



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

<b>1</b>	<b>Introducere</b>	<b>1</b>
1.1	Obiectivele lucrării de licență . . . . .	1
1.2	Descrierea domeniului din care face parte tema de licență . . . . .	1
1.3	Prezentare pe scurt a capitolelor . . . . .	2
<b>2</b>	<b>Descrierea problemei abordate</b>	<b>3</b>
2.1	Formularea problemei . . . . .	3
2.2	Studiu asupra realizărilor similare din domeniu . . . . .	4
2.3	Stabilirea cerințelor funcționale și nefuncționale ale sistemului . . . . .	7
<b>3</b>	<b>Stadiul actual în domeniu și selectarea soluției tehnice</b>	<b>9</b>
3.1	Stadiul actual al tehnologiilor utilizate pentru dezvoltarea soluției . . . . .	9
3.2	Prezentarea tehnologiilor și platformelor de dezvoltare alese . . . . .	11
<b>4</b>	<b>Considerente legate de implementarea soluției tehnice</b>	<b>13</b>
4.1	Arhitectura sistemului . . . . .	13
4.2	Implementarea sistemului . . . . .	21
4.3	Testarea sistemului . . . . .	23
<b>5</b>	<b>Studiu de caz</b>	<b>26</b>
5.1	Raspuns automat . . . . .	26
5.2	Raspuns de la distanță . . . . .	26
<b>6</b>	<b>Concluzii</b>	<b>27</b>
<b>7</b>	<b>Bibliografie</b>	<b>28</b>

# 1 Introducere

## 1.1 Obiectivele lucrării de licență

### 1.1.1 Realizarea unui studiu de piata pentru determinarea fezabilitatii solutiei

In continuare voi realiza un scurt studiu de piata pe nisa sistemelor [Internet of Things \(IoT\)](#) destinate uzului casnic. Un caz particular de astfel de dispozitive sunt cele care indeplinesc functia de interfon sau ofera contrulul accesului intr-o incinta de la distanta.

In momentul de fata exista pe piata o multitudine de produse de tip incuietoare inteligenta sau sisteme tip interfon GSM, atat de la producatori cunoscuti cat si de la branduri nou infiintate. Aceasta lucrare va analiza trei tipuri de solutii existente, cu implementari diferite, incercand sa identifice functionalitati comune, avantaje si dezvantaje dintr-o plaja cat mai mare de dispozitive.

Solutiile prezentate mai sus au dezavantajul ca nu au fost concepute sa fie integrate cu un sistem existent, intr-un bloc mai vechi. Prin urmare exista un segment de piata de utilizatori care ar dori sa beneficieze de functiile intefonului intelligent, dar nu pot deoarece asta ar presupune schimbarea sistemului din intreaga cladire.

### 1.1.2 Dezvoltarea unui sistem compatibil POTS pentru interfatarea in reteaua IoT

Pentru a putea oferi functiile inteligente unei audiente cat mai large, sistemul propus in aceasta lucrare se poate conecta la reteaua [Plain Old Telephone Service \(POTS\)](#) printr-o simpla mufa RJ11.

## 1.2 Descrierea domeniului din care face parte tema de licență

Aceasta lucrare face parte dintr-un domeniu mai vechi, dar care a prins amploare recent, domeniul automatizarilor casnice si IoT.

### 1.2.1 Iстория

Interesul in conectarea locuintelor pentru a obtine functionalitate aditionala dateaza inca din anii 60, majoritatea fiind concepte prototipate de entuziasti cu inclinatii spre electronica.

Jim Sutherland, inginer la Westinghouse a creat primul sistem de automatizare a domiciliului in anul 1964, ECHO IV. Acesta era capabil sa controleze temperatura, alte aparate casnice cat si sa permita retinerea de mementouri sau liste de cumparaturi. Cu introducerea retelei [Advanced Research Projects Agency Network \(ARPANet\)](#) in 1969, un precursor al Internetului, universul dispozitivelor casnice conectate a cunoscut o perioada rapida de dezvoltare in anii urmatori [14].

Trecerea de la o nouitate scumpa la un sistem ce ofera functii cu adevarat practice a venit sub forma proiectului "X10 Home Automation". Acesta se putea integra cu sistemul de climatizare existent al cladirii, controla electrocasnice mici, cat si coruri de iluminat.

In anul 1984, Asociatia Nationala a Constructorilor din Statele Unite a creat un grup de control numit "Smart House" pentru a accelerata includerea tehnologiei in proiectele viitoare [1].

Pentru consumatori, dezvoltarile din urmatorii ani au adus usi automate pentru garaje, termostate programabile si sisteme de securitate in cadrul monden, concomitant reducand preturile solutiilor oferite. In ciuda acestor semne, sociologii au concluzionat la vremea respectiva ca nu exista un interes real in conceptul "Smart House".

