.eu/legal-content/en/TXT/?uri=CELEX%3A52021DC0118

*** components101.com - platformă comerciant, NodeMCU ESP8266 pinout features and data sheet https://components101.com/development-boards/nodemcu-esp8266-pinout-features-and-datasheet

*** ElectronicWings (2022), Introduction to Servo motors https://www.electronicwings.com/sensors-modules/servo-motor

*** Fiberlogy, NYLON PA12 Technical Sheet https://fiberlogy.com/en/fiberlogy-filaments/nylon-pa12/

*** Institutul Naţional de Statistică (2021), Accesul populaţiei la tehnologia informaţiei şi comunicaţiilor https://insse.ro/cms/ro/tags/accesul-populatiei-la-tehnologia-informatiei-si-comunicatiilor

*** Institutul Național de Statistică (2017), Rezultate și performanțe ale întreprinderilor din comerț și servicii
https://insse.ro/cms/sites/default/files/field/publicatii/rezultate_si_performante_ale_intreprinderilor_din_comert_si_servicii_in_2017.pdf

*** Ivan Grokhotkov (2017), Filesystem - ESP8266 Arduino Core's documentation
https://arduino-esp8266.readthedocs.io/en/latest/filesystem.html

*** Json.org (2014), Introducing JSON
https://www.json.org/json-en.html

*** Ministerul Educației (2018), Rezultatele elevilor din România la evaluarea internațională PISA
https://www.edu.ro/rezultatele-elevilor-din-rom%C3%A2nia-la-evaluarea-interna%C8%9Bional%C4%83-pisa-2018

*** PISA (2018), Global Competence 2018
https://pisadataexplorer.oecd.org/ide/idepisa/dataset.aspx

*** REV Robotics (2021), SRS Programmer technical sheet
https://docs.revrobotics.com/15mm/actuators/servos/srs-programmer

*** Statista (2021), Romania: Age structure from 2010 to 2020
https://www.statista.com/statistics/373125/age-structure-in-romania/

*** Statista (2022), Smart Homes in Europe
https://www.statista.com/outlook/dmo/smart-home/europe

*** Teoalida (2020), Baza de date a blocurilor în Excel
https://www.teoalida.ro/lista-blocuri/

*** Uniunea Europeană (2021), Europe’s Digital Decade: digital targets for 2030
https://ec.europa.eu/info/strategy/priorities-2019-2024/europe-fit-digital-age/europes-digital-decade-digital-targets-2030_en#documents

Documentații oficiale

WiFiManager.h, https://github.com/tzapu/WiFiManager

ArduinoJson.h, https://github.com/bblanchon/ArduinoJson

PubSubClient.h, https://github.com/knolleary/pubsubclient

Servo.h, https://github.com/esp8266/Arduino/blob/master/libraries/Servo/src/Servo.h

Goqu, https://github.com/doug-martin/goqu

Paho, https://github.com/eclipse/paho.mqtt.golang

Gin, https://github.com/gin-gonic/gin

MySQLDriver, https://github.com/go-sql-driver/mysql

GSON, https://www.javadoc.io/doc/com.google.code.gson/gson/2.8.0/com/google/gson/Gson.html

Retrofit, https://square.github.io/retrofit/

PlatformIO, https://docs.platformio.org/en/latest/projectconf/index.html