



Universitatea Politehnica Bucureşti
Facultatea de Automatică și Calculatoare
Departamentul de Automatică și Ingineria Sistemelor

LUCRARE DE DIPLOMĂ

Sistem IoT pentru controlul accesului în clădire

Absolvent
Alexandru Cristian IONESCU

Coordonator
Prof. Dr. Ing. Mihnea Alexandru MOISESCU

Bucureşti, 2022

Cuprins

1	Introducere	1
1.1	Obiectivele lucrării de licență	1
1.2	Descrierea domeniului din care face parte proiectul de diplomă	2
1.3	Prezentare pe scurt a capitolelor	3
2	Descrierea problemei abordate	4
2.1	Formularea problemei	4
2.2	Studiu asupra realizărilor similare din domeniu	5
2.3	Stabilirea cerințelor funcționale și nefuncționale ale sistemului	8
3	Stadiul actual in domeniu și selectarea soluției tehnice	10
3.1	Stadiul actual al tehnologiilor utilizate pentru dezvoltarea soluției	10
3.2	Prezentarea tehnologiilor si platformelor de dezvoltare alese	12
4	Considerente legate de implementarea soluției tehnice	14
4.1	Arhitectura sistemului	14
4.2	Implementarea sistemului	22
4.3	Testarea sistemului	24
5	Studiu de caz	27
5.1	Oferirea accesului unui utilizator	27
5.2	Raspuns automat	28
5.3	Raspuns de la distanta	29
6	Concluzii	31
6.1	Concluzii	31
6.2	Contributii	31
6.3	Dezvoltari ulterioare	31
7	Bibliografie	32

1 Introducere

1.1 Obiectivele lucrării de licență

1.1.1 Realizarea unui studiu de piață pentru determinarea fezabilității soluției

Pentru realizarea unui sistem [Internet of Things \(IoT\)](#) care să se plieze pentru nevoiele secolului 21, este necesară o studiere mai aprofundată a caracteristicilor tehnice specifice. Astfel capitolul de introducere conturează ideile principale despre realizarea unui studiu de piață pentru determinarea fezabilității și pașii necesari pentru proiectarea unui sistem compatibil [POTS](#).

În continuare se va realiza un scurt studiu de piață pe nișa sistemelor [IoT](#) destinate uzului casnic. Un caz particular de astfel de dispozitive sunt cele care îndeplinesc funcția de interfon sau oferă contrulul accesului într-o incintă de la distanță.

În momentul de față există pe piață o multitudine de produse de tip încuietoare inteligentă sau sisteme tip interfon GSM, atât de la producători cunoscuți cât și de la branduri nou înființate. Această lucrare va analiza trei tipuri de soluții existente, cu implementări diferite, încercând să identifice funcționalități comune, avantaje și dezvantaje regăsite într-o plajă cât mai mare de dispozitive de pe piață.

Situatia actuala din Romania prezintă o piață cu o nevoie de îmbunătățire în ceea ce privește sistemele actuale de interfon, deoarece majoritatea dintre ele au fost integrate în infrastructura blocurilor construite în trecut. Prin urmare există un segment de piață de utilizatori care ar dori să beneficieze de funcțiile interfonului intelligent, dar nu au această posibilitatea deoarece ar presupune schimbarea sistemului din întreaga clădire.

1.1.2 Dezvoltarea unui sistem compatibil POTS pentru interfațarea în rețeaua IoT

Sistemul propus în această lucrare se poate conecta la rețeaua [Plain Old Telephone Service \(POTS\)](#) printr-o simplă mufă RJ11 lucrul ce ar trebui să ușureze adoptarea unei îmbunătățiri la soluția actuală.

1.2 Descrierea domeniului din care face parte proiectul de diplomă

Unul dintre scopurile principale ale acestui proiect este evidențierea domeniului de automatizări IoT casnice adoptat din ce în ce mai des de consumatori domestici. Potrivit studiilor din domeniu, numărul de dispozitive IoT conectate la internet vă ajunge aproximativ la 75.44 miliarde în 2025 [1]

1.2.1 Istoric

Interesul în conectarea locuințelor pentru a obține funcționalitate adițională datează încă din anii 60, majoritatea fiind concepte prototipate de entuziaști cu înclinații spre electronică.

Jim Sutherland, inginer la Westinghouse a creat primul sistem de automatizare a domiciliului în anul 1964, ECHO IV. Acesta era capabil să controleze temperatură, alte apărate casnice cât și să permită reținerea de mementouri sau liste de cumpărături. Cu introducerea rețelei Advanced Research Projects Agency Network (ARPANet) în 1969, un precursor al Internetului, universul dispozitivelor casnice conectate a cunoscut o perioadă rapidă de dezvoltare în anii următori [18].

Trecerea de la o nouătate scumpă la un sistem ce oferă funcții cu adevărat practice a venit sub forma proiectului "X10 Home Automation". Se putea integra cu sistemul de climatizare existent al clădirii, controla electrocasnice mici, cât și corpuri de iluminat.

În anul 1984, Asociația Națională a Constructorilor din Statele Unite a creat un grup de control numit "Smart House" pentru a accelera includerea tehnologiei în proiectele viitoare [2].

Pentru consumatori, dezvoltările din următoarii ani au adus uși automate pentru garaje, termostate programabile și sisteme de securitate în cadrul monden, concomitent reducând prețurile soluțiilor oferte. În ciuda acestor semne, sociologii au concluzionat la vremea respectivă că nu există un interes real în conceptul "Smart House".

1.2.2 Stadiu actual

În prezent soluțiile de tip Smart Home oferă o multitudine de funcționalități, adună și agregă informații de la diferiți senzori plasați în casă și agregă informațiile spre a fi afișat un rezumat utilizatorului. Printre informațiile monitorizate se pot

Soluțiile de tip "Smart Home" din prezent se integrează în general cu o rețea precum Espressif, Apple HomeKit sau Google Home. Această permite controlul dispozitivelor conectate prin intermediul telefonului mobil [15].

1.3 Prezentare pe scurt a capitolelor

Capitolul 1 prezintă noțiuni teoretice în scopul familiarizării cititorului în legătură cu soluțiile actuale prezente pe piață sistemelor IoT. Înainte de propunerea unei modalități care să permită atingerea obiectivelor propuse, este necesar un studiu din trecut până în prezent asupra metodelor pe baza cărora să se poată contura o idee a ceea ce s-a putut obține, până în acest moment pe piață.

Copitolul 2 ilustrează perspectiva utilizatorului doritor de automatizarea tehnologică a lucrurilor ce îl înconjoară. Acest capitol se axează și pe o prezentarea scurtă a câtorva dispozitive ce îndeplinesc funcția de încuietoare intelligentă, fiind comparate pentru a identifica punctele comune, dar și funcțiile unice ale sistemelor studiate.

Capitolul 3 începe prin prezentarea inovațiilor tehnologice implementate și acceptate de către societate în prezent. Aceste inovații au fost menționate datorita aportului adus în schimbarea paradigmăi procesării multiplelor surse de informații provenite de la utilizatori pentru facilitarea activităților zilnice. În urma analizei literaturii de specialitate se detaliază soluțiile componente ale sistemului ce va fi implementat.

Pe baza noțiunilor teoretice dobândite în capitoalele anterioare, Capitolul 4 prezintă detalii tehnice cât și algoritmi utilizați în soluția propusă. De asemenea se conțurează pașii efectuați în conceperea metodei propuse pentru îndeplinirea scopului. Așadar, în continuare lucrarea detaliază arhitectura sistemului, dezvoltarea unui HAT pentru Raspberry Pi, dar și componeta software alcătuită din server și clienți. Ultima etapă a procesului de dezvoltare a implicat testarea componentelor individuale cât și întregul ansamblu.

Capitolul 5 demonstrează 3 cazuri acoperite de funcționalitățile aplicației din perspectiva utilizatorului.

Structura proiectului de diplomă se termină prin menționarea concluziilor și contribuțiilor originale aduse în urma dobândirii cunoștințelor teoretice și practice. Faptul că tehnologia avansează cu pași rapizi și zilnic apar soluții noi, se vor contura de asemenea perspective viitoare de studiat.

2 Descrierea problemei abordate

2.1 Formularea problemei

În orașe precum București, majoritatea blocurilor au fost construite înainte de anul 1990 și prin urmare interfoanele lor se bazează pe **POTS**. Trăind în era digitală, utilizatorul ideal își dorește augmentarea functionalităților sistemului existent, pentru a nu trebui să își convingă toți vecinii să investească în modernizarea sistemului de acces. Pentru a putea adresa cât mai mulți utilizatori, soluția acestei probleme trebuie să fie agnostică de smartphoneul și interfonul existent al utilizatorului, dar să ofere integrări cu alte soluții de tip "Smart Home".

În funcție de perioada instalării, sistemele să împart în două categorii: analogice și digitale. Posturile de interfon analogice sunt legate la o unitate de comandă care decodează semnale **Dual-Tone Multi-Frequency (DTMF)**, generează un semnal sinusoidal cu frecvență de 20Hz și amplitudinea de 60-90V pentru sonerie, apoi realizează conexiunile necesare dintre postul de afara și cel al apartamentului căutat. Sistemele digitale folosesc o magistrală comună de comunicații, fiind adresate conform unei scheme prestabilite - fiecare terminal este programat cu o adresă, însă pentru acest procedeu este nevoie de o cheie asociată unității de comandă.

Din motive istorice, unitatea centrală **POTS** generează semnalul sinusoidal și poartă suficient curent pentru a putea alimenta clopotul apărut în prima generație de telefoane. Prin urmare, sistemele digitale sunt considerate mai eficiente și mai sigure, dar și mai greu de integrat, datorită implicării unei persoane autorizate care să programmeze întregul sistem.

Această lucrare vă analiza soluții digitale, însă sistemul final vă fi implementat pe un terminal analog. Așadar, trebuie să înțelegem în primul rând mecanismul de adresare și cum este el interpretat de centrală. După cum insinuează numele, **DTMF** presupune generarea a două tonuri de frecevente diferite în același timp, conform liniei și coloanei tastei apăsate. Acest semnal vă fi interpretat de decodorul de semnal al centralei cu ajutorul unor filtre de tip notch spre a se realiza conexiunile necesare.

	1209Hz	1336Hz	1477Hz
697Hz	1	2	3
770Hz	4	5	6
852Hz	7	8	9
941Hz	*	0	#

Tabelul 2.1: Tabel frecvențe DTMF

Sistemul descris până acum poate adresa $12 - 1 = 11$ posturi diferite (centrala este considerată și ea post și are un slot rezervat). Pentru a adresa mai multe posturi,

unitatea de comandă trebuie să includă și un circuit logic secvențial pentru a reține starea ultimelor taste apăsate. Astfel, ajungem la un număr satisfăcător de adrese pentru aplicația interfonului.