### 1.2.2 Stadiu actual

Solutiile de tip "Smart Home" din prezent se intregreaza in general cu o retea precum Espressif, Apple HomeKit sau Google Home. Aceasta permite controlul dispozitivelor conectate prin intermediul telefonului mobil.

[Apple HomeKit/Google Home](#)

Nest TC

## 1.3 Prezentare pe scurt a capitolelor

//To do

## 2 Descrierea problemei abordate

### 2.1 Formularea problemei

In orase precum Bucuresti, majoritatea blocurilor au fost construite inainte de anul 1990 si prin urmare interfoanele lor se bazeaza pe **POTS**. Traind in era digitala, utilizatorul ideal (e corporatist) isi doreste augmentarea functionalitatilor sistemului existent, pentru a nu trebui sa isi convinga toti vecinii sa investeasca in modernizarea sistemului de acces. Pentru a putea adresa cat mai multi utilizatori, solutia acestei probleme trebuie sa fie agnistica de smartphoneul si interfonul existent a utilizatorului, dar sa ofere integrari cu alte solutii de tip "Smart Home".

In functie de perioada instalarii, sistemele sa impart in doua categorii: analogice si digitale. Posturile de interfon analogice sunt legate la o unitate de comanda care decodeaza semnale **Dual-Tone Multi-Frequency (DTMF)**, genereaza un semnal sinusoidal cu frecventa de 20Hz si amplitudinea de 60-90V pentru sonerie, apoi realizeaza conexiunile necesare dintre postul de afara si cel al apartamentului cautat. Sistemele digitale folosesc o magistrala comună de comunicatii, fiind adresate conform unei scheme prestabilite - fiecare terminal este programat cu o adresa, insa pentru acest procedeu este nevoie de o cheie asociata unitatii de comanda.

Din motive istorice, unitatea centrala **POTS** genereaza semnalul sinusoidal si poarta suficient curent pentru a putea alimenta clopotul aparut in prima generatie de telefoane. Prin urmare, sistemele digitale sunt considerate mai eficiente si mai sigure, dar si mai greu de integrat, datorita implicarii unei persoane autorizate care sa programmeze intregul sistem.

Aceasta lucrare va analiza solutii digitale, insa sistemul final va fi implementat pe un terminal analog. Asadar, trebuie sa intelegem in primul rand mecanismul de adresare si cum este el interpretat de centrala. Dupa cum insinueaza numele, **DTMF** presupune generarea a doua tonuri de frecevente diferite in acelasi timp, conform liniei si coloanei tastei apasate. 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 frecvente **DTMF**

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

unitatea de comanda trebuie sa includa si un circuit logic sesequential pentru a retine starea ultimelor taste apasate. Astfel, ajungem la un numar satisfacator de adrese pentru aplicatia interfonului.

Un exemplu de analiza spectrala a unui astfel de semnal:

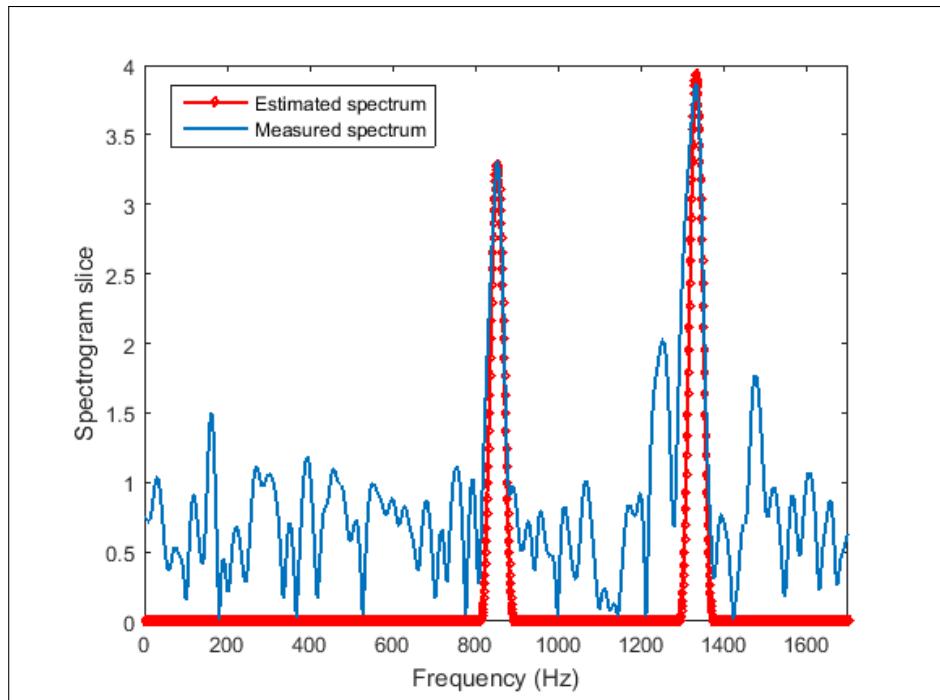


Figura 2.1: Spectrograma tasta "8" [2]

