



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

Coordonatori
Prof. Dr. Ing. Mihnea Alexandru MOISESCU
As. Drd. Ing. Miruna Elena ILIUTĂ

Bucureşti, 2022

Cuprins

1	Introducere	1
1.1	Obiectivele lucrării de licență	1
1.2	Descrierea domeniului din care face parte proiectul de diplomă	1
1.3	Prezentare pe scurt a capitolelor	2
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 și 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	26
5	Studiu de caz	29
5.1	Oferirea accesului unui utilizator	29
5.2	Răspuns automat	30
5.3	Răspuns de la distanță	31
6	Concluzii	33
6.1	Concluzii	33
6.2	Contribuții	33
6.3	Dezvoltări ulterioare	33
7	Bibliografie	34

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 prezinta 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 exista un segment de piață de utilizatori care ar dori să beneficieze de funcțiile telefonului inteligent, dar nu au această posibilitatea deoarece ar presupune schimbarea sistemului din întreaga clădire.

Sistemul propus în această lucrare se poate conecta la rețeaua Plain Old Telephone Service (POTS) printr-o simplă mufă RJ11 lucru 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]

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 apărițe 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 [20].

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, concomitant reducând preturile soluțiilor oferite. În ciuda acestor semne, sociologii au concluzionat la vremea respectivă că nu există un interes real în conceptul "Smart House".

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 capitolele 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 componența software alcătuită din server și clienti. 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 va analiza soluții digitale, însă sistemul final va 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 va 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:

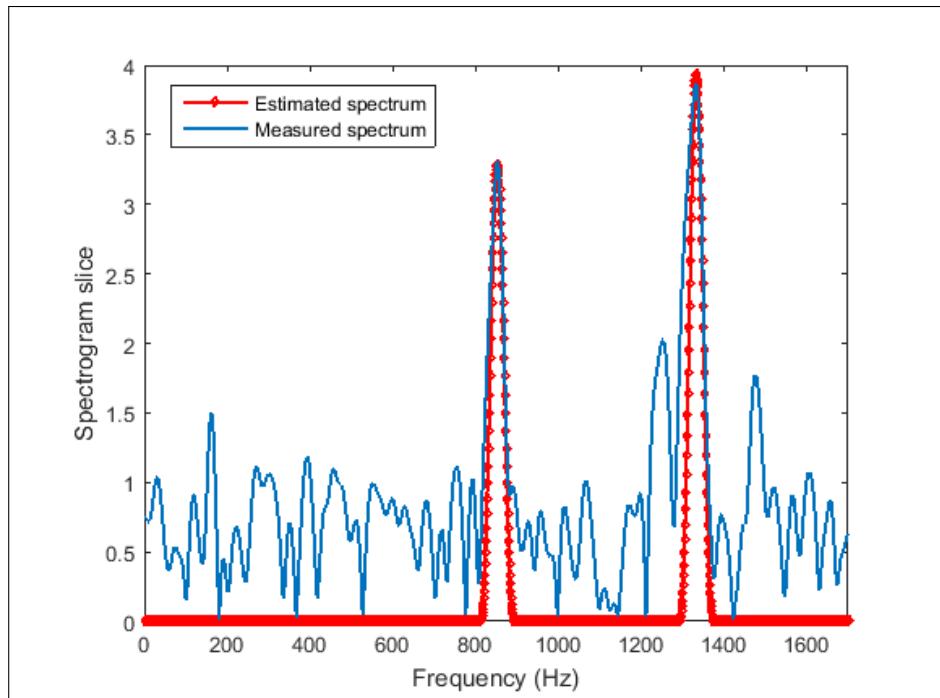


Figura 2.1: Spectrogramă tasta "8" [3]

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 [16].

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 [19]

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

(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.

(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.

(F3) Dezvoltarea unui client mobil Android

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

(F4) Criptarea comunicărilor cu serviciile web

Având în vedere nivelul de acces pe care l-ar oferi o exploatare a unei vulnerabilități al acestei soluții, comunicările între server și clienti 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.

(F5) Oferirea și revocarea accesului la sistem

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

(NF1) Implementarea unei funcții pentru răspuns automat

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

(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.

(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 în domeniu și selectarea soluției tehnice

3.1 Stadiul actual al tehnologiilor utilizate pentru dezvoltarea soluției

Dezvoltarea unui sistem IoT de automatizare presupune atât o parte hardware cât și una software. Pentru controlarea hardware-ului de la distanță este nevoie de un canal de comunicații prin care să se i trimită comenzi. În general, în cazul sistemelor embedded de acest tip folosesc un microcontroller sau un microprocesor care implementează stiva IP.

O altă abordare populară în proiectarea acestor sisteme este dezvoltarea unui controller conectat la internet care are rolul de a colecta informații de la alte dispozitive din incintă compatibile cu protocolul sau. Mai departe, informațiile colectate sunt transmise unui server spre a fi preprocesate, agregate, oferind utilizatorului date relevante momentului respectiv.

În funcție de complexitatea soluției, partea responsabilă pentru procesarea evenimentelor poate varia de la un simplu server conectat la o bază de date până la un cluster de big-data compus din sute de noduri capabile să ruleze algoritmi de agregare distribuți.

3.1.1 Apple, Amazon, Google

Potrivit articolului [4], Apple folosește Apache Mesos, un manager open-source pentru clustere de computație capabil să scaleze până la zeci de mii de noduri pentru a rula serviciile necesare asistentului inteligent Siri într-o manieră care oferă redundanță la eroare. Următorul nivel de integrare vine de la compania Amazon, care rulează algoritmi asistențului sau inteligent Alexa pe platforma sa de servicii web, [Amazon Web Services \(AWS\)](#). Într-o manieră similară putem specula că o companie precum Google folosește tehnologia sa de orchestration pentru clustere de computație, Kubernetes, pentru a rula serviciile necesare Google Assistant.

Toate aceste soluții includ integrări cu sisteme IoT precum lumini inteligente, aspiratoare autonome sau încuietori inteligente au o complexitate ridicată, justificând nevoie de un cluster computațional distribuit.

3.1.2 Espressif

Espressif Systems oferă o abordare alternativă problemei, prin protocolul de comunicații Espressif care compactează 5 layer din stiva **Open Systems Interconnection (OSI)** într-unul singur, reducând latenta cauzată de pierderea pachetelor în rețele congestionate. Fiind mai simplu, îroșește mai puțini cicli ai microprocesorului și consumă mai puțină memorie. Pentru a permite interacțiunea senzorilor și actorilor Espressif cu dispozitive mobile care nu implementează acest protocol este nevoie de un gateway care să realizeze traducerea pachetelor între cele două rețele, însă acest lucru este optional.

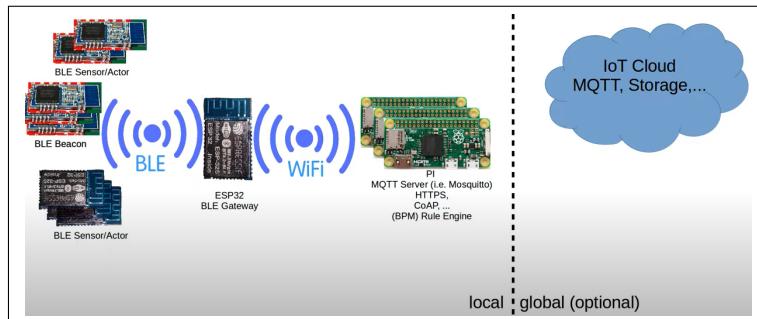


Figura 3.1: Sistem interconectare actori Espressif la internet [11]

Din perspectiva utilizatorului, dispozitivele noi necesită un pas de împerechere în rețea, după fiind complet autonome. Chiar dacă frameworkul oferă funcții ajutătoare pentru criptarea informațiilor transmise, aplicațiile pot alege să implementeze metoda standard Curve25519, să își implementeze propriul mecanism sau chiar să îl dezactiveze complet.

Compania din spate oferă printre altele și o familie de microcontrolere numită ESP, gândită pentru a accelera procesul de dezvoltare a noi senzori și actori în rețea, oferind o gamă largă în materie de conectivitate.

3.1.3 Soluția hobbyista

Proiectandu-ne propriul sistem, beneficiem de libertate în modelarea problemei și alegerea protocolelor de comunicare. Așadar, se poate concepe un sistem IoT care să ofere un set de funcționalități mai restrâns, folosind hardware disponibil consumatorilor de rând și tehnologii software open-source.

Adițional, pentru o integrare minimală cu unul din asistenții personali menționați mai sus, de obicei este pus la dispoziția dezvoltatorilor un API bazat pe webhook-uri. Aceasta sarcină presupune implementarea unor servicii REST pe baza unor specificații prestabilite.