Un exemplu de analiza spectrală a unui astfel de semnal:

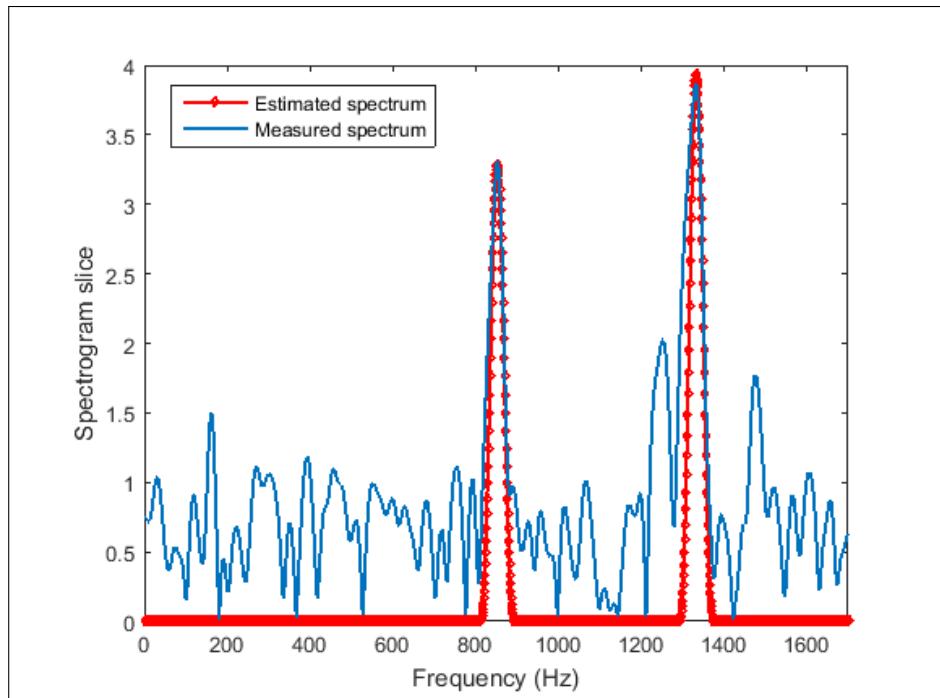


Figura 2.1: Spectrogramă tasta "8" [3]

Se pot distinge grafic două frecvențe predominante: 852Hz și 1336Hz - combinația corespunzătoare tastei "8".

2.2 Studiu asupra realizărilor similare din domeniu

2.2.1 Videx UK

Interfoanele [GSM](#) de la Videx sunt conectate la rețeaua mobilă de telefonie și permit operarea unei porți prin intermediul unui releu. Ele necesită doar o sursă de curent externă, o antenă și o cartă [Subscriber Identity Module \(SIM\)](#) pentru a opera.

Printre funcționalitățile principale se numără:

- Poate include un cititor de carduri [RFID](#) și cheie
- Versiune rezistentă la vandalism
- Până la 4 numere de telefon per apartament, pentru redundantă. În cazul în care primul număr nu se poate apela sau nu răspunde, se va încerca următorul număr programat
- Oferă aplicație Android și iOS pentru programat unitatea



Figura 2.2: Sistem interfon Videx GSM [16]

Dezavantaje:

- Nu oferă integrare cu servicii din rețeaua IoT

2.2.2 Google Nest x Yale Lock

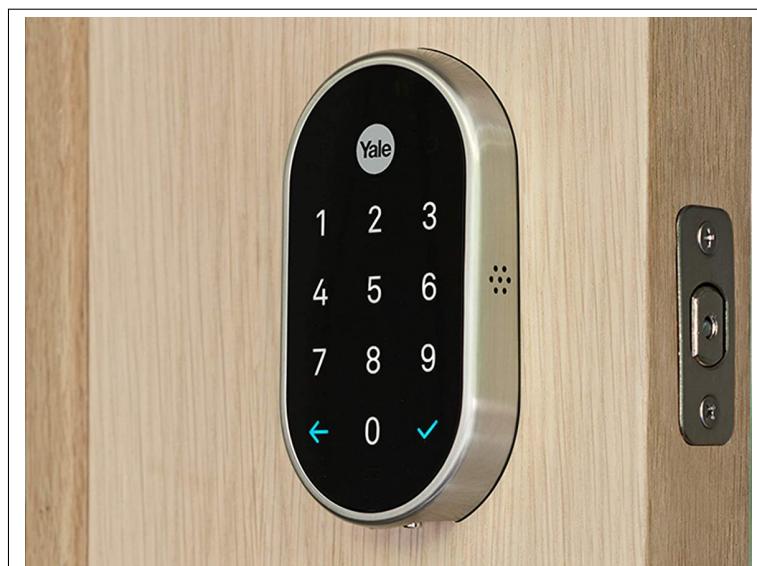


Figura 2.3: Next x Yale Lock [17]

Avantaje:

- Permite accesul prin intermediul unui PIN ales de utilizator
- Oferă alerte când cineva închide sau deschide ușa
- Oferă integrare cu Google Home și Nest Home

Dezavantaje:

- Are nevoie de 4 baterii tip AA pentru a funcționa
- Nu are acces cu cheie sau cartelă
- Nu are versiune rezistentă

2.2.3 Level Lock - Touch Edition

Level Lock este o încuietoare inteligentă de tip zăvor. Are un design minimalist și ascunde partea electronică în interiorul ușii pentru mai multă securitate.



Figura 2.4: Level Lock [10]

Avantaje:

- Multiple modalități de acces, printre care: amprentă, PIN
- Oferă alerte când cineva încuietoarea se închide sau deschide ușa
- Oferă integrare cu Google Home și Nest Home

2.2.4 Comparații

Produsele de mai sus adreseză probleme ușor diferite, dar încearcă să ofere funcționalități similare. Sistemul oferit de Videx Security prezintă un design rezistent, dar familiar tuturor utilizatorilor și este destinat clădirilor cu mai mulți locatari. În contrast, cele două încuietori inteligente oferă o integrare avansată în rețeaua IoT și multiple căi de acces, dar sunt destinate unei singure locuințe.

Încuietoarea de la Yale prezintă cea mai inovativă abordare a acestui design prin decizia deliberată de a nu oferi posibilitatea de acces cu cheie. Astfel, simplifică partea mecanică eliminând singura cale de acces din exterior către mecanismul încuietorii.

Produsul celor de la Videx Security se bazează pe o tehnologie utilizată la scară largă și prin urmare beneficiază de robustețea unui sistem matur. Spre deosebire de celelalte două produse analizate, soluția celor de la Videx Security este agnostică de sistemul de operare al telefonului mobil, având nevoie doar de o conexiune GSM.

Din lipsa unor standarde în domeniu, dispozitivele noi suferă de alte tipuri de probleme și vulnerabilități, după un studiu realizat de cercetătorii de la Bitdefender. Majoritatea sunt în faza inițială de setare, oferind protocoale de securitate învechite sau omitându-le complet. Este menționat și un dispozitiv care expune un port Telnet, un protocol învechit și ușor de exploatat, fără posibilitatea de a fi dezactivat [7].

2.3 Stabilirea cerințelor funcționale și nefuncționale ale sistemului

Cerințele funcționale ale acestui sistem vor fi împărțite conform tabelului următor, ele fiind detaliate mai jos.

Cerințe funcționale (F)	Cerințe nefuncționale (NF)
F1. Controlul accesului în apartament	NF1. Funcție răspuns automat
F2. Expunerea unui serviciu REST	NF2. Control granular asupra datelor
F3. Dezvoltare client Android	NF3. Expunerea unui flux duplex VoIP
F4. Criptarea informațiilor transmise	
F5. Managementul accesului la sistem	

Tabelul 2.2: Tabel stabilire cerințe

2.3.1 (F1) Controlul accesului în apartament

Scopul principal al acestui sistem este de a oferi sau nu acces într-o incintă, prin urmare consider aceasta cea mai importantă cerință funcțională.

2.3.2 (F2) Expunerea unui serviciu REST pentru interfațarea cu alte sisteme

Expunerea și abstractizarea terminalului **POTS** este realizată printr-un set de servicii **Representational State Transfer (REST)** care controlează starea sa. Acest lucru ne permite interfațarea cu aplicația mobilă, interfața de administrare web și alte servicii precum Google Home/Google Assistant/Apple HomeKit.

2.3.3 (F3) Dezvoltarea unui client mobil Android

Principalul client care va interacționa cu serviciile **REST** va fi aplicația mobilă ce vă avea rolul de a notifica userul în eventualitatea declansării soneriei interfonul și de a controla starea sistemului.

2.3.4 (F4) Criptarea comunicațiilor cu serviciile web

Având în vedere nivelul de acces pe care l-ar oferi o exploatare a unei vulnerabilitati al acestei soluții, comunicațiile între server și clienți trebuie realizate printr-un canal criptat de tip **Secure Sockets Layer (SSL)**. Credentialele userului și ulterior tokenul de acces trebuie trimise doar după verificarea autenticității serverului și a pachetelor trimise.

2.3.5 (F5) Oferirea și revocarea accesului la sistem

Dorim de exemplu să oferim acces necondiționat unui prieten apropiat pentru a intra în bloc fară a mai suna la interfon. De asemenea ar trebui să putem realiza și inversul acestei operații.

2.3.6 (NF1) Implementarea unei funcții pentru răspuns automat

Această funcție vă permite utilizatorului să stabilească o perioadă de timp pentru care sistemul vă oferi accesul necondiționat.

2.3.7 (NF2) Control granular asupra datelor stocate

Arhitectura aplicației necesită interacțiunea cu o bază de date, care poate fi ținută în cloud, pentru convenabilitate sau local. Folosind tehnologii de containerizare precum Docker, putem stoca baza de date local, informațiile fiind stocate într-un mediu controlat.

2.3.8 (NF3) Expunerea unui flux duplex audio prin tehnologia VoIP

Pasul final în dezvoltarea acestui sistem ar fi interfațarea cu un [Analog to Digital Convertor \(ADC\)](#) și un [Digital to Analog Convertor \(DAC\)](#) și expunerea streamurilor de date prin [Voice Over IP \(VoIP\)](#)

3 Stadiul actual in domeniu și selectarea soluției tehnice

3.1 Stadiul actual al tehnologiilor utilizate pentru dezvoltarea soluției

Dezvoltarea unui sistem IoT de automatizare presupune atat o parte hardware cat si una software. Pentru controlarea hardware-ului de la distanta este nevoie de un canal de comunicatii prin care sa se i trimita comenzi. In general, in cazul sistemelor embedded de acest tip folosesc un microcontroller sau un microprocesor care implementeaza stiva IP.

O alta abordare populara in proiectarea acestor sisteme este dezvoltarea unui controller conectat la internet care are rolul de a colecta informatii de la alte dispozitive din incinta compatibile cu protocolul sau. Mai departe, informatiile colectate sunt transmise unui server spre a fi preprocesate, aggregate, oferind utilizatorului date relevante momentului respectiv.

In functie de complexitatea solutiei, partea responsabila pentru procesarea evenimentelor poate varia de la un simplu server conectat la o baza de date pana la un cluster de big-data compus din sute de noduri capabile sa ruleze algoritmi de agregare distribuiti.

3.1.1 Apple, Amazon, Google

Potrivit articolului [4], Apple foloseste Apache Mesos, un manager open-source pentru clustere de computatie capabil sa scaleze pana la zeci de mii de noduri pentru a rula serviciile necesare asistentului inteligent Siri intr-o maniera care ofera redundanta la eroare. Urmatorul nivel de integrare vine de la compania Amazon, care ruleaza algoritmi asistentului sau inteligent Alexa pe platforma sa de servicii web, [Amazon Web Services \(AWS\)](#). Intr-o maniera similara putem specula ca o companie precum Google foloseste tehnologia sa de orchesterare pentru clustere de computatie, Kubernetes, pentru a rula serviciile necesare Google Assistant.

Toate aceste solutii includ integrari cu sisteme IoT precum lumini inteligente, aspiratoare autonome sau incuietori inteligente au o complexitate ridicata, justificand necesitatea unui cluster computational distribuit.