Se pot distinge grafic doua 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 redundanță. În cazul în care primul număr nu se poate apela sau nu răspunde, se va încerca următorul număr programat
- Ofere aplicatie [Android](#) și [iOS](#) pentru programat unitatea



Figura 2.2: Sistem interfon Videx GSM [12]

Dezavantaje:

- Nu ofera integrare cu servicii din reteaua IoT

### 2.2.2 Google Nest x Yale Lock



Figura 2.3: Next x Yale Lock [13]

Avantaje:

- Permite accesul prin intermediul unui PIN ales de utilizator
- Ofera alerte cand cineva inchide sau deschide usa
- Ofera integrare cu Google Home si Nest Home

Dezavantaje:

- Are nevoie de 4 baterii tip AA pentru a functiona
- Nu are acces cu cheie sau cartela
- Nu are versiune rezistenta

### 2.2.3 Level Lock - Touch Edition

Level Lock este o incuietoare inteligenta de tip zavor. Are un design minimalist si ascunde partea electronica in interiorul usii pentru mai multa securitate.



Figura 2.4: Level Lock [7]

Avantaje:

- Multiple modalitati de acces, printre care: amprenta, PIN
- Ofera alerte cand cineva inchide sau deschide usa
- Ofera integrare cu Google Home si Nest Home

### 2.2.4 Comparatii

Produsele de mai sus adreseaza probleme usor diferite, dar incearca sa ofere functionalitati similare. Sistemul oferit de Videx Security prezinta un design rezistent, dar familiar tuturor utilizatorilor si este destinat cladirilor cu mai multi locatari. In contrast, cele doua incuietori inteligente ofera o integrare avansata in reteaua IoT si multiple cai de acces, dar sunt destinate unei singure locuinte.

Incuietoarea de la Yale prezinta cea mai inovativa abordare a acestui design prin decizia deliberata de a nu oferi posibilitatea de acces cu cheie. Astfel, simplifica partea mecanica eliminand singura cale de acces din exterior catre mecanismul incuietorii.

Produsul celor de la Videx Security se bazeaza pe o tehnologie utilizata la scara larga si prin urmare beneficiaza de robustetea unui sistem matur. Spre deosebire de celelalte doua produse analizate, solutia celor de la Videx Security este agnistica de sistemul de operare al telefonului mobil, avand nevoie doar de o conexiune GSM.

Din lipsa unor standarde in domeniu, dispozitivele noi sufera de alte tipuri de probleme si vulnerabilitati, dupa un studiu realizat de cercetatorii de la Bitdefender. Majoritatea sunt in faza initiala de setare, oferind protocoale de securitate invecite sau omitandu-le complet. Este mentionat si un dispozitiv care expune un port Telnet, un protocol invecit si usor de exploatat, fara posibilitatea de a fi dezactivat [6].

## 2.3 Stabilirea cerintelor functionale si nefunctionale ale sistemului

// todo: impartit in functionale/nefunctionale

### 2.3.1 Controlul accesului intr-un apartament

Scopul principal al acestui sistem este de a oferi sau nu acces intr-o incinta, prin urmare consider aceasta cea mai importanta cerinta functionala.

### 2.3.2 Expunerea unui serviciu REST pentru interfatarea cu alte sisteme

Expunerea si abstractizarea terminalului **POTS** este realizata printr-un set de servicii **Representational State Transfer (REST)** care controleaza starea sa. Acest lucru ne permite interfatarea cu aplicatia mobila, interfata de administrare web si alte servicii precum Google Home/Google Assistant/Apple HomeKit.

### 2.3.3 Implementarea unei functii pentru raspuns automat

Aceasta functie va permite utilizatorului sa stabileasca o perioada de timp pentru care sistemul va oferi accesul neconditionat.

### 2.3.4 Dezvoltarea unui client mobil Android

Principalul client care va interactiona cu serviciile **REST** va fi aplicatia mobila ce va avea rolul de a notifica userul cand ii suna interfonul si de a controla starea sistemului.

### 2.3.5 Control granular asupra datelor stocate

Arhitectura aplicatiei necesita interactiunea cu o baza de date, care poate fi tinuta in cloud, pentru convenabilitate sau local. Folosind tehnologii de containerizare precum Docker, putem stoca baza de date local, informatiile fiind stocate intr-un mediu controlat.

### **2.3.6 Criptarea comunicatiilor cu serviciile web**

Avand in vedere nivelul de acces pe care l-ar oferi un exploit al acestei solutii, comunicatiile intre server si clienti trebuie realizate printr-un canal criptat de tip [Secure Sockets Layer \(SSL\)](#). Credentialele userului si ulterior tokenul de acces trebuie trimise doar dupa verificarea autenticitatii serverului si a pachetelor trimise.