3.2 Prezentarea tehnologiilor și platformelor de dezvoltare alese

3.2.1 Hardware

Deoarece proiectul necesită atât interacțiunea cu sisteme electrice cât și cu sisteme digitale precum stiva IP, am ales placă de dezvoltare "Raspberry Pi 3 Model B Rev 1.2". Aceasta oferă un procesor quad core cu arhitectură armv7 de 1.2 Ghz, 1 GB RAM și 26 de pini General-Purpose Input/Output (GPIO) pentru interacțiunea cu terminalul POTS.

Considerând complexitatea relativ scăzută 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 ușor de folosit și dispune de o librărie care conține majoritatea componentelor analogice și digitale. În cazul în care nu există model pentru o componentă, utilizatorul are posibilitatea de a crea un model din poze și măsurători.

3.2.2 Backend

Într-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 nouălea an consecutiv. NodeJS a urcat pe locul 5 în popularitate, în timp ce Typescript este pe locul 6. Datorită cerinței de portabilitate am ales NodeJS ca limbaj pentru implementarea serverului aplicației. [13]

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

Ca framework de dezvoltare a serverului am ales NestJS, oferind o arhitectură Model View Controller (MVC) și multe funcționalități convenabile precum:

- Framework de injectare a dependințelor: graful (aciclic) de dependințe al aplicației este calculat la pornire, fiecărui modul îi sunt satisfăcute dependințele, instantiindu-se obiectele necesare o singura data. Dacă sunt detectați cicli în graful de dependințe sau nu există informații despre cum se poate instantia o clasa, atunci se va arunca o eroare de runtime și aplicația va ieși cu un status code de eroare.
- Separarea logicii de control a aplicației de interfață și de date. Utilizatorul interacționează cu interfață, care notifică controllerul de acțiunile utilizatorului, controllerul execută logica aplicației și actualizează modelul corespunzător, schimbări ce se vor reflecta în interfață.
- Îmbină elemente de Object Oriented Programming (OOP), Functional Programming (FP) și Functional Reactive Programming (FRP). De exemplu: modulele și serviciile sunt clase, iar decoratorii claselor sunt funcții care modifică comportamentul funcțiilor anotate prin compunere.

3.2.3 Baza de date

Din punct de vedere al scalabilității, pradigma relațională scalează vertical (puține servere puternice), pe când cea nerelațională este orizontală (multe servere mici). Prin urmare se va utiliza MongoDB, o soluție de tip NoSQL rulată în modul "cluster" pentru a oferi redundanță datelor prin replicarea lor de 3 ori pe noduri diferite fizic.

Deoarece MongoDB are nevoie de suport pentru 64 biți, nu poate fi instalată pe același Raspberry Pi unde va rula și serverul. Pentru simplitudine, s-a ales un serviciu online de hosting gratis, numit Mongo Atlas. Așadar, serverul NodeJS trebuie să țină cont de eventuala latență mai ridicată în comunicarea cu baza de date și retransmiterea comenziilor în cazul în care niciunul din nodurile clusterului nu este disponibil.

Object Document Mapping

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

3.2.4 Firebase Cloud Messaging

Pentru transmiterea notificărilor și evenimentelor în timp real către dispozitive mobile s-au luat în calcul mai multe soluții de tip Pub/Sub, însă soluția finală aleasă a fost Firebase, datorită stabilității ridicate prin integrarea strânsă cu sistemul de operare. În spate, se folosesc Google Play Services pentru a se livra informațiile utilizatorului într-o manieră eficientă, sistemul putând să întârzie ușor evenimentele pentru trezi cât mai puțin procesorul dispozitivului notificat [9].

Alte soluții investigate au fost OneSignal și MQTT, cel din urmă fiind folosit de Facebook în aplicația lor de mesagerie [21]. Acest protocol a fost proiectat pentru a trimite date de telemetrie de la sonde din spațiu, menținând consumul de baterie și lățimea de banda la minim. Chiar dacă acest sistem prezintă anumite avantaje, complexitatea ridicată de implementare a fost factorul care a influențat decizia finală de a de a folosi Firebase.

3.2.5 Client

Android este o platformă mobilă care s-a maturizat pe parcursul a 12 versiuni majore și principalul competitor de piață al iOS. Având experiență anterioară ca programator Android și în special cu limbajul de programare Java, a fost o alegere convenabilă pentru un prototip rapid. Este de menționat că aceasta alegere de platformă este pur subiectivă, un client similar putând fi dezvoltat pentru iOS sau cu o tehnologie care suportă cross-compilation cum ar fi React Native sau Ionic.

4 Considerente legate de implementarea soluției tehnice

4.1 Arhitectura sistemului

Sistemul prezentat presupune atât o partare hardware, cât și una software. Hardwareul realizează adaptarea dintre terminalul analog POTS și placa digitală de dezvoltare Raspberry Pi, iar ca software am folosit NodeJS pentru server și Android pentru a implementa un client al serverului.

Cu ajutorul multimetrului am dedus schema electrică a tastaturii și am găsit contactele care sunt conectate în cazul apăsării butonului de pe ultimul rând, coloana din mijloc. Scurtcircuitarea contactorului lamelar care depistează ridicarea receptorului și legarea la difuzorul terminalului au fost realizate mai ușor, circuitul fiind parcurs vizual.



Figura 4.1: Depanare și interfațare terminal POTS

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

4.1.1 Raspberry Pi HAT

După ce etapa de prototipare pe breadboard a fost finalizată, am transcris schema electrică a circuitului cu ajutorul softwareul Fritzing. Proiectarea unui Printed Circuit

Board (PCB) reprezintă penultimul pas înainte de etapa de producție în masa. Printre parametrii importanți în deciderea designului unui circuit printat se numără:

- Tehnologia de montare a componentelor pe **PCB** (Through Hole Technology (**THT**) sau Surface Mounted Device (**SMD**))
- Numărul de straturi de circuit (alegeri comune sunt 2, 4, însă dispozitive complexe precum plăcile video pot folosi până la 12 straturi)
- Grosimea și culoarea plăcii de fibra de sticlă
- Lățimea unui traseu pe placă
- Distanța minima intre trasee
- Diametrul via-urilor (găuri verticale în placă folosite pentru a conecta straturile)
- Materialul folosit pentru lipire și materialul folosit pentru pinii de interfață

Pentru a realiza un **Hardware attached on top (HAT)** compatibil cu Raspberry Pi, am folosit softwareul Fritzing. Acesta permite proiectarea schemei electrice și ulterior trasarea conexiunilor pe layoutul fizic al plăcii. Considerand complexitatea redusă a proiectului, s-a ales folosirea configurației implicate cu două straturi, împreună cu următorii parametri pe care i-am introdus în Fritzing ca și constrângeri de design:

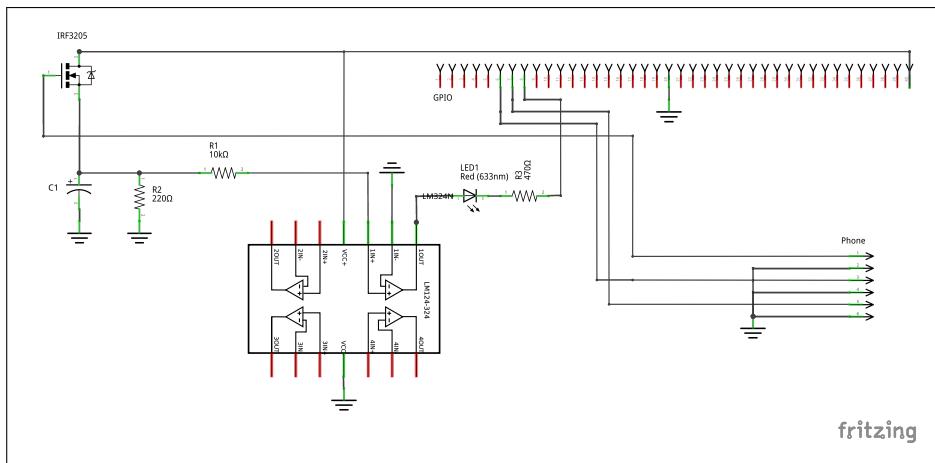


Figura 4.2: Schema electrică HAT Raspberry PI

ACTIONAREA butoanelor terminalului **POTS** se realizează cu ajutorul unor optocuploare, izolând circuitul interfonului care este proiectat pentru a funcționa cu spike-uri de până la 90V de circuitul Raspberry Pi.