3.1.2 Espressif

Espressif Systems ofera o abordare alternativa problemei, prin protocolul de comunicatii Espressif care compacteaza 5 layer din stiva **Open Systems Interconnection (OSI)** intr-unul singur, reducand latenta cauzata de pierderea pachetelor in retele congestionate. Fiind mai simplu, iroseste mai putini cicli ai microprocesorului si consuma mai putina memorie. Pentru a permite interactiunea senzorilor si actorilor Espressif cu dispozitive mobile care nu implementeaza acest protocol este nevoie de un gateway care sa realizeze traducerea pachetelor intre cele doua retele, insa acest lucru este optional.

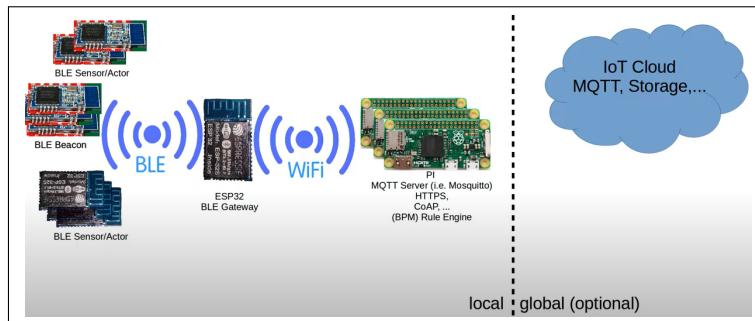


Figura 3.1: Sistem interconectare actori Espressif la internet [11]

Din perspectiva utilizatorului, dispozitivele noi necesita un pas de imperechere in retea, dupa acest pas fiind complet autonome. Chiar daca frameworkul ofera functii ajutatoare pentru criptarea informatiilor transmise, aplicatiile pot alege sa implementeze metoda standard Curve25519, sa isi implementeze propriul mecanism sau chiar sa il dezactiveze complet.

Compania din spate ofera printre altele si o familie de microcontrolere numita ESP, gandita pentru a accelera procesul de dezvoltare a noi senzori si actori in retea, oferind o gama larga in materie de conectivitate.

3.1.3 Solutia hobbyista

Proiectandu-ne propriul sistem, beneficiem de libertate in modelarea problemei si alegerea protocoalelor de comunicare. Asadar, se poate concepe un sistem **IoT** care sa ofere un set de functionalitati mai restrans, folosind hardware disponibil consumatorilor de rand si tehnologii software open-source.

Aditional, pentru o integrare minimala cu unul din asistentii personali mentionati mai sus, de obicei este pus la dispozitia dezvoltatorilor un API bazat pe webhook-uri. Aceasta sarcina presupune implementarea unor servicii **REST** pe baza unor specificatii prestabilite.

3.2 Prezentarea tehnologiilor si platformelor de dezvoltare alese

3.2.1 Hardware

Deoarece proiectul necesita atat interacciunea cu sisteme electrice cat si cu sisteme digitale precum stiva [IP](#), am ales placa de dezvoltare "Raspberry Pi 3 Model B Rev 1.2". Aceasta ofera un procesor quad core cu arhitectura armv7 de 1.2 Ghz, 1 GB RAM si 26 de pini [General-Purpose Input/Output \(GPIO\)](#) pentru interacciunea cu terminalul [POTS](#).

Considerand complexitatea relativ scazuta a circuitului electric, pentru proiectarea [PCB](#) am ales Fritzing, un soft open-source de [Computer Assisted Design \(CAD\)](#). Spre deosebire de un program mai profesionist precum Eagle, Fritzing este usor de folosit si dispune de o librerie care contine majoritatea componentelor analogice si digitale. In cazul in care nu exista model pentru o componenta, utilizatorul are posibilitatea de a crea un model din poze si masuratori.

3.2.2 Backend

Intr-un studiu anual realizat de Stack Overflow, peste 80,000 de dezvoltatori software au ales JavaScript ca cel mai folosit limbaj de programare pentru al noualea an consecutiv. NodeJS a urcat pe locul 5 in popularitate, in timp ce Typescript este pe locul 6. Datorita cerintei de portabilitate am ales NodeJS ca limbaj pentru implementarea serverului aplicatiei. [13]

Printre alternative viabile pentru acest tip de proiect se numara Java, C# sau Python, limbaje aflate in primele 10 in topul celor de la Stack Overflow.

Ca framework de dezvoltare a serverului am ales NestJS, oferind o arhitectura [Model View Controller \(MVC\)](#) si multe functionalitati convenabile precum:

- Framework de injectare a dependintelor: graful (aciclic) de dependinte al aplicatiei este calculat la pornire, fiecarui modul ii sunt satisfacute dependintele, instantiindu-se obiectele necesare o singura data. Daca sunt detectati cicli in graful de dependinte sau nu exista informatii despre cum se poate instantia o clasa, atunci se va arunca o eroare de runtime si aplicatia va iesi cu un status code de eroare.
- Separarea logicii de control a aplicatiei de interfata si de date. Utilizatorul interactioneaza cu interfata, care notifica controllerul de actiunile utilizatorului, controllerul executa logica aplicatiei si actualizeaza modelul corespunzator, schimbari ce se vor reflecta in interfata.
- Imbina elemente de [Object Oriented Programming \(OOP\)](#), [Functional Programming \(FP\)](#) si [Functional Reactive Programming \(FRP\)](#). De exemplu: modulele si serviciile sunt clase, iar decoratorii claselor sunt functii care modifica comportamentul functiilor adnotate prin computere.

3.2.3 Baza de date

Din punct de vedere al scalabilitatii, pradigma relationala scaleaza vertical (putine servere puternice), pe cand cea nerelationala este orizontala (multe servere mici). Prin urmare am ales MongoDB, o solutie de tip NoSQL rulata in modul "cluster" pentru a oferi redundanta datelor prin replicarea lor de 3 ori pe noduri diferite fizic.

Deoarece MongoDB are nevoie de suport pentru 64 biti, nu poate fi instalata pe acelasi Raspberry Pi unde va rula si serverul. Pentru simplitudine, am ales un serviciu online de hosting gratis, numit Mongo Atlas. Asadar, serverul NodeJS trebuie sa tina cont de eventuala latenta mai ridicata in comunicarea cu baza de date si retransmiterea comenzilor in cazul in care niciunul din nodurile clusterului nu este disponibil.

Object Document Mapping

Pentru transformarea si validarea obiectelor de JavaScript in documente ce vor fi stocate in baza de date, am ales Mongoose.

3.2.4 Firebase Cloud Messaging

Pentru transmiterea notificarilor si evenimentelor in timp real catre dispozitive mobile s-au luat in calcul mai multe solutii de tip Pub/Sub, insa solutia finala aleasa a fost Firebase, datorita stabilitatii ridicata prin integrarea stransa cu sistemul de operare. In spate, se folosesc Google Play Services pentru a se livra informatiile utilizatorului intr-o maniera eficienta, sistemul putand sa interzie usor evenimentele pentru trezi cat mai putin procesul dispozitivului notificat [9].

Alte solutii investigate au fost OneSignal si MQTT, cel din urma fiind folosit de Facebook in aplicatia lor de mesagerie [19]. Acest protocol a fost proiectat pentru a trimite date de telemetrie de la sonde din spatiu, mentionand consumul de baterie si latime de banda la minim. Chiar daca acest sistem prezinta anumite avantaje, complexitatea ridicata de implementare a fost factorul care a influentat decizia finala de a de a folosi Firebase.

3.2.5 Client

Android este o platforma mobile care s-a maturizat pe parcursul a 12 versiuni majore si principalul competitor de piata al iOS. Avand experienta anterioara ca programator Android si in special cu limbajul de programare Java, a fost o alegere convenabila pentru un prototip rapid. Este de mentionat ca aceasta alegere de platforma este pur subiectiva, un client similar putand fi dezvoltat pentru iOS sau cu o tehnologie care suporta cross-compilation cum ar fi React Native sau Ionic.

4 Considerente legate de implementarea soluției tehnice

4.1 Arhitectura sistemului

Sistemul prezentat presupune atat o partare hardware, cat si una software. Hardwareul realizeaza adaptarea dintre terminalul analog POTS si placa digitala de dezvoltare Raspberry Pi, iar ca software am folosit NodeJS pentru server si Android pentru a implementa un client al serverului.

Cu ajutorul multimetrului am dedus schema electrica a tastaturii si am gasit contactele care sunt conectate in cazul apasarii butonului de pe ultimul rand, coloana din mijloc. Scurtcircuitarea contactorului lamelar care depisteaza ridicarea receptorului si legarea la difuzorul terminalului au fost realizate mai usor, circuitul fiind usor de parcurs vizual.



Figura 4.1: Depanare si interfatare terminal POTS

1 - contacte buton deschidere, 2 - contactor lamelar receptor, 3 - contacte difuzor

4.1.1 Raspberry Pi HAT

Dupa ce etapa de prototipare pe breadboard a fost finalizata, am transcris schema electrica a circuitului cu ajutorul softwareul Fritzing. Proiectarea unui Printed Circuit

Board (PCB) reprezinta penultimul pas inainte de etapa de productie in masa. Printre parametrii improtanti in deciderea designului unui circuit printat se numara:

- Tehnologia de montare a componentelor pe **PCB** (Through Hole Technology (**THT**) sau Surface Mounted Device (**SMD**))
- Numarul de straturi de circuit (alegeri comune sunt 2, 4, insa dispozitive complexe precum placile video pot folosi pana la 12 straturi)
- Grosimea si culoarea placii de fibra de sticla
- Latimea unui traseu pe placa
- Distanța minima intre trasee
- Diametrul via-urilor (gauri verticale in placa folosite pentru a conecta straturile)
- Materialul folosit pentru lipire si metериалul folosit pentru pinii de interfatare

Pentru a proiecta un **Hardware attached on top (HAT)** compatibil cu Raspberry Pi, am folosit softwareul Fritzing. Acesta permite proiectarea schemei electrice si ulterior trasarea conexiunilor pe layoutul fizic al placii. Considerand complexitatea redusa a proiectului, am ales sa folosesc doua o placă cu doua straturi, impreuna cu urmatorii parametri pe care i-am introdus in Fritzing ca si constrangeri de design:

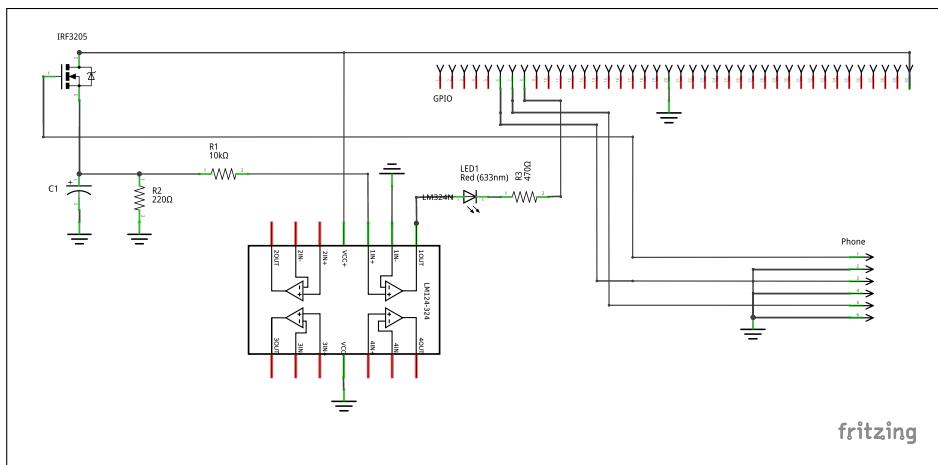


Figura 4.2: Schema electrica HAT Raspberry PI

Actionarea butoanelor terminalului **POTS** se realizeaza cu ajutorul unor optocuploare, izoland circuitul interfonului care este proiectat pentru a functiona cu spike-uri de pana la 90V de circuitul Raspberry Pi.