### **2.3.7 Oferirea si revocarea accesului la sistem**

Dorim de exemplu sa oferim acces neconditionat unui prieten apropiat pentru a intra in bloc fara a mai suna la interfon. De asemenea ar trebui sa putem realiza si inversul acestei operatii.

### **2.3.8 Expunerea unui flux duplex audio prin tehnologia VoIP**

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

# **3 Stadiul actual in domeniu si 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 [3], 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.

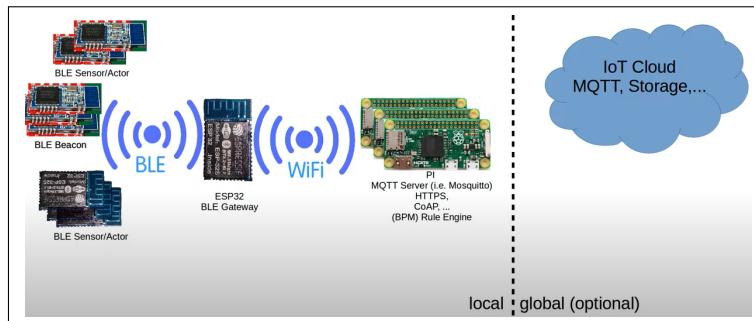


Figura 3.1: Sistem interconectare actori Espressif la internet [8]

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 protocolelor 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. [\[10\]](#)

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 Client**

Android este o platforma mobile care s-a maturizat pe parcusul 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:

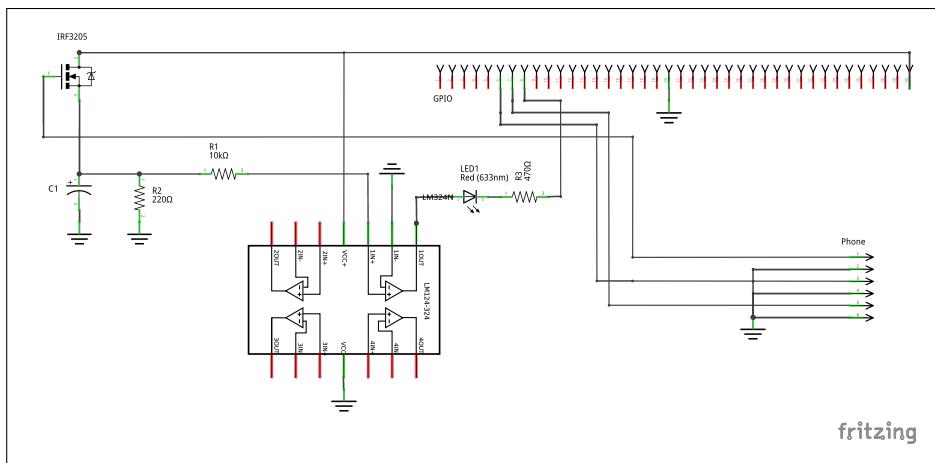


Figura 4.2: Schema electrica HAT Raspberry PI

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.

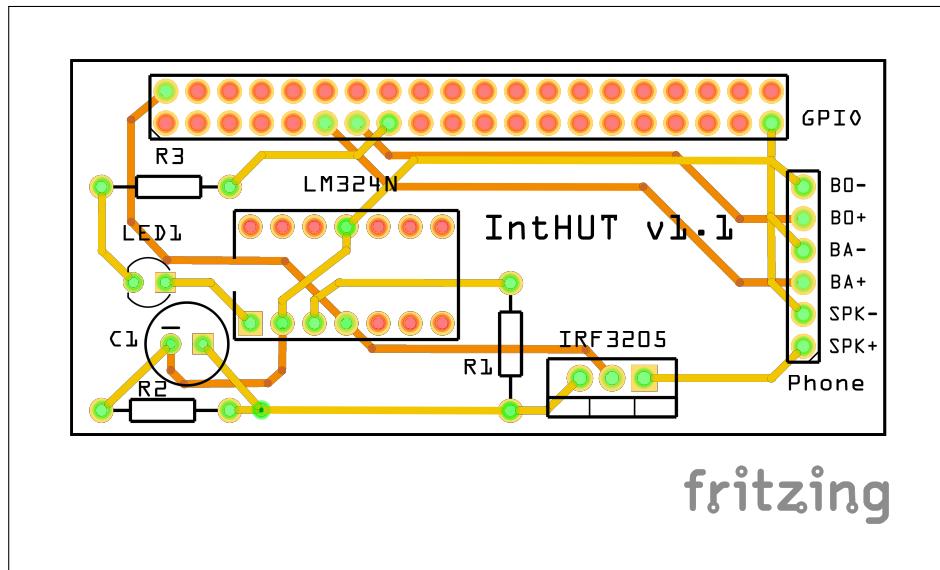


Figura 4.3: Design PCB HAT  
(galben - stratul de sus, portocaliu - stratul de jos)

### 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  $90\Omega$  la anodul ledului si am incastrat LED-ul impreuna cu fotorezistorul intr-o incinta fara lumina.