Detectarea unui apel este realizată prin legarea unui **Metal Oxide Semiconductor Field Effect Transistor (MOSFET)** la bornele difuzorului terminalului **POTS** și inserierea cu un amplificator operațional în regim de comparator cu referință de 0.1V. Am folosit de asemenea și un Filtru Trece Jos deoarece terminalul este sensibil la zgomote, declanșând accidental notificarea.

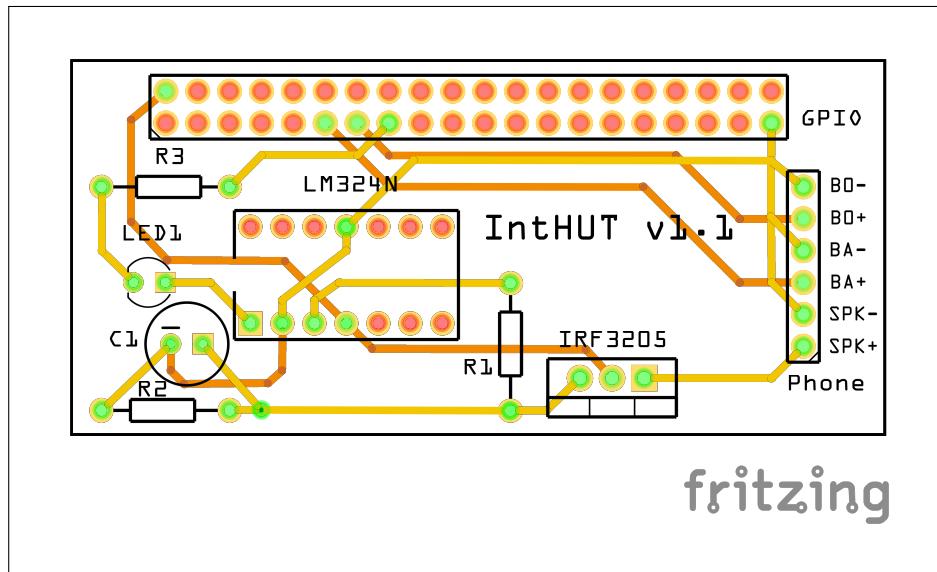


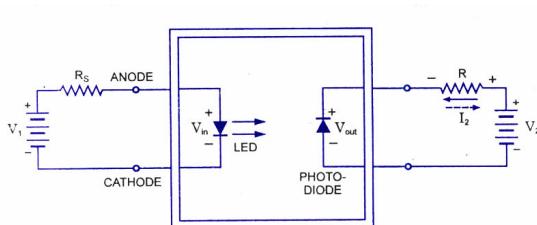
Figura 4.3: Design PCB HAT
(galben - stratul de sus, portocaliu - stratul de jos)

Optocuploare improvizate

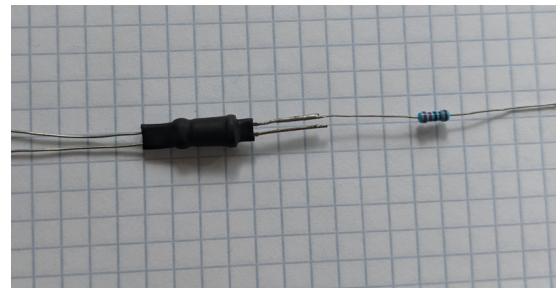
Datorită crizei globale de semiconductori, o placă breakout care include două optocuploare costă aproximativ 50 RON [18]. Considerând simplitatea funcționării acestui circuit, am decis să construiesc propria soluție, folosind componentele de bază: un rezistor pentru limitat curentul, un LED și un fotorezistor. Platforma Raspberry Pi furnizează pinilor saii GPIO 3.3V și un curent maxim de 16mA, iar un LED roșu are o cădere 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 și am încastrat LED-ul împreună cu fotorezistorul într-o incintă fără lumină.



(a) Schema optocuplor [5]



(b) Componenta realizată manual

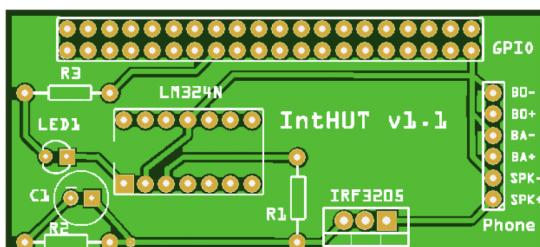
Figura 4.4: Schema electrică optocuplor (a) și rezultat ansablu (b)

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

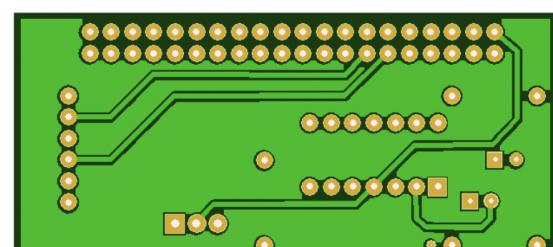
După ce toate traseele au fost puse, ultimul pas este umplerea spațiului rămas pe fiecare strat cu un plan legat la GND pentru a reduce emisiile electromagnetice. Înainte de a trimite fișierul Gerber spre a fi produs, am folosit constrângerile definite inițial pentru a valida proiectul, asigurându-ne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

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

Tabelul 4.1: Parametri aleși pentru fabricarea PCB



(a) Vedere din fata PCB



(b) Vedere din spate PCB

Figura 4.5: Randare plăcută 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 ce folosesc rețea. [12]

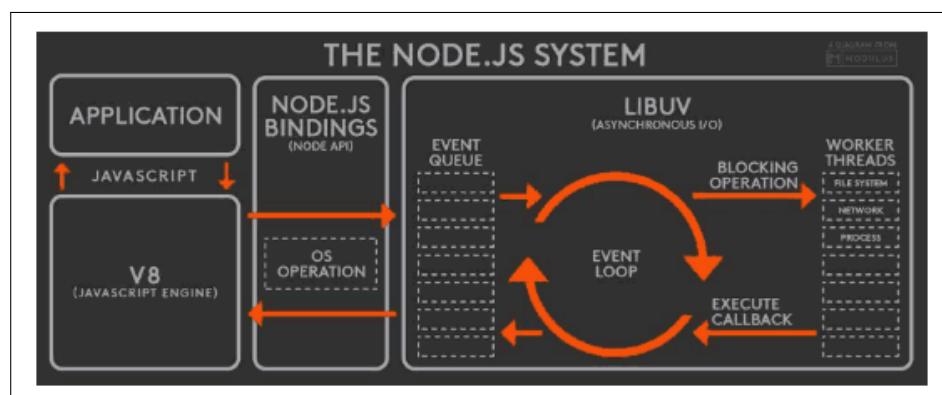


Figura 4.6: Ilustrare event loop și callback queue [14]

Pentru a înțelege cum rulează codul javascript în acest mediu de rulare, trebuie să ne familiarizăm cu modeul de asincroneitate pus la dispoziție, care diferă semnificativ

de cel multi-threaded. Codul JavaScript este parcurs cu ajutorul engine-ului V8 și este tradus în apeluri la librăria nativă 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 cerințelor funcționale, canalul de comunicare dintre client și server trebuie să fie securizat cu un protocol **Hypertext Transfer Protocol Secure (HTTPS)** pentru a permite trimitera credentialelor 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 afișeze un sir de date la o rută statică prestabilită, dovedind astfel autoritatea asupra serverului căruia îi se va emite 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 aplicație NestJs pentru a folosi certificate

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

Autentificare

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

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

3. Semnătură

Semnătura este calculată cu ajutorul unui algoritm simetric de hashing [HMAC SHA-256 \(HS256\)](#) și a unei chei private aplicat pe forma codată în baza 64 a conținutului și metadatelor despre token (expirare, emițător, audiență, subiect). Toate cele trei informații sunt apoi concatenate cu caracterul "." între, constituind forma finală ce va trimisă clientului. Clientul va transmite acest token în comunicațiile ulterioare cu serverul, acesta folosindu-se de mesaj, semnătură și cheia privată pentru a determina dacă a fost sau nu modificat pe parcurs.

Folosirea algoritmului simetric implica faptul că secretul este utilizat atât pentru generarea de tokenuri, cât și pentru validarea lor. În cazul aplicației din lucrare, aceasta nu este o problemă, deoarece ambele metode rulează în interiorul aceluiași proces, fară să expună aplicația la vulnerabilități.

Un avantaj al [JWT](#) este faptul că serverul nu trebuie să întrețină starea unei tabele de sesiuni a utilizatorilor. În esență daca un utilizator este în posesia unui token ne-expirat, acesta este considerat autentificat. Prin urmare, mai multe instanțe ale serverului pot valida în paralel identitatea utilizatorilor, fară nevoie de a interoga o baza de date sau o memorie cache partajată, permitând astfel scalarea pe orizontală a aplicației.