Detectarea unui apel este realizata prin legarea unui **Metal Oxide Semiconductor Field Effect Transistor (MOSFET)** la bornele difuzorului terminalului **POTS** si inserierea cu un amplificator operational in regim de comparator cu referinta de 0.1V. Am folosit de asemenea si un Filtru Trece Jos deoarece terminalul este sensibil la zgomote, declansand accidental notificarea.

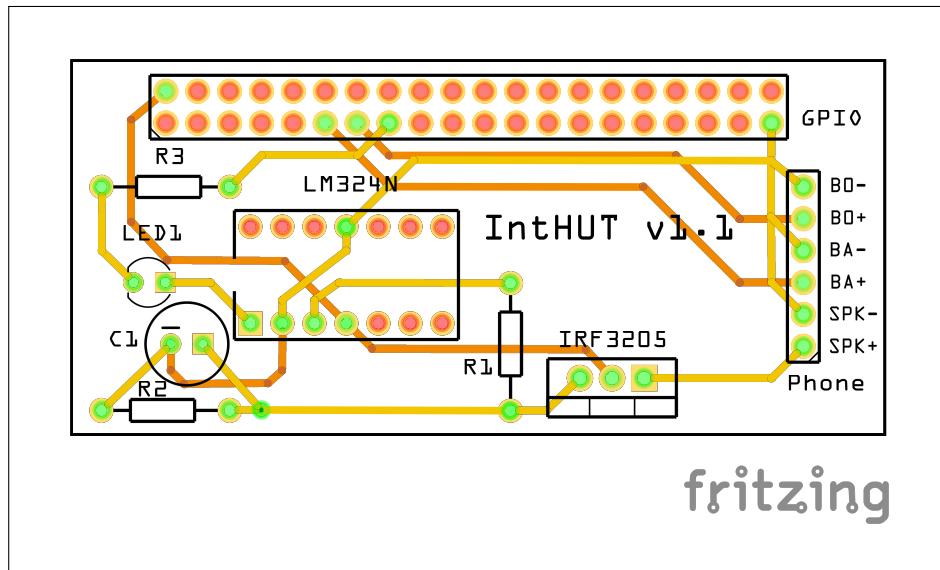


Figura 4.3: Design PCB HAT
(galben - stratul de sus, portocaliu - stratul de jos)

Optocuploare improvizate

Datorita crizei globale de semiconductori, o placuta breakout care include doua optocuploare costa aproximativ 50 RON. Considerand simplitatea functionarii acestui circuit, am decis sa construiesc propria solutie, folosind componente de baza: un rezistor pentru limitat curentul, un LED si un fotorezistor. Platforma Raspberry Pi furnizeaza pinilor sai [GPIO](#) 3.3V si un curent maxim de 16mA, iar un LED rosu are o cadere de tensiune de 2.4V:

$$\begin{aligned}
 I_{GPIO} &= 10 \text{ mA} = 10^{-2} \text{ A} \\
 V_R &= V_{GPIO} - V_{LED} = 3.3V - 2.4V = 0.9V \\
 I_{GPIO} &= \frac{V_R}{R} \text{ sau } R = \frac{V_R}{I_{GPIO}} = \frac{0.9V}{10^{-2}A} = 90\Omega
 \end{aligned} \tag{4.1}$$

Am lipit rezistorul de 90Ω la anodul ledului si am incastrat LED-ul impreuna cu fotorezistorul intr-o incinta fara lumina.

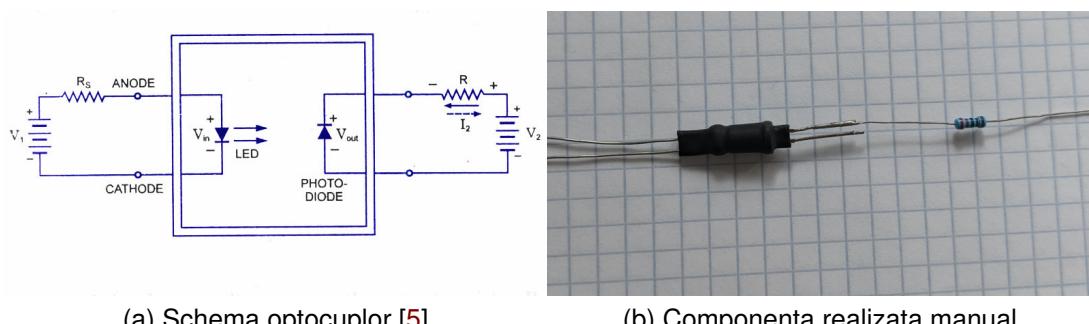


Figura 4.4: Schema electrica optocuplora (a) si rezultat ansablu (b)

Deoarece aveam nevoie să scurtcircuitez un contactor pe partea terminalului **POTS** am omis rezistența legată fotorezistorului. Atunci când ledul este aprins, rezistența fotorezistorului scade, comportându-se aproape ca un conductor ideal, atingând lamelele contactorului corespunzător receptorului.

Dupa ce toate traseele au fost puse, ultimul pas este umplerea spațiului ramas pe fiecare strat cu un plan legat la GND pentru a reduce emisiile electromagnetice. Înainte de a trimite fisierul Gerber spre a fi produs, am folosit constrangerile definite initial pentru a valida proiectul, asigurându-ne că acesta poate fi produs în realitate.

Nr. straturi	Tehnologie	Grosime traseu	Latime via	Diametru via	Material finisaj
2	THT	1.6mm	1mm	0.5mm	LeadFree HASL-RoHS

Tabelul 4.1: Parametri aleși pentru fabricarea PCB

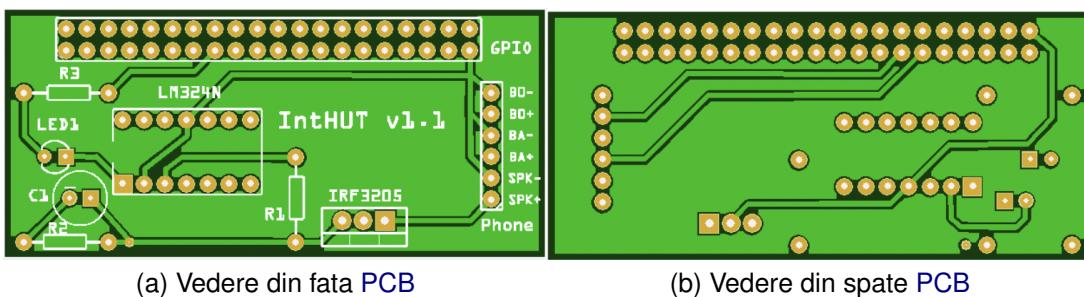


Figura 4.5: Randare placuta înainte de comandat

4.1.2 Webserver NodeJS

NodeJS este un mediu de rulare JavaScript asincron, cu un design centrat în jurul unei bucle de evenimente. Este proiectat pentru realizarea aplicațiilor scalabile care comunică prin rețea. [12]

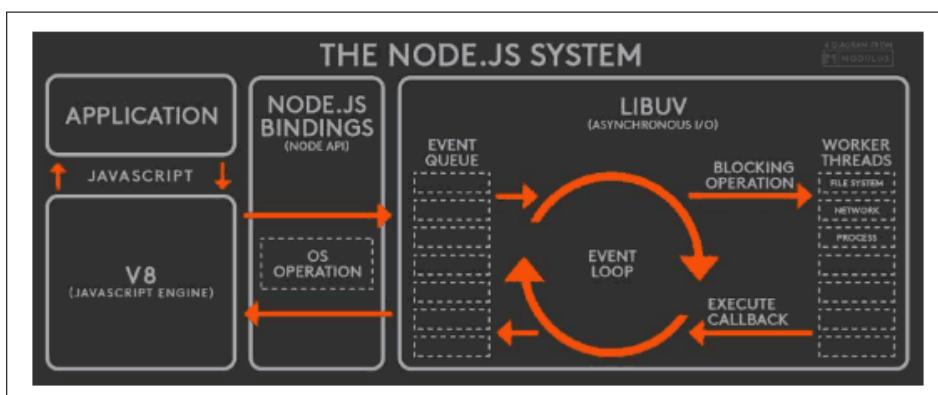


Figura 4.6: Ilustrare event loop și callback queue [14]

Pentru a înțelege cum rulează codul javascript în acest mediu de rulare, trebuie să ne familiarizăm cu modelul de asincroneitate pus la dispozitie, care difera semnificativ

de cel multi-threaded. Codul JavaScript este parcurs cu ajutorul engine-ului V8 și este tradus în apeluri la libraria nativa "libuv" care se va ocupa de executarea lor și plasarea în coada de evenimente. Atunci când "libuv" are nevoie să facă un apel sincron de sistem, precum interogarea unui server DNS, o face prin intermediul unui thread-pool, atașând un callback pentru a fi chemat atunci când este gata.

Criptare HTTPS

Conform cerintelor funktionale, canalul de comunicare dintre client și server trebuie să fie securizat cu un protocol **Hypertext Transfer Protocol Secure (HTTPS)** pentru a permite trimitera credențialelor fără a fi susceptibile la atacuri de tip **Man in the Middle (MITM)**. S-a ales folosirea Let's Encrypt, un serviciu gratuit pentru emiterea unui certificat semnat de o autoritate reputabilă.

Pentru a verifica identitatea serverului, clientului îi este cerut să afiseze un sir de date la o rută statică prestabilită, dovedind astfel autoritatea asupra serverului căruia îi va fi emis certificatul. După completarea acestor pași și emiterea certificatului, configurarea serverului să implementeze protocolul **HTTPS** este relativ simplă:

```
const httpsConfig: IHttpsOptions | undefined = config.getExpressConfig().https;
const options: NestApplicationOptions | undefined = process.env.NODE_ENV === 'production' && httpsConfig != null
    ? {
        httpsOptions: {
            key: readFileSync(httpsConfig.keyPath),
            cert: readFileSync(httpsConfig.certPath),
        }
    }
    : undefined;

const app: NestExpressApplication = await NestFactory.create<NestExpressApplication>(AppModule, options);
```

Figura 4.7: Configurare aplicatie NestJs pentru a folosi certificate

Este de menționat că începând cu versiunea de Android 9 (Pie) ca parte dintr-o inițiativă de marire a securității, sistemul de operare va bloca conexiunile PLAINTEXT dacă aplicațiile nu configurează o excepție pentru server [6]. De asemenea, prezența unor certificate self-signed sau insuficiente informații vor produce rezultate similare, fiind nevoie ca serverul să prezinte întregul lanț de certificate (al serverului, ale autoritatilor intermediare și în final cel al autoritatii centrale).

Autentificare

Pentru autentificarea și validarea credențialelor utilizatorilor, am ales metoda **JSON Web Token (JWT)**, un standard open source în industrie conform RFC 7519. În cea mai compactă formă să să, acesta este compus din trei parti, despartite prin caracterul ":".