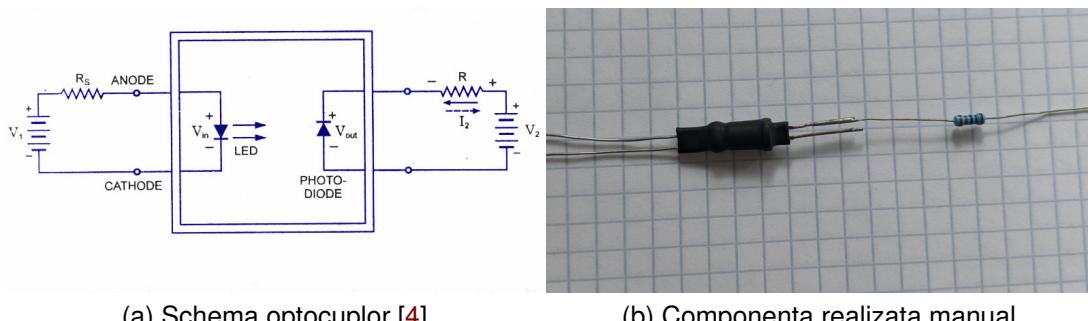


Figura 4.4: Schema electrica optocuplora (a) si rezultat ansablu (b)

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	Latime traseu	Diametru via	Material finisaj
2	THT	1.6mm	1mm	0.5mm	LeadFree HASL-RoHS

Tabelul 4.1: Parametri aleși pentru fabricarea PCB

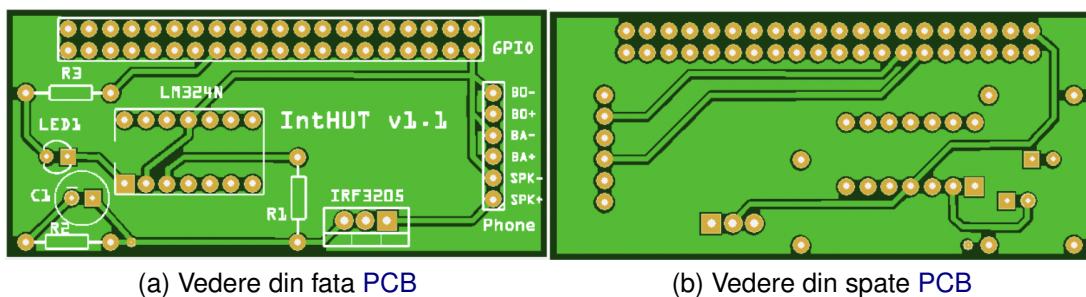


Figura 4.5: Randare placuta înainte de comandat

#### 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. [9]

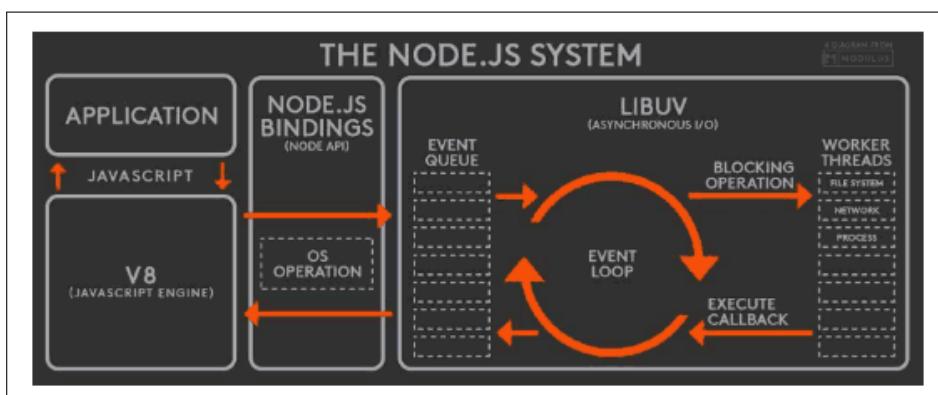


Figura 4.6: Ilustrare event loop și callback queue [11]

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 si este tradus in apeluri la libraria nativa "libuv" care se va ocupa de executarea lor si plasarea in coada de evenimente. Atunci cand "libuv" are nevoie sa faca un apel sincron de sistem, precum interogarea unui server DNS, o face prin intermediul unui thread-pool, atasand un callback pentru a fi chemat atunci cand este gata.

## Criptare HTTPS

Conform cerintelor functionale, canalul de comunicare dintre client si server trebuie sa fie securizat cu un protocol **Hypertext Transfer Protocol Secure (HTTPS)** pentru a permite trimitera credentialelor fara 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 reputabila.