NestJS

Construit peste cel mai popular framework pentru aplicații web disponibil pentru Node.js, NestJS îmbunătățește experiență dezvoltării serviciilor web folosind funcții experimentale din Typescript care permit interpretarea la transpilare a decoratorilor pentru metode și clase. De asemenea urmărește un design [MVC](#) și are pachete întreținute oficial pentru taskuri comune, cum ar fi conectarea la o bază de date sau proiectarea unui mecanism de autentificare și verificare a identității utilizatorilor.

Mutex

Din natura asincronă a limbajului și posibilitatea sistemului de a avea mai mult de un utilizator, trebuie luat în considerare cazul în care mai mulți utilizatori încearcă să interacționeze cu sistemul în același timp. Așadar, trebuie implementat un mecanism similar semaforului binar, numit mutex. Diferență dintre acestea fiind că în cazul mutexului, doar deținătorul original poate să îl elibereze spre a fi folosit din nou.

Acest comportament este dorit pentru a informa clienții serviciului în cazul în care requestul nu poate fi satisfăcut deoarece resursa este ocupată de altcineva.

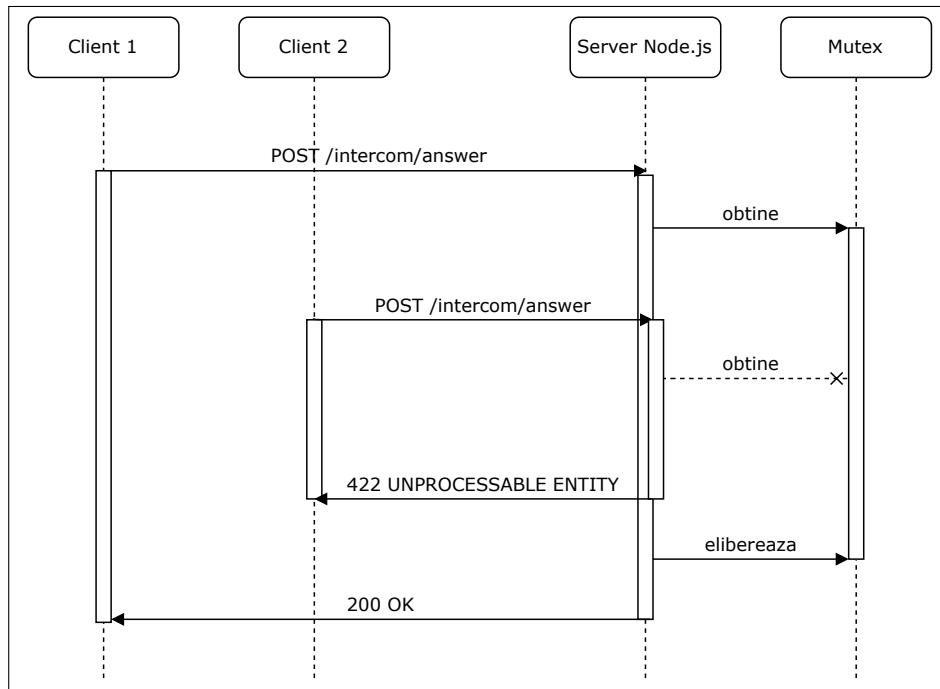


Figura 4.8: Ilustrare resursă disputată între doi utilizatori

Validarea entităților

Folosește funcții din Typescript și NestJS pentru a adaugă și citi metadate prin intermediu decoratorilor de metode. [Data Transfer Object \(DTO\)](#) este reprezentarea unui model din baza de date, dar encodat favorabil pentru interpretarea ușoară a clientilor. Majoritar, acesta va conține mai puține câmpuri decât există în baza de date (repräsentarea unui utilizator nu va avea câmpul pentru parola).

Câmpurile unui [Data Transfer Object \(DTO\)](#) sunt anotate cu tipul așteptat la runtime, și printr-un interceptor de nest care rulează atunci când este invocată metoda unui controller [REST](#), obiectul primit că parametru este verificat contra specificațiilor din metadate. În cazul în care validarea eșuează, clientului îi este întors status 400 (Bad Request) indicând o eroare de formatare a requestului.

Mutând responsabilitatea verificării entităților în cadrul serverului, se mitighează și lipsa unei scheme la nivelul bazei de date, inerente soluțiilor NoSQL. Prin urmare se reduce riscul unor inconsistențe în datele stocate.

Roluri

Urmărind rețeta de dezvoltare observată în validarea entităților, am creat un mecanism de adăugare a metadatelor pe rutele controllerelor în vederea implementării rolurilor utilizatorilor.

Partea de verificare a unui user în timpul unui request, revine așa-numitelor Guard-uri care implementează o interfață comună. În cazul vederilor pentru aplicația de admin dorim să restrictionăm afișarea să utilizatorilor normali, aşadar înlănțuim

Guardurile ce garantează autentificarea unui utilizator și rolul de administrator. În cazul în care un user nu este administrator, acestuia î se întoarce un cod de status 403 (Forbidden) - serverul a autentificat utilizatorul, dar acesta nu are suficiente permisiuni pentru a accesa resursa.

Documentație

Într-un proiect de lungă durată, documentația joacă un rol esențial în ușurința de mențenanță și dezvoltare a codului. Despărțirea logică a componentelor aplicației cât și explicarea eventualelor căi logice și răspunsuri posibile ajută atât clienții externi consumatori ai API-ului cât și dezvoltatorii noi care încearcă să introducă primele modificări.

Astfel, am ales Swagger pentru a parurge proiectul și genera pagini de documentație pentru toate rutele serviciului REST, împreună cu comentarii și potențiale status coduri la care trebuie să se aștepte clienții. De asemenea, Swagger include și un client REST în pagină, împreună cu schema obiectelor și tipurile de date așteptate nu lasă loc de interpretat în comportamentul serverului.

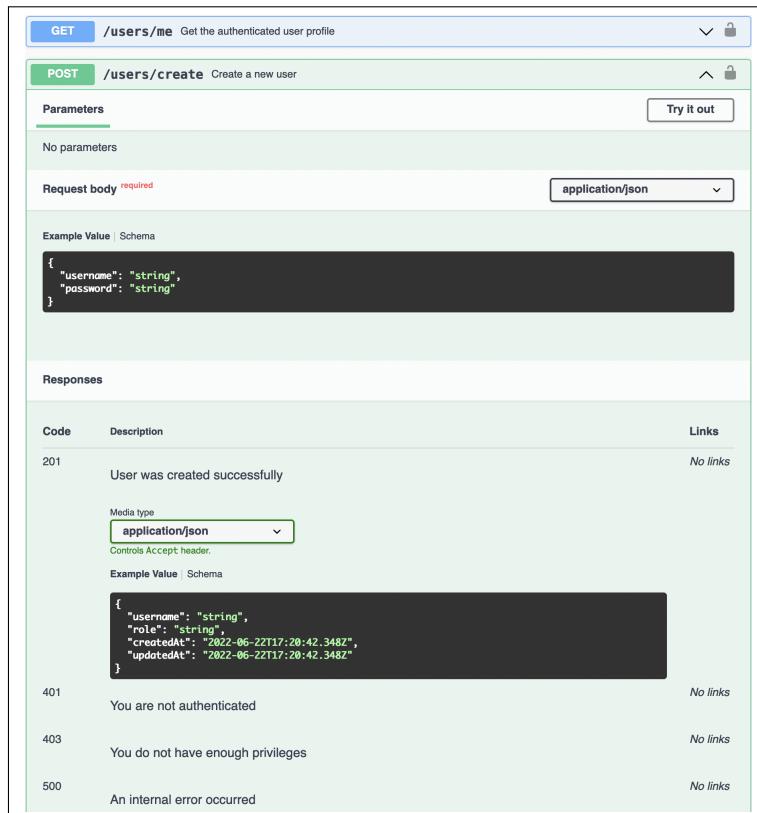


Figura 4.9: Documentație generată pentru ruta /users/create

4.1.3 Android

Android este o platformă mobilă care s-a maturizat pe parcursul a 12 versiuni majore și principalul competitor de piață al iOS. În dezvoltarea acestei aplicații am folosit o abordare similară cu cea a serverului, fiind organizată într-un pattern de design MVC.

O aplicație Android poate fi alcătuită din mai multe componente cu roluri specifice: Activity, Service și BroadcastReceiver. Fiind un sistem de operare care vizează dispozitivele mobile, este optimizat pentru folosirea eficientă a resurselor și a bateriei, astfel că folosirea incorectă a componentelor anterioare poate duce la oprirea accidentală a aplicației sau a unui serviciu în timpul unor operațiuni importante.