1. Header (algoritm și tipul tokenului)
2. Continut (id utilizator, nume, rol)

3. Semnatura

Semnatura este calculata cu ajutorul unui algoritm simetric de hashing [HMAC SHA-256 \(HS256\)](#) si a unei chei private aplicat pe forma codata in baza 64 a continutului si metadatelor despre token (expirare, emitator, audienta, subiect). Toate cele trei informatii sunt apoi concatenate cu caracterul "." intre, constituind forma finala ce va trimisa clientului. Clientul va transmite acest token in comunicatiile ulterioare cu serverul, acesta folosindu-se de mesaj, semnatura si cheia privata pentru a determina daca a fost sau nu modificat pe parcurs.

Folosirea algoritmului simetric implica faptul ca secretul este utilizat atat pentru generarea de tokenuri, cat si pentru validarea lor. In cazul aplicatiei din lucrare, aceasta nu este o problema, deoarece ambele metode ruleaza in interiorul aceluiasi proces, fara sa expuna aplicatia la vulnerabilitati.

Un avantaj al [JWT](#) este faptul ca serverul nu trebuie sa intretina starea unei tabele de sesiuni a utilizatorilor. In esenta daca un utilizator este in posesia unui token ne-expirat, acesta este considerat autentificat. Prin urmare, mai multe instante ale serverului pot valida in paralel identitatea utilizatorilor, fara nevoia de a interoga o baza de date sau o memorie cache partajata, permitand astfel scalarea pe orizontala a aplicatiei.

NestJS

Construit peste cel mai popular framwork pentru aplicatii web disponibil pentru Node.js, NestJS imbunatatesta experienta dezvoltarii serviciilor web folosind functii experimentale din Typescript care permit interpretarea la transpilare a decoratorilor pentru metode si clase. De asemenea urmareste un design [MVC](#) si are pachete intreținute oficial pentru taskuri comune, cum ar fi conectarea la o baza de date sau proiectarea unui mecanism de autentificare si verificare a identitatii utilizatorilor.

Mutex

Din natura asincrona a limbajului si posibilitatea sistemului de a avea mai mult de un utilizator, trebuie luat in considerare cazul in care mai multi utilizatori incearca sa interactioneze cu sistemul in acelasi timp. Asadar, trebuie implementat un mecanism similar semaforului binar, numit mutex. Diferenta dintre acestea fiind ca in cazul mutexului, doar detinatorul original poate sa il elibereze spre a fi folosit din nou.

Acet comportament este dorit pentru a informa clientii serviciului in cazul in care requestul nu poate fi satisfacut deoarece resursa este ocupata de altcineva.

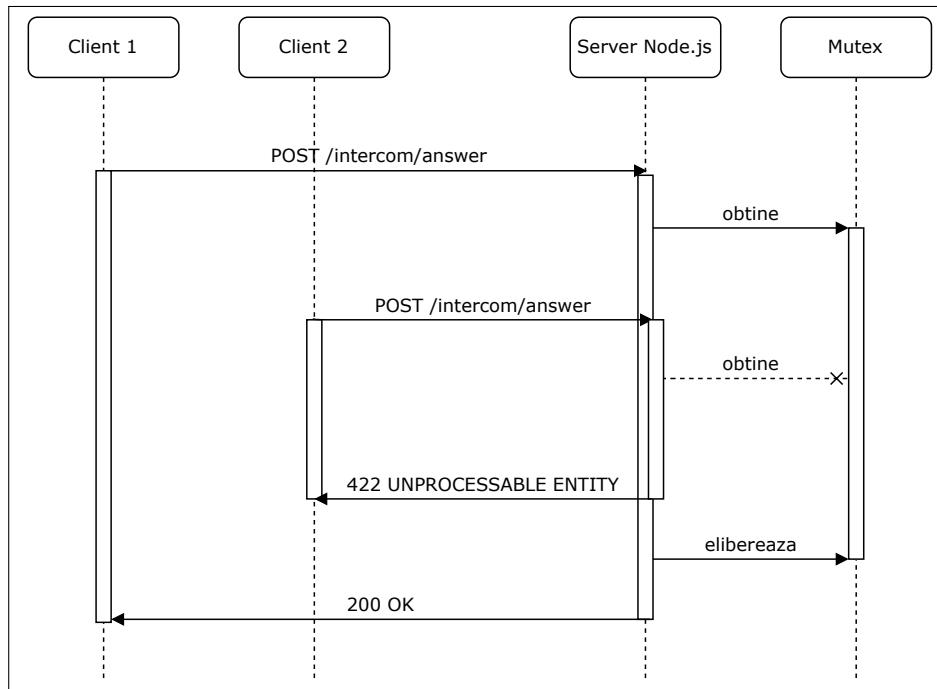


Figura 4.8: Ilustrare resursa disputata intre doi utilizatori

Validarea entitatilor

Foloseste functii din Typescript si NestJS pentru a adauga si citi metadate prin intermediul decoratorilor de metode. [Data Transfer Object \(DTO\)](#) este reprezentarea unui model din baza de date, dar encodat favorabil pentru interpretarea usoara a clientilor. Majoritar, acesta va contine mai putine campuri decat exista in baza de date (reprzentarea unui utilizator nu va avea campul pentru parola).

Campurile unui [Data Transfer Object \(DTO\)](#) sunt anotate cu tipul asteptat la runtime, si printr-un interceptor de nest care ruleaza atunci cand este invocata metoda unui controller [REST](#), obiectul primit ca parametru este verificat contra specificatiilor din metadate. In cazul in care validarea esueaza, clientului ii este intors status 400 (Bad Request) indicand o eroare de formatare a requestului.

Mutand responsabilitatea verificarii entitatilor in cadrul serverului, se mitigheaza si lipsa unei scheme la nivelul bazei de date, inerente solutiilor NoSQL. Prin urmare se reduce riscul unor inconsistente in datele stocate.

Roluri

Urmardind "reteta" de dezvoltare observata in validarea entitatilor, am creat un mecanism de adaugare a metadatelor pe rutele controllerelor despre ce roluri de user au acces sa le cheme.

Partea de verificare a unui user in timpul unui request, revine asa-numitelor "Guard-uri" care implementeaza o interfata comună. In cazul vederilor pentru aplicatia de admin dormin sa restrictionam afisarea sa userilor normali, asadar inlantuim

”Guardurile” ce garanteaza autentificarea unui utilizator si rolul de administrator. In cazul in care un user nu este administrator, acestuia i se intoarce un cod de status 403 (Forbidden) - serverul a autentificat utilizatorul, dar acesta nu are suficiente permisiuni pentru a accesa resursa

Documentatie

Intr-un proiect de lunga durata, documentatia joaca un rol esential in usurinta de menținanta si dezvoltare a codului. Despartirea logica a componentelor aplicatiei cat si explicarea eventualelor cai logice si raspunsuri posibile ajuta atat clientii externi consumatori ai API-ului cat si dezvoltatorii noi care incearca sa introduca primele modificari.

Astfel, am ales Swagger pentru a parurge proiectul si genera pagini de documentatie pentru toate rutele serviciului REST, impreuna cu comentarii si potentiile status code-uri la care trebuie sa se astepte clientii. De asemenea, Swagger include si un client REST in pagina, impreuna cu schema obiectelor si tipurile de date asteptate nu lasa loc de interpretat in comportamentul serverului.

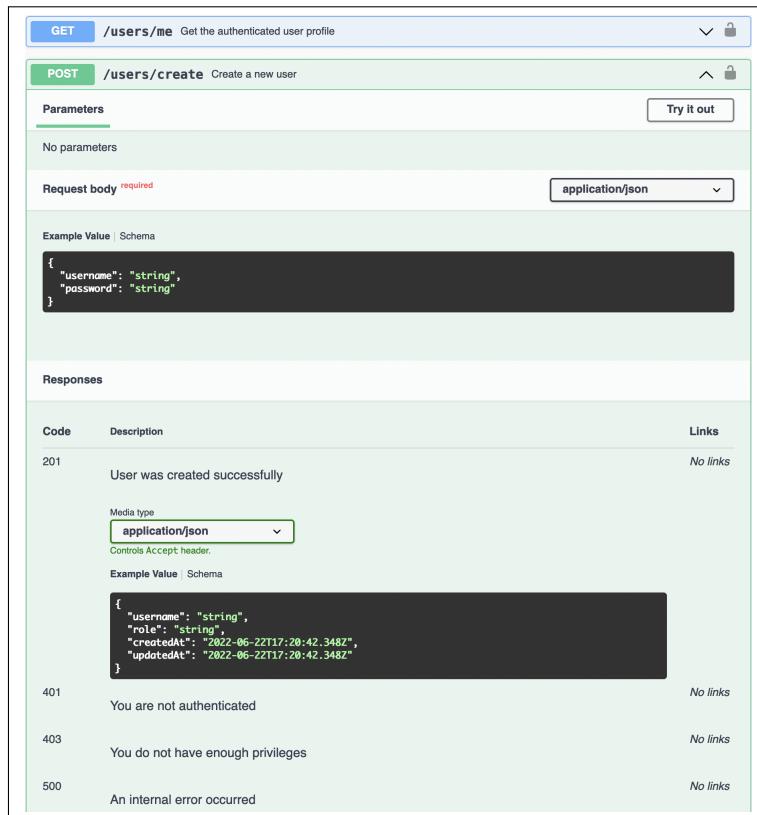


Figura 4.9: Documentatie generata pentru ruta /users/create

4.1.3 Android

Android este o platforma mobila care s-a maturizat pe parcusul a 12 versiuni majore si principalul competitor de piata al iOS. In dezvoltarea acestei aplicatii am folosit o abordare similara cu cea a serverului, fiind organizata intr-un pattern de design MVC.

O aplicatie Android poate fi alcatauita din mai multe componente cu roluri specifice: Activity, Service si BroadcastReceiver. Fiind un sistem de operare care vizeaza dispozitivele mobile, este optimizat pentru folosirea eficienta a resurselor si a bateriei, astfel ca folosirea incorecta a componentelor anterioare poate duce la oprirea accidentalala a aplicatiei sau a unui serviciu in timpul unor operatiuni importante.

O alta particularitate in programarea acestor aplicatii este ca desenatul interfelei si interceptarea evenimentelor de la utilizator se realizeaza intr-o bucla pe firul de executie principal. Astfel, orice operatiuni aditionale execute pe acest thread trebuie alese cu atentie pentru a nu duce la aparenta ingreunare prin introducerea latentei la afisaj si la interpretat evenimente de input de la utilizator. Operatiuni care presupun asteptarea dupa sisteme de [IO](#) precum un apel prin retea, sunt complet blocate din a fi execute pe threadul principal, aruncand o exceptie numita "NetworkOnMainThreadException" [8].

Dependency Injection

Pentru a gestiona mai usor modulele aplicatiei si diferitele surse de date, am ales sa folosesc Dagger2, un framework bine cunoscut de Java. El vine cu extensii pentru Android ce permit controlul granular asupra instantierii modulelor necesare in functie de ciclul de viata al aplicatiei sau al unui Activity. Astfel putem defini module globale, precum cel care cheama api-ul web, instantiat o singura data pe durata aplicatiei, sau module locale, precum cel de SharedPreferences care se realoca de fiecare data cand se intra in ecranul principal.

Pe langa organizarea proiectului in module logice in functie de functionalitati, Dagger ajuta si la managementul memoriei intr-un limbaj de programare cu Garbage Collector, precum Java, evitand alocari neneccesare sau frecvente.