Pentru a verifica identitatea serverului, clientul ii este cerut sa afiseze un sir de date la o ruta statica prestabilita, dovedind astfel autoritatea asupra serverului care ii va fi emis certificatul. Dupa completarea acestor pasi si emiterea certificatului, configurarea serverului sa implementeze protocolul **HTTPS** este relativ simpla:

```
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 mentionat ca incepand cu versiunea de Android 9 (Pie) ca parte dintr-o initiativa de marire a securitatii, sistemul de operare va bloca conexiunile PLAINTEXT daca aplicatiile nu configureaza o exceptie pentru server [5]. De asemenea, prezenta unor certificate self-signed sau insuficiente informatii vor produce rezultate similare, fiind nevoie ca serverul sa prezinte intregul lant de certificate (al serverului, ale autoritatilor intermediare si in final cel al autoritatii centrale).

## Autentificare

Pentru autentificarea si validarea credentialelor utilizatorilor, am ales metoda **JSON Web Token (JWT)**, un standard open source in industrie conform RFC 7519. In cea mai compacta forma a sa, acesta este compus din trei parti, despartite prin caracterul ":".

1. Header (algoritm si 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.

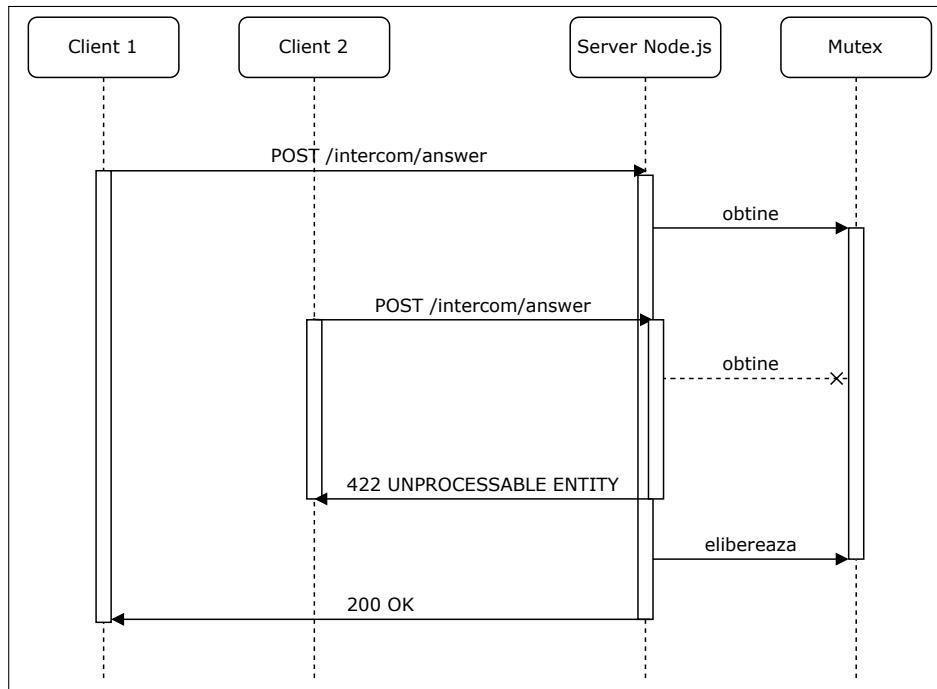


Figura 4.8: Ilustrare resursa disputata intre doi utilizatori

### 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.

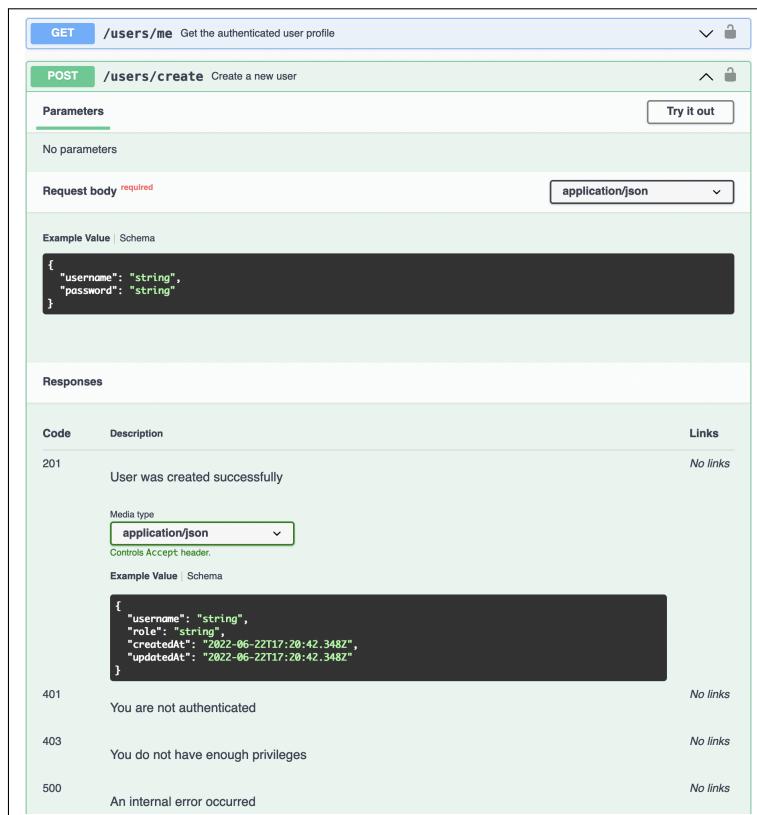


Figura 4.9: Documentatie generata pentru ruta /users/create

### 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. Fiiind 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 interfetei 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".

## 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 interactua 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

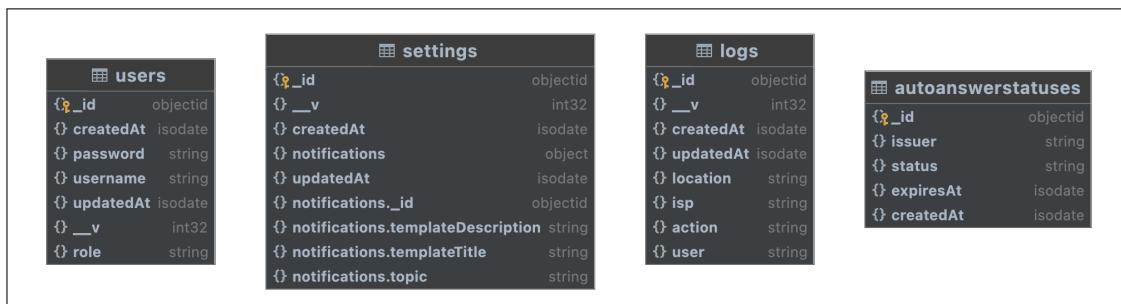


Figura 4.11: Schema bazei de date Mongo

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 dependintele 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 instance 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 librerie statica ce realizeaza serializarea si deserializarea datelor dintre Node.js/V8 VM si C/C++. Aceasta librerie 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 sa creez un fisier `.spec.ts` pentru fiecare controller al API-ului acoperind astfel intreaga functionalitate a aplicatiei. Pentru controllerului responsabil deschiderii interfonului am mockuit serviciile Firebase si baza de date, testand mecanismul de deschidere/inchidere a interfonului in cazul in care aceeasi resursa este accesata de mai multi utilizatori concomitent.