O alta particularitate în programarea acestor aplicații este că desenatul interfeței și interceptarea evenimentelor de la utilizator se realizează într-o buclă pe firul de execuție principal. Astfel, orice operațiuni adiționale executate pe acest thread trebuie alese cu atenție pentru a nu duce la aparentă îngreunare prin introducerea latenței la afișaj și la interpretat evenimente de input de la utilizator. Operațiuni care presupun așteptarea după sisteme de IO precum un apel prin rețea, sunt complet blocate din a fi executate pe threadul principal, aruncând o excepție numită "NetworkOnMainThreadException" [8].

Dependency Injection

Pentru a gestionarea ușoară a modulelor aplicației și diferențele surse de date, s-a ales folosirea Dagger2, un framework bine cunoscut de Java. El vine cu extensii pentru Android ce permit controlul granular asupra instantierii modulelor necesare în funcție de ciclul de viață al aplicației sau al unui Activity. Astfel putem defini module globale, precum cel care cheamă API-ul web, instantiat o singură dată pe durata aplicației, sau module locale, precum cel de SharedPreferences care se realoca de fiecare dată când se intră în ecranul principal.

Pe lângă organizarea proiectului în module logice în funcție de funcționalități, Dagger ajută și la managementul memoriei într-un limbaj de programare cu Garbage Collector, precum Java, evitând alocări neneccare sau frecvente.

4.2 Implementarea sistemului

4.2.1 Server

Chiar dacă autentificarea prin interfață grafică de administrator beneficiază de același standard de securitate că și ceilalți clienți, s-a ales crearea unui alt server pe un port diferit de cel al API-ului, permitând flexibilitate maximă utilizatorului final în alegerea expunerii serviciilor.

Personal, am expus serviciul de API pe un port rutat public și interfață pe un port privat, blocat din firewall. În esență, orice utilizator normal poate interacționa cu sistemul atâtă timp cât este autentificat, însă interfața de administrare este disponibilă doar din interiorul rețelei din acasă.

```
/**  
 * @category Strategy  
 */  
  
@Injectable()  
export class JwtStrategy extends PassportStrategy(Strategy) {  
    constructor(  
        private readonly userService: UserService,  
        readonly settingsService: SettingsService,  
    ) {  
        super({  
            jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),  
            ignoreExpiration: false,  
            secretOrKey: settingsService.getJwtSecret(),  
        });  
    }  
}
```

(a) Strategie **JWT** pentru autentificarea clientilor API

```
@Injectable()  
export class JwtCookieStrategy extends PassportStrategy(Strategy, { name: 'jwt-cookie' }) {  
    constructor(  
        private readonly userService: UserService,  
        readonly settingsService: SettingsService,  
    ) {  
        super({  
            jwtFromRequest: (req: Request) => {  
                if (req.cookies == null || req.cookies['api_token'] == null) {  
                    return null;  
                }  
  
                return req.cookies['api_token'];  
            },  
            ignoreExpiration: false,  
            secretOrKey: settingsService.getJwtSecret(),  
        })  
    }  
}
```

(b) Strategie **JWT** pentru autentificarea din browser, prin intermediul cookies

Figura 4.10: Implementare logica autentificare utilizatori

Secvența de generare a comenziilor pentru deschiderea interfonului se află în clasa IntercomService, această secvență include: obținerea mutexului, ridicarea receptorului, așteptarea pentru o secundă, apăsarea butonului pentru răspuns de două ori, cu o pauză de 1.2s între și în final eliberarea mutexului. Secvență pentru respingerea unui apel a fost implementată într-o manieră similară, presupunând doar ridicarea receptorului și închiderea sa după un interval preestabilit 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 4.11: Implementarea metodei pentru permis accesul din IntercomService

Pentru statistici, metoda care persistă un Log în baza de date oferă utilizatorului posibilitatea de a-si declară poziția curentă prin prezentarea latitudinii și longitudinii. În cazul în care clientul alege să nu împărtăsească aceasta informație, serverul vă extrage adresa IP din request sau din headerul X-Forwarded-For dacă este chemat prin intermediul unui proxy și vă folosi serviciul extern ip-api.com care va returna o locație aproximativă.

```


/** 
 * @category Decorator
 */
export type ParamDecoratorType = (
    ...dataOrPipes: Array<unknown>
) => ParameterDecorator;

/** 
 * @category Decorator
 */
export const IpAddress: ParamDecoratorType = createParamDecorator(
    factory: (_: unknown, ctx: ExecutionContext) => {
        const request: Request = ctx.switchToHttp().getRequest();
        return request.header('name: "x-forwarded-for") || request.socket.remoteAddress;
    }
);


```

Figura 4.12: Implementarea metodei pentru permis accesul din IntercomService

4.2.2 Baza de date

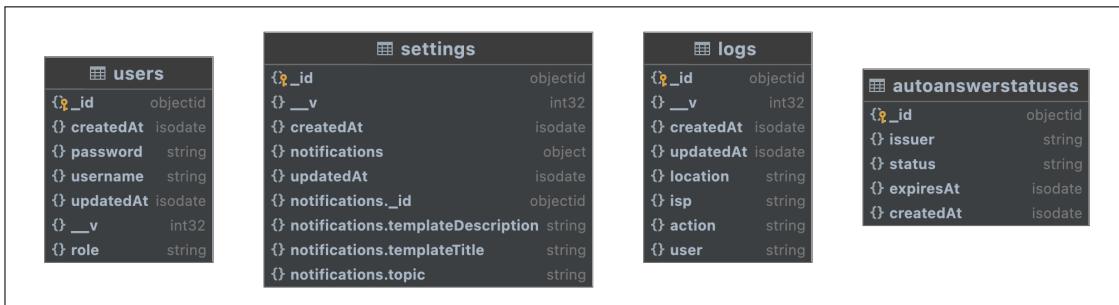


Figura 4.13: Schema bazei de date Mongo

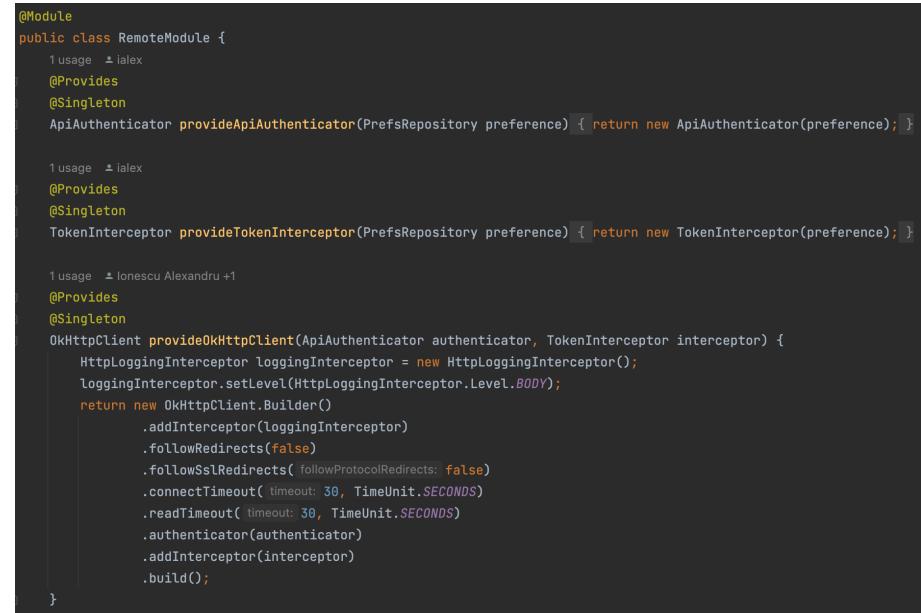
Baza de date nu oferă nicio relație între documente, însă colecția "logs" sau "autoanswerstatuses" trebuie să fie legate din punct de vedere logic cu un utilizator. Putem din nou să beneficiem de adnotări, specificând tipul așteptat pentru validare sau o referință către o alta colecție.

Salvăm astfel id-ul utilizatorului care le-a generat împreună cu ele, iar la momentul interogării bazei de date putem folosi metoda "populate()" din driverul Mongoose care va cauta modelul pentru adnotări de tip referință. În final, driverul va genera query-uri pentru a aduce referințele din colecțiile corespunzătoare. Dacă o referință nu este găsită, se va întoarce null fiind responsabilitatea aplicației să trateze acest caz.