4.2 Implementarea sistemului

4.2.1 Server

Chiar daca autentificarea prin interfata grafica de administrator beneficiaza de acelasi standard de securitate ca si ceilalti clienti, am ales sa o expun pe un port diferit de cel al [API-ului](#), permitand flexibilitate maxima utilizatorului final in alegerea expunerii serviciilor. Personal, am expus serviciul de API pe un port rutat public si interfata pe un port privat, blocat din firewall. In esenta, orice utilizator normal poate interactiona cu sistemul atata timp cat este autentificat, insa interfata de administrare este disponibila doar din interiorul retelei din acasa.

Secventa de generare a comenziilor pentru deschiderea interfonului se afla in clasa IntercomService, aceasta secventa include: obtinerea mutexului, ridicarea receptorului, asteptarea pentru o secunda, apasarea butonului pentru raspuns de doua ori, cu o pauza de 1.2s intre si in final eliberarea mutexului. Secventa pentru respingerea unui apel a fost implementata intr-o maniera similara, presupunand doar ridicarea receptorului si inchiderea sa dupa un interval arbitrar de timp.

```

public async answer(): Promise<void> {
    const lock: LMLockSuccessData = await this.mutexService.acquireLock({ key: "status" });

    rpio.write(this.PIN_HOOK, rpio.HIGH);
    await Sleep( ms: 1000 );
    rpio.write(this.PIN_ZERO, rpio.HIGH);
    await Sleep( ms: 1000 );
    rpio.write(this.PIN_ZERO, rpio.LOW);
    await Sleep( ms: 1200 );
    rpio.write(this.PIN_ZERO, rpio.HIGH);
    await Sleep( ms: 500 );
    rpio.write(this.PIN_HOOK, rpio.LOW);
    rpio.write(this.PIN_ZERO, rpio.LOW);

    await this.mutexService.releaseLock(lock);
}
    
```

Figura 4.10: Implementarea metodei pentru permis accesul din IntercomService

4.2.2 Baza de date

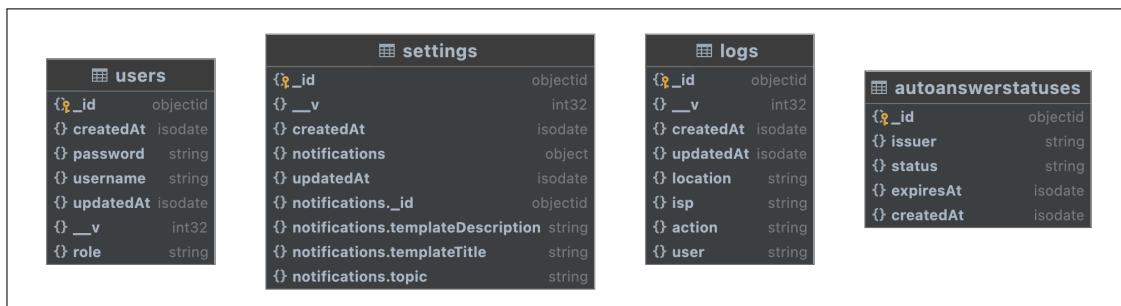


Figura 4.11: Schema bazei de date Mongo

Baza de date nu ofera nicio relatie intre documente, insa colectia "logs" sau "autoanswerstatuses" trebuie sa fie legate din punct de vedere logic cu un utilizator. Putem din nou sa beneficiem de adnotari, specificand tipul asteptat pentru validare sau o referinta catre o alta colectie.

Salvam astfel id-ul utilizatorului care le-a generat impreuna cu ele, iar la momentul interogarii bazei de date putem folosi metoda "populate()" din driverul Mongoose care va cauta modelul pentru adnotari de tip referinta. In final, driverul va genera query-uri pentru a aduce "referintele" din colectiile corespunzatoare. Daca o referinta nu este gasita, se va intoarce "null" si este responsabilitatea aplicatiei sa trateze acest caz.

4.2.3 Aplicatie mobila

Pentru a evita complet problema rularii apelurilor de retea pe threadul gresit, am folosit un client REST numit Retrofit, bazat pe OkHttp. Aceasta se ocupa sa coordoneze un ThreadPool (o colectie de threaduri ce executa taskuri de acelasi tip), astfel ca atunci cand aplicatia cheama un serviciu, se alege unul din threaduri spre a se executa cererea, raspunsul venind sub forma unui callback pe threadul principal.

```

@Module
public class RemoteModule {
    1 usage  ± ialex
    @Provides
    @Singleton
    ApiAuthenticator provideApiAuthenticator(PrefsRepository preference) { return new ApiAuthenticator(preference); }

    1 usage  ± ialex
    @Provides
    @Singleton
    TokenInterceptor provideTokenInterceptor(PrefsRepository preference) { return new TokenInterceptor(preference); }

    1 usage  ± Ionescu Alexandru +1
    @Provides
    @Singleton
    OkHttpClient provideOkHttpClient(ApiAuthenticator authenticator, TokenInterceptor interceptor) {
        HttpLoggingInterceptor loggingInterceptor = new HttpLoggingInterceptor();
        loggingInterceptor.setLevel(HttpLoggingInterceptor.Level.BODY);
        return new OkHttpClient.Builder()
            .addInterceptor(loggingInterceptor)
            .followRedirects(false)
            .followSslRedirects( followProtocolRedirects: false )
            .connectTimeout( timeout: 30, TimeUnit.SECONDS )
            .readTimeout( timeout: 30, TimeUnit.SECONDS )
            .authenticator(authenticator)
            .addInterceptor(interceptor)
            .build();
    }
}

```

Figura 4.12: Configurare modul retea, impreuna cu dependințele sale directe

Datorita adnotarilor `@Singleton` toate obiectele din acest modul vor fi instantiatate o singura data pe durata aplicatiei, oferind un punct de acces centralizat pentru toate operatiile ce implica chemarea serviciilor REST.

De asemenea, am configurat Retrofit sa foloseasca Gson pentru serializarea si deserializarea **DTO**-urilor din format **JSON** in instante ale claselor din Java. Pentru aceasta operatiune se foloseste de **API**-ul reflectiv oferit de limbajul Java spre a chama constructorul implicit al unei clase si a seta valorile variabilelor sale.

4.3 Testarea sistemului

4.3.1 Server

Chiar daca NestJS ofera suport pentru scrierea de teste unitare prin adaugarea de fisere `.spec.ts`, cea mai mare provocare a fost mockuirea modulului care comunica cu pinii **GPIO** ai Raspberry Pi. Wrapperul de JavaScript comunica prin intermediul **Node-API (N-API)** cu o librarie statica ce realizeaza serializarea si deserializarea datelor dintre Node.js/V8 VM si C/C++. Aceasta librarie se linkeaza la randul ei cu `bcm2835` pentru a obtine acces la pinii **GPIO** si alte functii din sistemul de **Input/Output (IO)** al cipului Broadcom 2835.

Astfel suntem prezentati cu o problema, libraria **N-API** poate fi compilata pe alte arhitecturi, dar cu siguranta va genera o exceptie la rulare cand dependinta sa, `bcm2835`, va incerca sa execute instructiuni invalide. Este nevoie de o logica de control care sa permita rularea si returnarea unor date prestabilite, cand se detecteaza rularea pe o arhitectura invalida, cum ar fi in cazul testelor rulate in mod automat prin intermediul pipelineului de integrare continua.

Teste unitare

Urmatorul pas a fost elaborarea testelor unitare. Am ales să creez un fisier `.spec.ts` pentru fiecare controller al API-ului acoperind astfel întreaga funcționalitate a aplicației. Pentru controllerului responsabil deschiderii interfonului am mockuit serviciile Firebase și baza de date, testând mecanismul de deschidere/inchidere a interfonului în cazul în care aceeași resursă este accesată de mai mulți utilizatori concomitent.

Teste end-to-end

Pentru a ne asigura că serviciul **REST** răspunde corect se impune necesitatea testelor cap-coadă, unde se instantiază aplicația într-o manieră similară producției, iar cu ajutorul unui client REST se trimit cereri prestaibile și se așteaptă după răspunsurile lor. Astfel parcurgem toată logica serverului, cap-coadă și ne vom asigura de un comportament predictibil.

4.3.2 Aplicatie mobila

Din cauza fragmentării foarte mari a versiunilor și configurațiilor posibile pentru toate dispozitivele Android pe care ar putea rula aplicația, am decis să folosesc "Firebase Test Lab", un serviciu de la Google care oferă posibilitatea rularii unui test automat sau scriptat pe o gamă largă de dispozitive. Asigurăm astfel un minim control al calității prin descoperirea timpurie a excepțiilor neînțăratate.

Tipul de test ales este cel "Robo", care va analiza inteligent interfața vizuală a aplicației, după care va genera evenimente de input ca și cum ar fi un utilizator. Planul "Spark" de la Firebase ne da acces la 5 rulari pe dispozitive fizice pe zi și 10 dispozitive virtuale, deci testelete au fost executate în limita acestor constrainte.

Pentru a rula un testă, trebuie să încarcăm un APK sau un AAB, alegem dispozitivele pe care dorim să le testăm și eventualii pași adiționali pe care îi dorim acoperiți de Robo. După introducerea ID-ului pentru campurile de username și parola împreună cu valorile asociate unui cont de test, a reușit să se autentifice și să facă un stress-test al aplicației. Următoarele dispozitive de test au fost alese:

- SM-F926U1, API Level 30 - telefon pliabil, care prezintă o dimensiune non-standard și provocări unice în ceea ce privește adaptarea interfețelor pentru o experiență cât mai placută
- Nexus 5, API Level 23 - versiune mai veche, dar foarte populară de Android, rezoluție standard FullHD, acest test se asigură că aplicația este compatibilă cu versiuni anterioare ale sistemului de operare
- SM-T720, API Level 28 - tabletă, ne asigurăm că aplicația este folosibilă pe un ecran cu diagonala mare

Metrica	Valoare
Druata pana la prima afisare	700ms
Eveniment VSync Pierdut	3%
Latenta ridicata input	0%
Thread UI incet	4%
Comanda draw inceata	1%
Incarcare bitmap inceata	0%

Tabelul 4.2: Metrice de performanta raportate pentru Nexus 5

Analizand tabelul pentru performanta vizuala, observam ca aplicatia ruleaza in margini acceptabile pana si pe un dispozitiv mai vechi precum Nexus 5.

De asemenea "Test Lab" poate fi chemat din interiorul pipelineul de integrare continua pentru a rula teste automat dupa terminarea unui build pe un anumit branch, verificand flow-uri exesitente din aplicatie impotriva potențialelor regresii aparute.