## Teste end-to-end

Pentru a ne asigura ca serviciul **REST** raspunde corect se impune necesitatea testelor cap coada, unde se instantiaza aplicatia intr-o maniera similara productiei, iar cu ajutorul unui client rest se trimit cereri prestabile si se asteapta dupa raspunsurile lor. Astfel parcurgem toata logica serverului, cap-coada si ne vom asigura de un comportament predictibil.

### 4.3.2 Aplicatie mobila

Din cauza fragmentarii foarte mari a versiunilor si configuratiilor posibile pentru toate dispozitivele Android pe care ar putea rula aplicatia, am decis sa folosesc "Firebase Test Lab", un serviciu de la Google care ofera posibilitatea rularii unui test automat sau scriptat pe o gama larga de dispozitive. Asiguram astfel un minim control al calitatii prin descoperirea timpurie a exceptiilor ne tratate.

Tipul de test ales este cel "Robo", care va analizeaza intelligent interfata vizuala a aplicatiei, dupa care va genera evenimente de input ca si cum ar fi un utilizator. Planul "Spark" de la Firebase ne da acces la 5 rulari pe dispozitive fizice pe zi si 10 dispozitive virtuale, deci testelete au fost executate in limita acestor constrainti.

Pentru a rula un testa, trebuie sa incarcam un APK sau un AAB, alegem dispozitivele pe care dorim sa testam si eventualii pasi aditionali pe care ii dorim acoperiti de Robo. Dupa introducerea id-ului pentru campurile de username si parola impreuna cu valorile asociate unui cont de test, a reusit sa se autentifice si sa faca un stress-test al aplicatiei. Urmatoarele dispozitive de test au fost alese:

- SM-F926U1, API Level 30 - telefon pliabil, care prezinta o dimensiune non-standard si provocari unice in ceea ce priveste adaptarea intefetelor pentru o experienta cat mai placuta
- Nexus 5, API Level 23 - versiune mai veche, dar foarte populara de Android, rezolutie standard FullHD, acest test se asigura ca aplicatia este compatibila cu versiuni anterioare ale sistemului de operare