4.2.3 Aplicație mobilă

Pentru a evita complet problema rulării apelurilor de rețea pe threadul greșit, am folosit un client REST numit Retrofit, bazat pe OkHttp. Aceasta se ocupă să coordoneze un ThreadPool (o colecție de threaduri ce execută taskuri de același tip), astfel că

atunci când aplicația cheamă un serviciu, se alege unul din threaduri spre a se executa cererea, răspunsul venind sub forma unui callback pe threadul principal.



```
@Module
public class RemoteModule {
    @Provides
    @Singleton
    ApiAuthenticator provideApiAuthenticator(PrefsRepository preference) { return new ApiAuthenticator(preference); }

    @Provides
    @Singleton
    TokenInterceptor provideTokenInterceptor(PrefsRepository preference) { return new TokenInterceptor(preference); }

    @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 4.14: Configurare modul rețea, împreună cu dependințele sale directe

Datorită adnotărilor `@Singleton` toate obiectele din acest modul vor fi instantiată o singură dată pe durata aplicației, oferind un punct de acces centralizat pentru toate operațiile ce implică chemarea serviciilor **REST**.

De asemenea, am configurat Retrofit să folosească Gson pentru serializarea și deserializarea **DTO**-urilor din format **JSON** în instanțe ale claselor din Java. Pentru această operațiune se folosește de **API**-ul reflectiv oferit de limbajul Java spre a chama constructorul implicit al unei clase și a seta valorile variabilelor sale.

Urmărind un model **MVC** în realizarea aplicației, trebuie să definim metodele implementate de view pentru a fi chemate din interiorul callbackurilor de la nivelul layerului de date. Se decouplează astfel logica de business de prezentarea interfelei, structura ce facilitează testarea și extensibilitatea codului. Cuplarea modulelor trebuie să fie slabă pentru a evita modificarea excesivă a componentelor când se schimbă specificațiile [17].

```
public interface MainView {  
  
    1 usage 1 implementation ✎ Ionescu Alexandru  
    void onLogsResponse(LogsResponse response);  
  
    2 usages 1 implementation ✎ Ionescu Alexandru  
    void onLogsFailure(String error);  
  
    1 usage 1 implementation ✎ Ionescu Alexandru  
    void onAnswerSuccess();  
  
    1 usage 1 implementation ✎ Ionescu Alexandru  
    void onDeclineSuccess();  
  
    1 usage 1 implementation ✎ Ionescu Alexandru  
    void onUserRequired();  
  
    2 usages 1 implementation ✎ Ionescu Alexandru  
    void onNetworkError(Throwable t);  
  