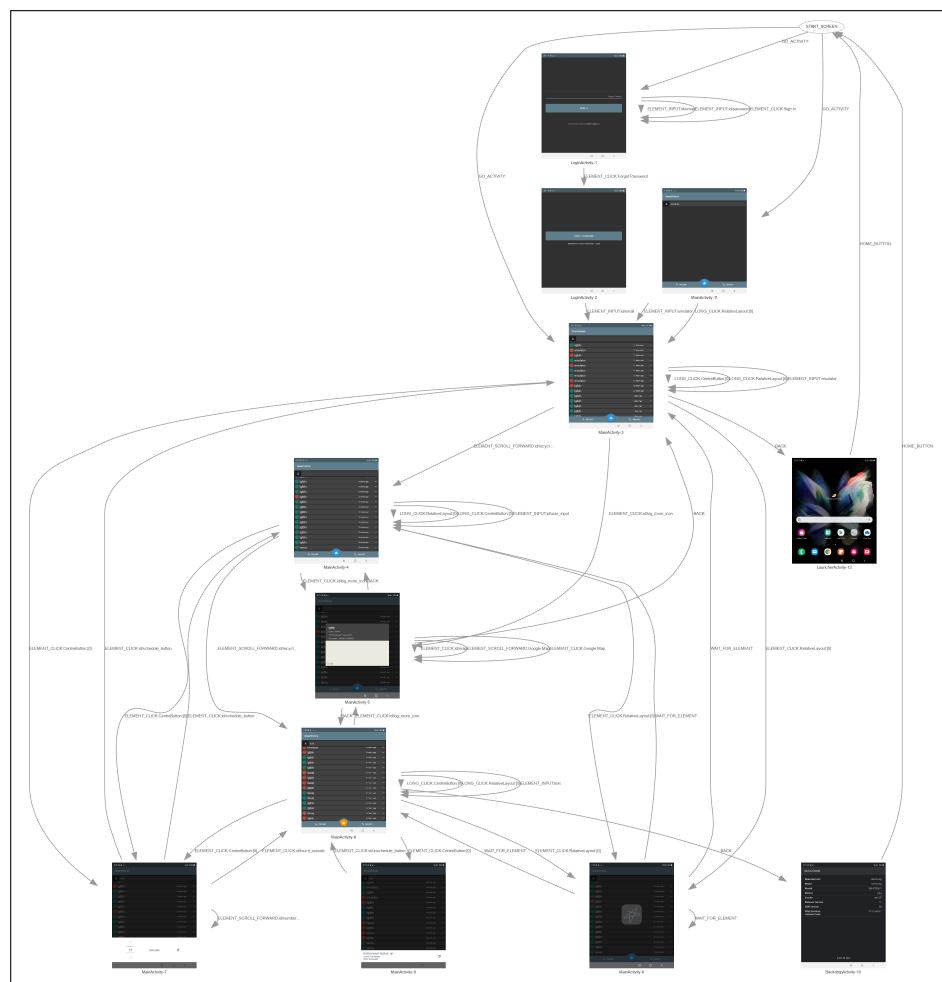


Figura 4.13: Flow testare Robo pe Samsung SM-F926U1

5 Studiu de caz

5.1 Oferirea accesului unui utilizator

Operatia de oferire a accesului unui nou utilizator presupune folosirea interfetei de administrare. Dupa setarea numeului noului utilizator si a parolei, noul cont este gata pentru folosit, iar pentru convenabilitate codul QR generat poate fi scanat cu aplicatia pentru a se autentifica automat.

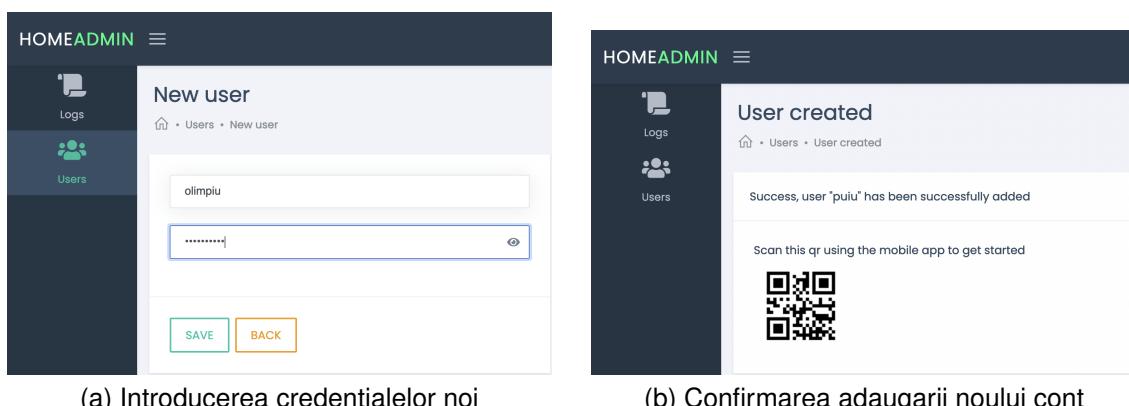


Figura 5.1: Creare utilizator nou

Dupa descarcarea aplicatiei din Google Play Store, se va putea realiza autentificarea. Un alt avantaj al unui astfel de sistem este eliminarea nevoii cartelelor fizice, usor pierdut sau incurcat.

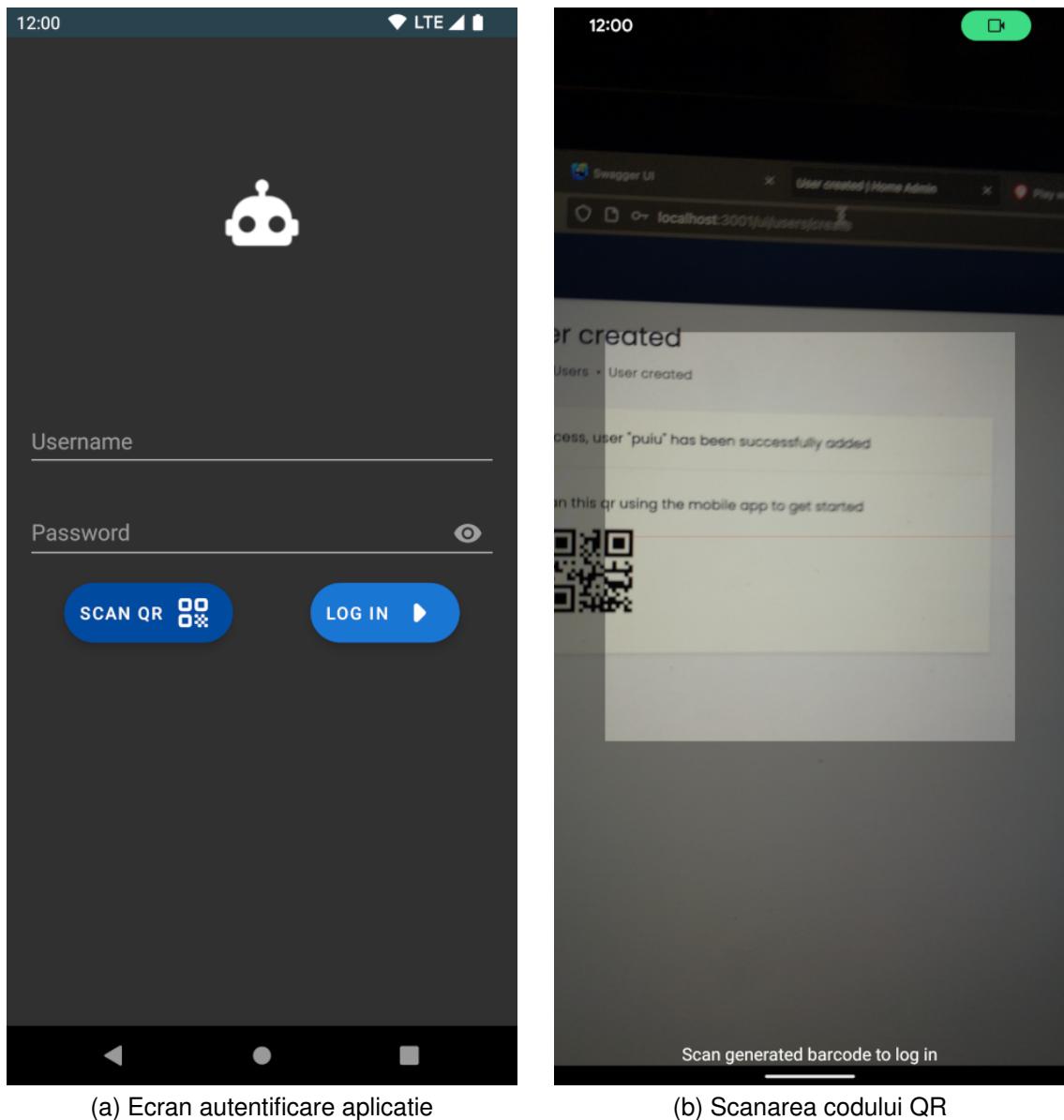


Figura 5.2: Autentificare prin scanare cod QR

5.2 Raspuns automat

Majoritatea vizitelor si apelurilor la interfon sunt previzibile, de exemplu venirea curierului sau a unui postas este anuntata printr-un interval aproximativ de timp. Deoarece utilizatorul duce o viata ocupata si stie ca urmeaza sa fie sunat, poate programa sistemul sa raspunda si sa deschida automat usa la urmatorul apel pentru o durata predefinita.

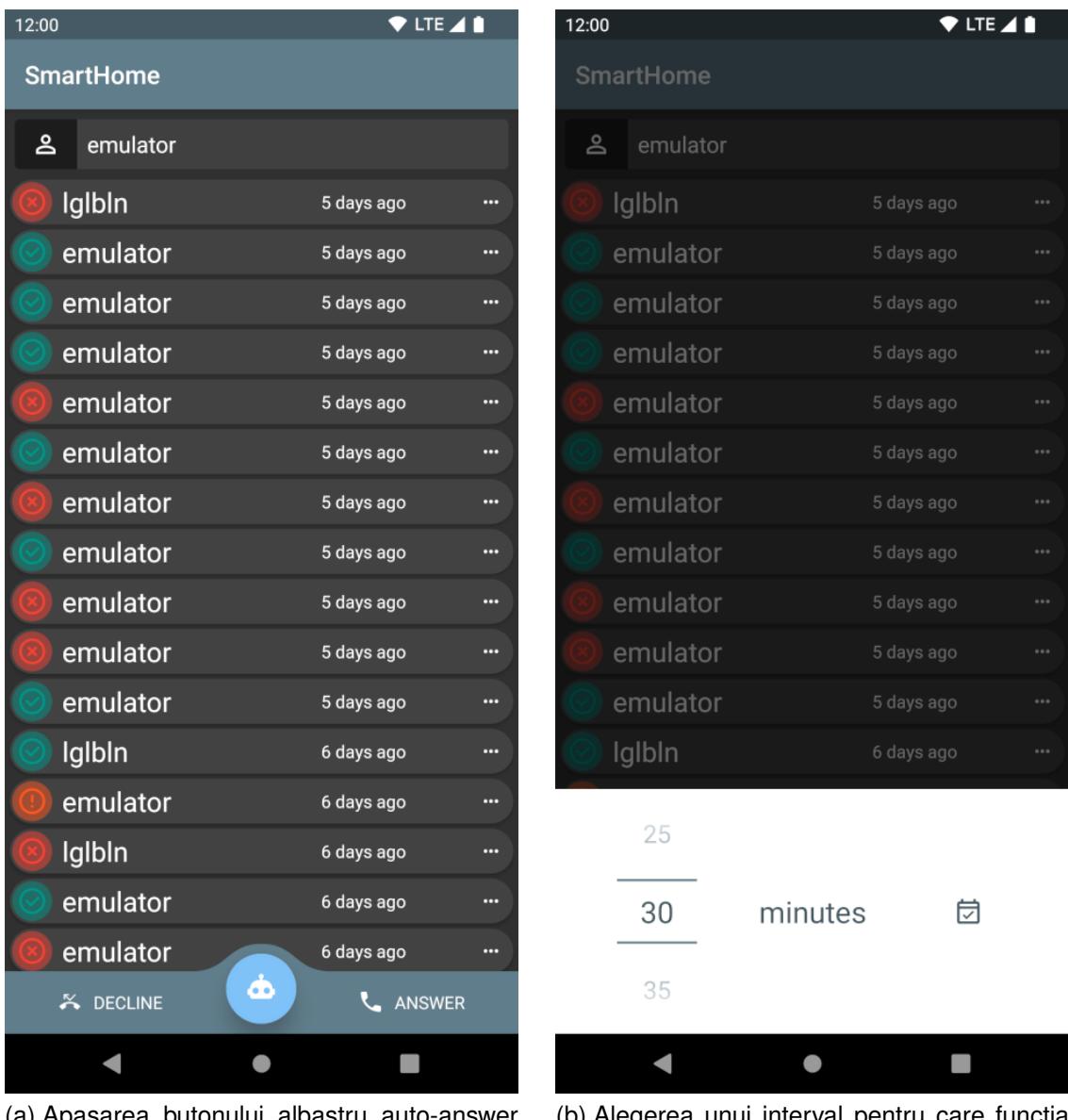


Figura 5.3: Pasi pentru setare functie auto-answer

Se confirma activarea functiei prin colorarea butonului auto-answer in portocaliu. Intr-o maniera similara, daca se doreste oprirea inainte de termen, se poate face acest lucru din meniul anterior.