- SM-T720, API Level 28 - tableta, ne asiguram ca aplicatia este folosibila pe un ecran cu diagonala mare

Metrică	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 existente din aplicatie impotriva potenialelor regresii aparute.

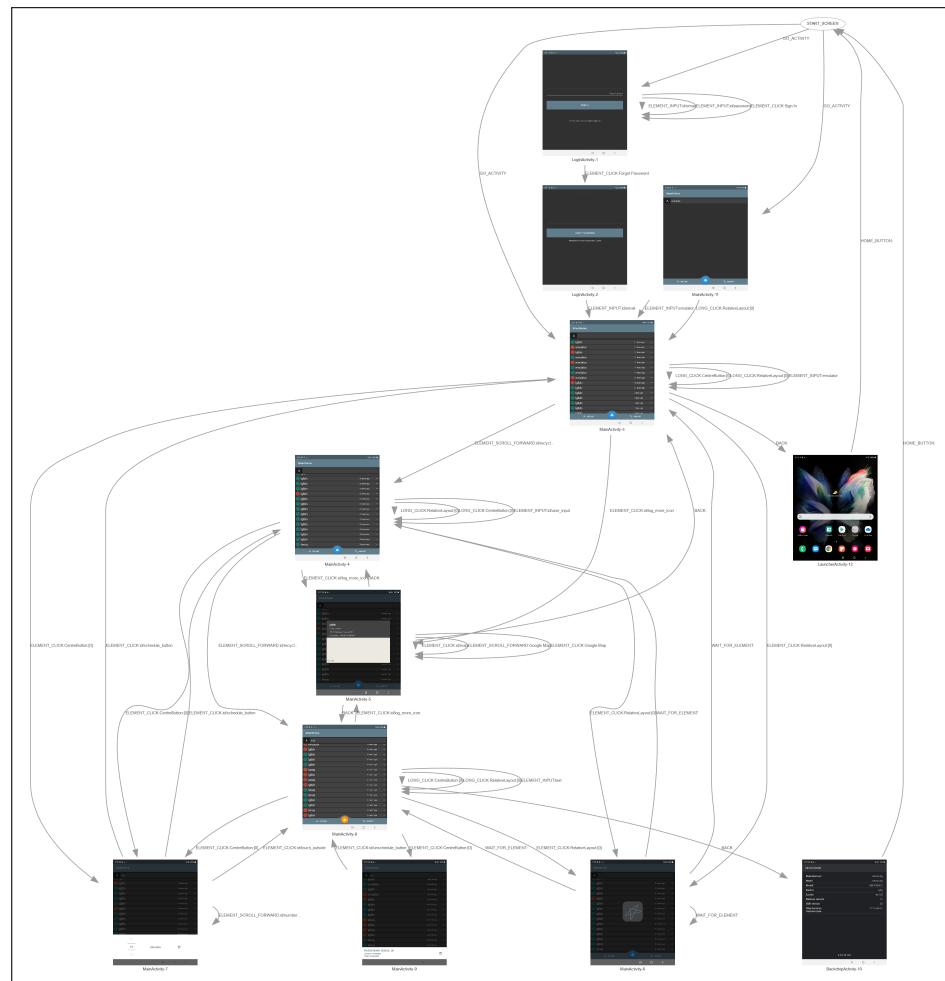


Figura 4.13: Flow testare Robo pe Samsung SM-F926U1

# **5 Studiu de caz**

## **5.1 Raspuns automat**

Mi-am comandat pizza si ajunge in timp ce folosesc pistolul de lipit. Prin urmare voi programa interfonul sa raspunda si sa deschida automat usa la urmatorul apel.

## **5.2 Raspuns de la distanta**

Mi-am pierdut cartela standard [Radio-Frequency Identification \(RFID\)](#) pentru a intra in bloc. Asadar, voi suna la numerul apartamentului meu si voi folosi aplicatia mobila pentru a raspunde la propriul apel.

# **6 Concluzii**

concluzia domnuleee?

# 7 Bibliografie

- [1] Frances K Aldrich. "Smart homes: past, present and future". În: *Inside the smart home*. Springer, 2003, pag. 18–18.
- [2] Nattapol Aunsri. "A bayesian filtering approach with time-frequency representation for corrupted dual tone multi frequency identification". În: 24 (ian. 2016), pag. 370–377.
- [3] 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).
- [4] CircuitsToday. *Optocoupler*. 2009. URL: <https://www.circuitstoday.com/wp-content/uploads/2009/08/optocoupler.jpg> (vizitat 17/06/2022).
- [5] 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).
- [6] Alexandra Gheorghe. *The Internet of Things: Risks in the connected home*. Research Paper BD-NGZ-86847. Bitdefender, 2016.
- [7] Level Home. *Level Lock: The Smallest and Most Advanced Smart Lock Ever*. 2019. URL: <https://level.co/products/lock> (vizitat 29/04/2022).
- [8] 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).
- [9] Node.js. *About Node.js*. 2021. URL: <https://nodejs.org/en/about/> (vizitat 22/06/2022).
- [10] 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).
- [11] 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).
- [12] Videx Security. *GSM Intercoms*. 2016. URL: <https://www.videxuk.com/system/gsm-intercoms> (vizitat 28/04/2022).
- [13] Google Store. *Nest x Yale Lock*. 2018. URL: [https://store.google.com/us/product/nest\\_x\\_yale\\_lock?hl=en-US](https://store.google.com/us/product/nest_x_yale_lock?hl=en-US) (vizitat 28/04/2022).
- [14] 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).