    1 usage 1 implementation ✎ iAlex97  
    void onMoreLogEvent(@NotNull LogEvent event, int pos);
```

Figura 4.15: Definirea evenimentelor viewului principal

4.3 Testarea sistemului

4.3.1 Server

Chiar dacă NestJS oferă suport pentru scrierea de teste unitare prin adăugarea de fisier .spec.ts, cea mai mare provocare a fost mockuirea modulului care comunica cu pinii **GPIO** ai Raspberry Pi. Wrapperul de JavaScript comunică prin intermediul **Node-API (N-API)** cu o librărie statică ce realizează serializarea și deserializarea datelor dintre Node.js/V8 VM și C/C++. Această librărie se linkează la rândul ei cu *bcm2835* pentru a obține acces la pinii **GPIO** și alte funcții din sistemul de **Input/Output (IO)** al cipului Broadcom 2835.

Astfel suntem prezenți cu o problemă, librăria **N-API** poate fi compilată pe alte arhitecturi, dar cu siguranță va genera o excepție la rulare când dependință să, *bcm2835*, va încerca să execute instrucțiuni invalide. Este nevoie de o logică de control care să permită rularea și returnarea unor date prestabilite, când se detectează porneirea pe o arhitectură invalidă, cum ar fi în cazul testelor rulate în mod automat prin intermediul pipelineului de integrare continuă.

Teste unitare

Următorul pas a fost elaborarea testelor unitare. Prin crearea un fișier .spec.ts pentru fiecare controller al API-ului acoperind astfel întreaga funcționalitate a aplicației. Pentru controllerului responsabil deschiderii interfonului serviciile Firebase și baza de date au fost mockuite, testând mecanismul de deschidere/închidere 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 prestabilite și se așteaptă după răspunsurile lor. Astfel parcurgem toată logica serverului, cap-coadă și ne vom asigura de un comportament predictibil.

4.3.2 Aplicația mobilă

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

Tipul de test ales este cel "Robo", care analizează intelligent interfața vizuală a aplicației, după care va genera evenimente de input ca și cum ar fi un utilizator. Planul "Spark" de la Firebase oferă acces la 5 rulări pe dispozitive fizice pe zi și 10 dispozitive virtuale, deci testele au fost executate în limita acestor constrângeri.

Pentru a rula un test, trebuie încărcat un APK sau un AAB, ales dispozitivele pentru executarea testului și eventualii pași adiționali pe care îi dorim acoperiți de Robo. După introducerea ID-ului pentru câmpurile de username și parolă împreună cu valorile asociate unui cont de test, acesta 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 plăcută
- 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 diagonală mare

Analizând tabelul pentru performanță vizuală, observăm că aplicația rulează în margini acceptabile până și pe un dispozitiv mai vechi precum Nexus 5.

De asemenea "Test Lab" poate fi chemat din interiorul pipeline-ului de integrare continuă pentru a rula teste automat după terminarea unui build pe un anumit branch, verificând flow-uri existente din aplicație împotriva potențialelor regresii apărute.

Metrică	Valoare
Druata până la prima afișare	700ms
Eveniment VSync Pierdut	3%
Latenta ridicata input	0%
Thread UI încet	4%
Comandă draw înceată	1%
Încărcare bitmap înceată	0%

Tabelul 4.2: Metrice de performanță raportate pentru Nexus 5

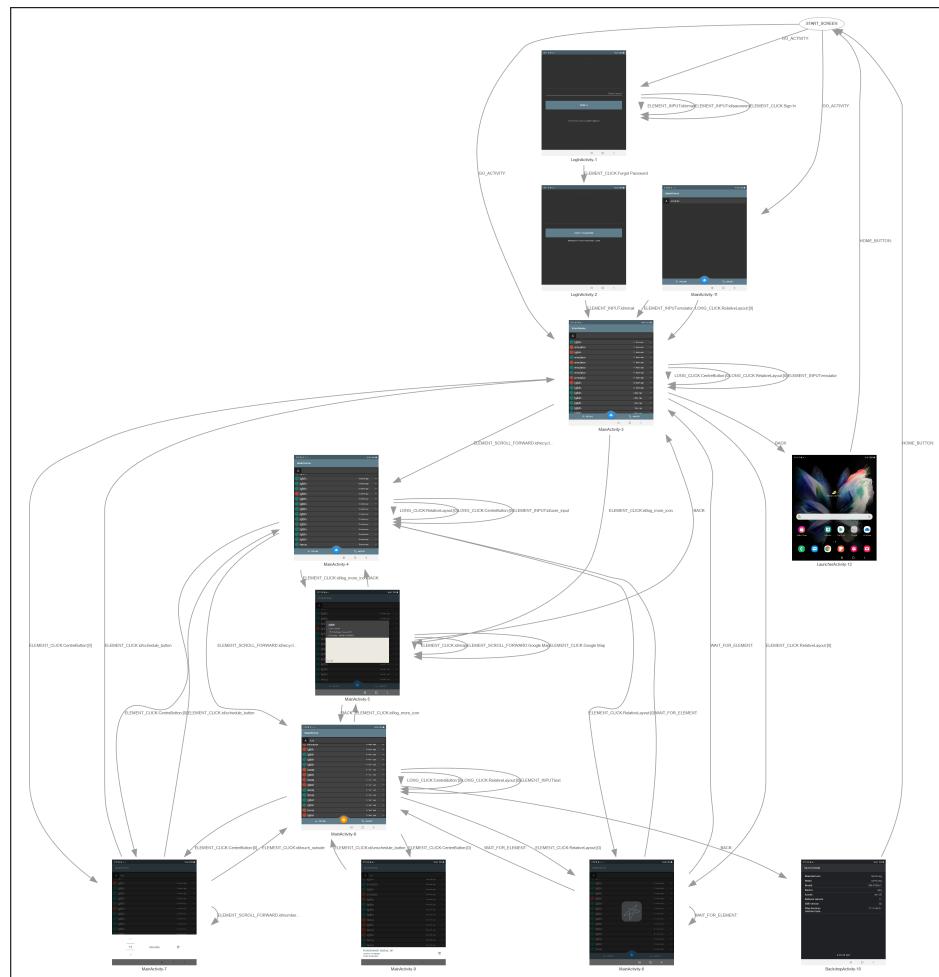


Figura 4.16: Flow testare Robo pe Samsung SM-F926U1

5 Studiu de caz

5.1 Oferirea accesului unui utilizator

Operația de oferire a accesului unui nou utilizator presupune folosirea interfeței de administrare. După setarea numeului de utilizator și a parolei, noul cont este gata pentru folosit, iar pentru convenabilitate codul QR generat poate fi scanat cu aplicația pentru a se autentifica automat.

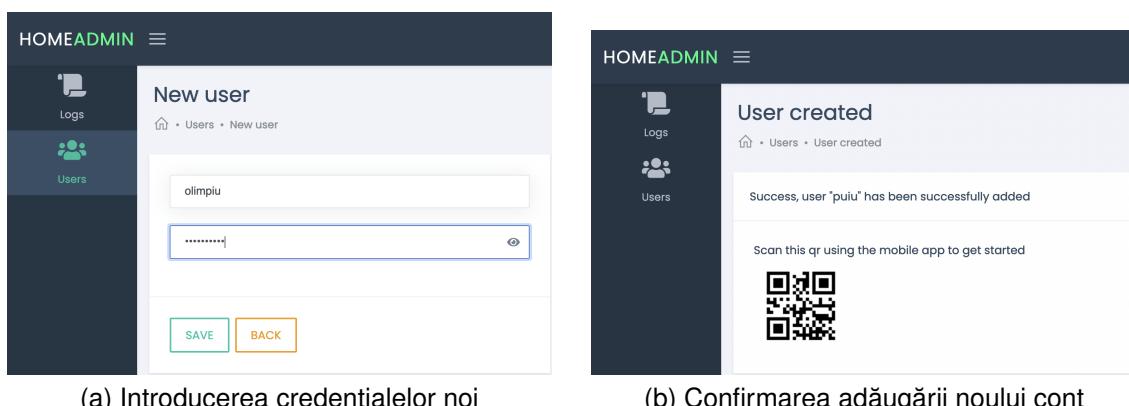


Figura 5.1: Creare utilizator nou