5.3 Raspuns de la distanta

Prin conectarea la reteaua IoT sistemul este expus Internetului, acesta fiind avantajul solutiei, dar si unul din factorii limitanti ai sai. Posibilitatea de deschidere de oriunde clientul are un dispozitiv conectat la Internet inseamna ca in lipsa cartelei standard

Radio-Frequency Identification (RFID) se poate apela propriul apartament. Se va folosi aplicatia mobila pentru a se raspunde la propriul apel.

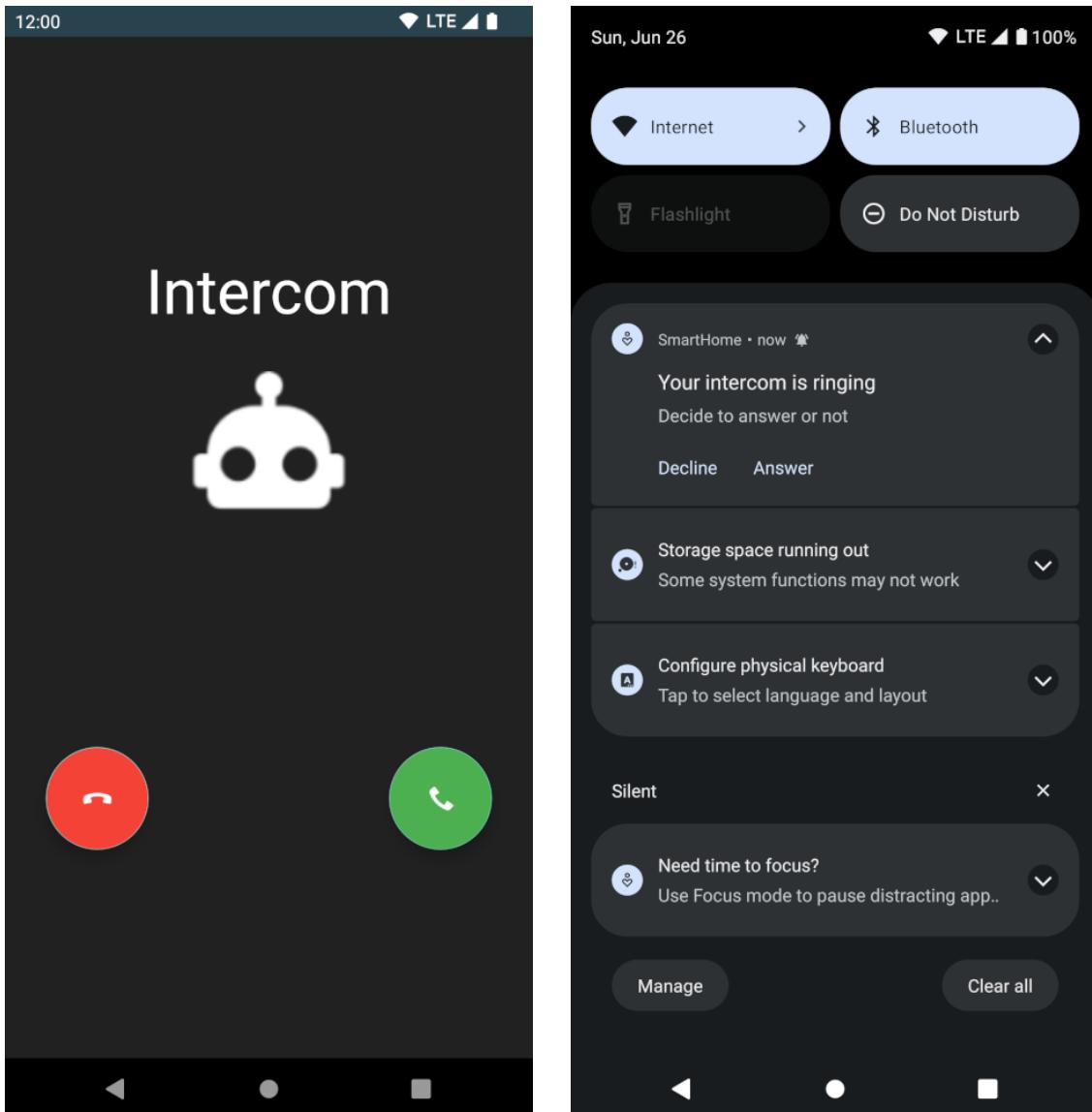


Figura 5.4: Ecran apel

O alta varianta pentru mitigarea acestei probleme a fost folosirea unui tag **Near Field Communication (NFC)** care contine codat url-ul de acces impreuna cu un token care nu expira. Astfel prin scanarea lui cu un telefon ce dispune de anena **NFC** se va realiza request-ul din browserul default al utilizatorului, rezultand in deschiderea interfonului.

6 Concluzii

6.1 Concluzii

Avand in vedere informatiile expuse in capitolele 1, 2, 3 si cunostintele dobandite in realizarea arhitecturii unui sistem de tip incuietoare inteligenta, dar si rezultatele studiului de caz demonstreaza ca se pot crea solutii folosind tehnologii open-source. Prin alegeri sensibile din punct de vedere tehnic in ceea ce priveste securitatea sistemului, utilizatorul normal poate folosi aplicatia pentru a isi controla interfonul intr-un mod sigur. De asemenea, includerea tehnologiilor open-source inseamna ca utilizatorii avansati au multiple alegeri in ceea ce priveste implementarea solutiei (pot alege unde sunt stocate datele si cine are acces la ele, pot configura din firewall accesul la [API](#) si la interfata administrativa).

Prin urmare, aceasta lucrare arata ca se poate concepe o solutie IoT la nivelul standardelor secolului 21, atat din punct de vedere al securitatii cat si al proprietatii datelor stocate. Integrarea sa usoara poate doar sa beneficieze adoptarii mai rapide a acestui sistem.

6.2 Contributii

- Sinteza informatiilor in ceea ce priveste nisa incuietorilor inteligente.
- Analiza solutiilor existente din domeniu pentru intelegherea pietii.
- Conceperea arhitecturii unui sistem similar care sa satisfaca cerintele functionale.
- S-a demonstrat ca se pot folosi componente electronice simple pentru a realiza un optocuplaj mai ieftin, dar care ofera aceiasi functionalitate in cazul acestei aplicatii.
- Implementarea fizica a acestui concept si testarea in viata de zi cu zi.

6.3 Dezvoltari ulterioare

Ca dezvoltari ulterioare ale sistemului ar presupune construirea intreaga a unui terminal [POTS](#) ca [HAT](#) pentru RaspberryPi. Astfel vom putea atinge si cerinta de a oferi utilizatorului fluxuri audio pentru a comunica cu persoana de la celalalt capat al interfonului.

7 Bibliografie

- [1] Tanweer Alam. "A Reliable Communication Framework and Its Use in Internet of Things (IoT)". În: 3 (mai 2018).
- [2] Frances K Aldrich. "Smart homes: past, present and future". În: *Inside the smart home*. Springer, 2003, pag. 18–18.
- [3] Nattapol Aunsri. "A bayesian filtering approach with time-frequency representation for corrupted dual tone multi frequency identification". În: 24 (ian. 2016), pag. 370–377.
- [4] Daniel Bryants. *Apple Rebuilds Siri Backend Services Using Apache Mesos*. 2015. URL: <https://www.infoq.com/news/2015/05/mesos-powers-apple-siri/> (vizitat 13/06/2022).
- [5] CircuitsToday. *Optocoupler*. 2009. URL: <https://www.circuitstoday.com/wp-content/uploads/2009/08/optocoupler.jpg> (vizitat 17/06/2022).
- [6] digicert. *Android P will default to https connections for all apps*. 2018. URL: <https://www.digicert.com/blog/android-p-will-default-https-connections-apps> (vizitat 24/06/2022).
- [7] Alexandra Gheorghe. *The Internet of Things: Risks in the connected home*. Research Paper BD-NGZ-86847. Bitdefender, 2016.
- [8] Google. *Android Developers Documentation*. 2022. URL: <https://developer.android.com/reference/android/os/NetworkOnMainThreadException> (vizitat 26/06/2022).
- [9] Google. *Firebase Cloud Messaging Documentation*. 2022. URL: <https://firebase.google.com/docs/cloud-messaging> (vizitat 26/06/2022).
- [10] Level Home. *Level Lock: The Smallest and Most Advanced Smart Lock Ever*. 2019. URL: <https://level.co/products/lock> (vizitat 29/04/2022).
- [11] Nina. *Why use mqtt server for BLE gateway?* 2021. URL: <https://stackoverflow.com/questions/68195682/why-use-mqtt-server-for-ble-gateway> (vizitat 05/06/2022).
- [12] Node.js. *About Node.js*. 2021. URL: <https://nodejs.org/en/about/> (vizitat 22/06/2022).
- [13] Stack Overflow. *Stack Overflow Developer Survey 2021*. 2021. URL: <https://insights.stackoverflow.com/survey/2021#section-most-popular-technologies-programming-scripting-and-markup-languages> (vizitat 05/06/2022).
- [14] Develop Paper. *What's the difference between browsers and Node's Event Loop?* 2019. URL: <https://developpaper.com/whats-the-difference-between-browsers-and-nodes-event-loop/> (vizitat 23/06/2022).

- [15] Biljana L. Risteska Stojkoska și Kire V. Trivodaliev. “A review of Internet of Things for smart home: Challenges and solutions”. În: *Journal of Cleaner Production* 140 (2017), pag. 1454–1464. ISSN: 0959-6526. DOI: <https://doi.org/10.1016/j.jclepro.2016.10.006>. URL: <https://www.sciencedirect.com/science/article/pii/S095965261631589X>.
- [16] Videx Security. *GSM Intercoms*. 2016. URL: <https://www.videxuk.com/system/gsm-intercoms> (vizitat 28/04/2022).
- [17] Google Store. *Nest x Yale Lock*. 2018. URL: https://store.google.com/us/product/nest_x_yale_lock?hl=en-US (vizitat 28/04/2022).
- [18] Zeus Integrated Systems. *A Brief History of Smart Home Automation*. 2019. URL: <https://zeusintegrated.com/blog/item/a-brief-history-of-smart-home-automation> (vizitat 02/06/2022).
- [19] Lucy Zhang. *Building Facebook Messenger*. 2021. URL: <https://www.facebook.com/notes/10158791547142200/> (vizitat 26/06/2022).