După descărcarea aplicației din Google Play Store, se va putea realiza autentificarea. Un alt avantaj al unui astfel de sistem este eliminarea nevoii cartelelor fizice, ușor pierdut sau încurcat.

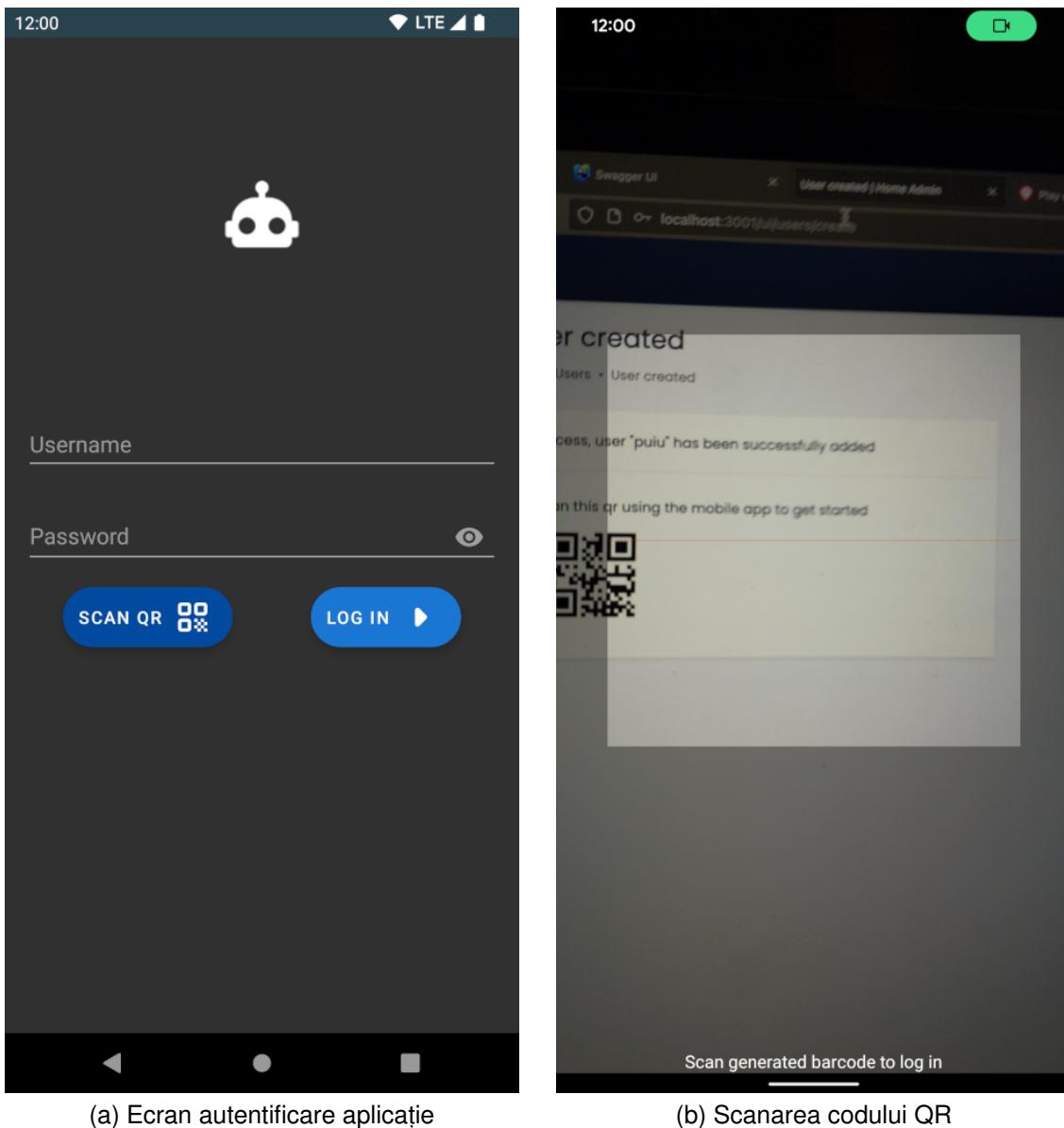
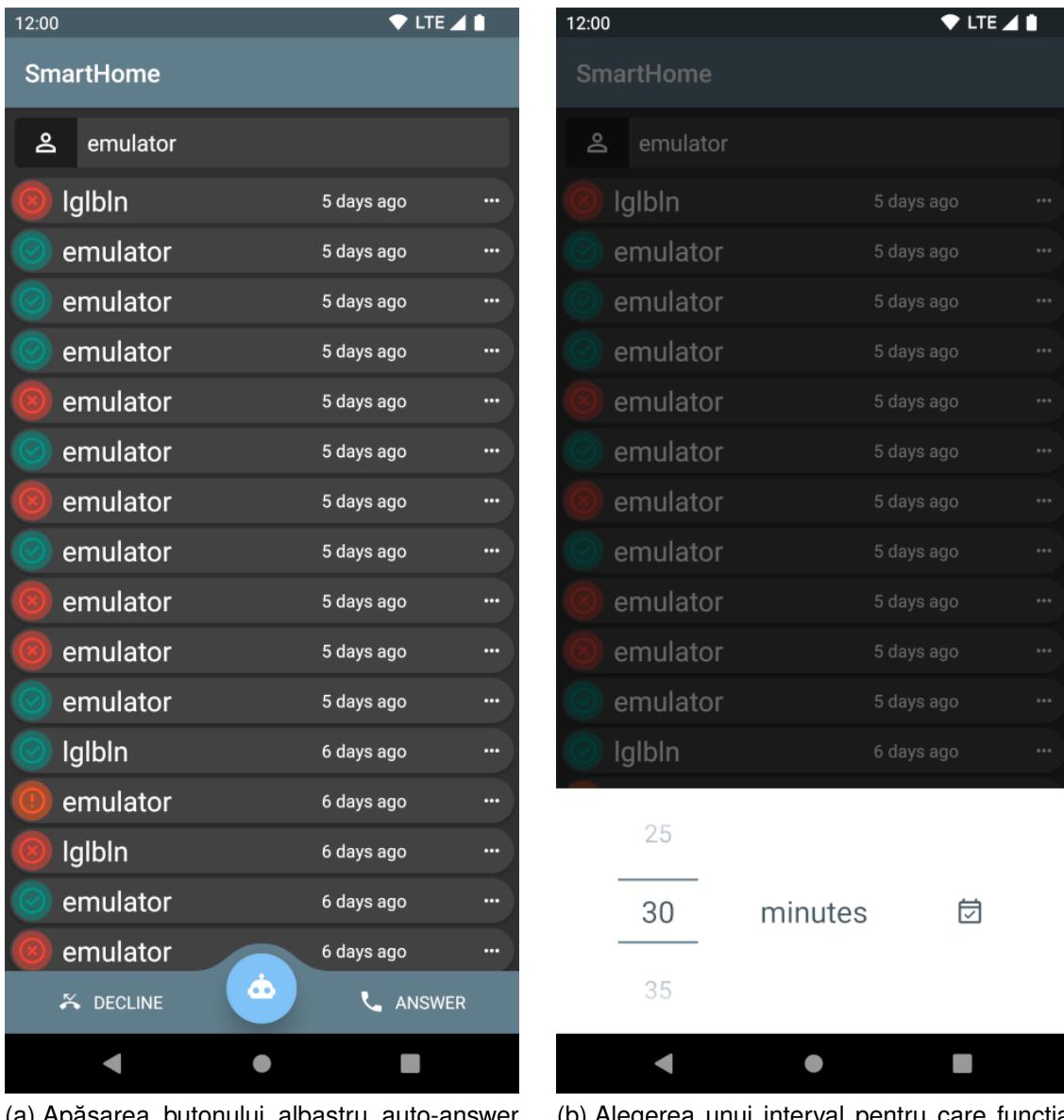


Figura 5.2: Autentificare prin scanare cod QR

5.2 Răspuns automat

Majoritatea vizitelor și apelurilor la interfon sunt predictibile, de exemplu venirea curierului sau a unui poștaș este anunțată printr-un interval aproximativ de timp. Deoarece utilizatorul duce o viață ocupată și știe că urmează să fie sunat, poate programa sistemul să răspundă și să deschidă automat ușa la următorul apel pentru o durată predefinită.



(a) Apăsarea butonului albastru auto-answer va arăta meniul pentru configurarea funcționalității

(b) Alegerea unui interval pentru care funcția va fi activă

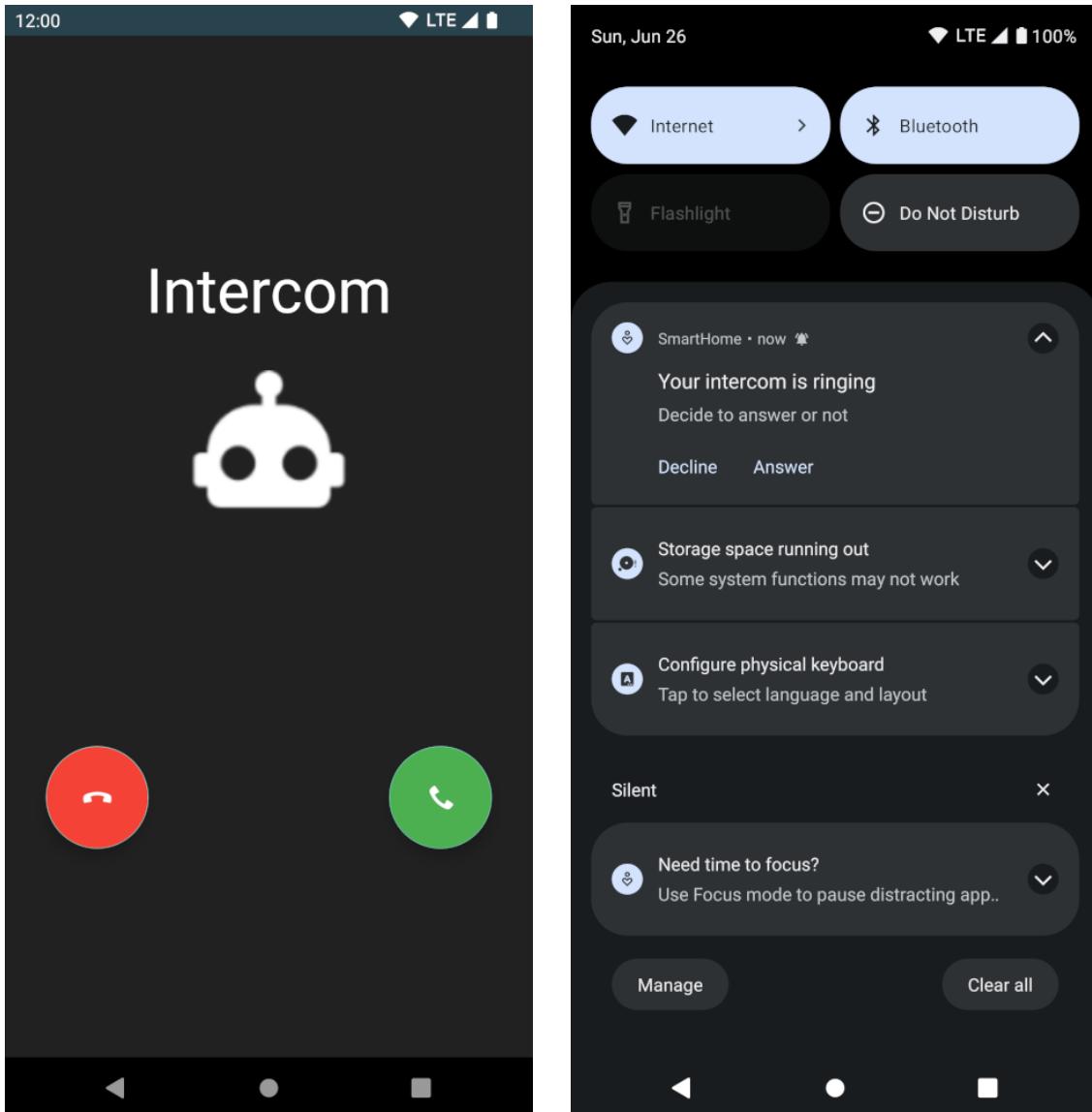
Figura 5.3: Pași pentru setare funcție auto-answer

Se confirmă activarea funcției prin colorarea butonului auto-answer în portocaliu. Într-o manieră similară, dacă se dorește oprirea înainte de termen, se poate face acest lucru din meniul anterior.

5.3 Răspuns de la distanță

Prin conectarea la rețeaaua IoT sistemul este expus Internetului, acesta fiind avantajul soluției, dar și unul din factorii limitanți ai sai. Posibilitatea de deschidere de oriunde clientul are un dispozitiv conectat la Internet înseamnă că în lipsa cartelei standard

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



(a) În cazul în care aplicația este pornită, utilizatorul este redirectat în IncomingActivity

(b) În cazul în care aplicația nu este în foreground, se trimite o notificare de sistem cu două acțiuni

Figura 5.4: Ecran apel

O altă variantă pentru mitigarea acestei probleme a fost folosirea unui tag Near Field Communication (NFC) care conține codat url-ul de acces împreună cu un token care nu expiră. Astfel prin scanarea lui cu un telefon ce dispune de antenă NFC se va realiza request-ul din browserul default al utilizatorului, rezultând în deschiderea interfonului.

6 Concluzii

6.1 Concluzii

Având în vedere informațiile expuse în capituloare 1, 2, 3 și cunoștințele dobândite în realizarea arhitecturii unui sistem de tip încuietoare inteligentă, dar și rezultatele studiului de caz demonstrează că se pot crea soluții folosind tehnologii open-source. Prin alegeri sensibile din punct de vedere tehnic în ceea ce privește securitatea sistemului, utilizatorul normal poate folosi aplicația pentru a își controla interfonul într-un mod sigur. De asemenea, includerea tehnologiilor open-source înseamnă că utilizatorii avansați au multiple alegeri în ceea ce privește implementarea soluției (pot alege unde sunt stocate datele și cine are acces la ele, pot configura din firewall accesul la API și la interfața administrativă).

Prin urmare, această lucrare arată că se poate concepe o soluție IoT la nivelul standardelor secolului 21, atât din punct de vedere al securității cât și al proprietății datelor stocate. Integrarea sa ușoară poate doar să beneficieze adoptării mai rapide a acestui sistem.

6.2 Contribuții

- Sinteză informațiilor în ceea ce privește nișa încuietorilor inteligente.
- Analiza soluțiilor existente din domeniu pentru înțelegerea pieței.
- Conceperea arhitecturii unui sistem similar care să satisfacă cerințele funcționale.
- S-a demonstrat că se pot folosi componente electronice simple pentru a realiza un optocuplător mai ieftin, dar care oferă același funcționalitate în cazul acestei aplicații.
- Implementarea fizica a acestui concept și testarea în viață de zi cu zi.

6.3 Dezvoltări ulterioare

Dezvoltări ulterioare ale sistemului ar presupune construirea unui terminal POTS întreg ca HAT pentru RaspberryPi. Astfel vom putea atinge și cerința de a oferi utilizatorului fluxuri audio pentru a comunica cu persoana de la celălalt capăt al interfonului.