

PROIECT DE DIPLOMA  
Sistem IoT pentru controlul  
accesului in clădire

Alexandru Cristian IONESCU

2021  
Noiembrie

# Cuprins

<b>1</b>	<b>Introducere</b>	<b>3</b>
1.1	Obiectivele lucrării de licență	3
1.1.1	Realizarea unui studiu de piață pentru determinarea fezabilității soluției	3
1.1.2	Dezvoltarea unui sistem compatibil POTS pentru interfatarea în rețeaua IoT	3
1.2	Descrierea domeniului din care face parte tema de licență	3
1.2.1	Istoric	4
1.2.2	Curent	4
1.3	Prezentare pe scurt a capitolelor	4
<b>2</b>	<b>Descrierea problemei abordate</b>	<b>5</b>
2.1	Formularea problemei	5
2.2	Studiu asupra realizărilor similare din domeniu	5
2.2.1	Videx UK	5
2.2.2	Google Nest x Yale Lock	6
2.2.3	Level Lock - Touch Edition	7
2.2.4	Comparatii	7
2.3	Stabilirea cerințelor funcționale și nefuncționale ale sistemului	8
2.3.1	Controlul accesului într-un apartament	8
2.3.2	Expunerea unui serviciu REST pentru interfatarea cu alte sisteme	8
2.3.3	Implementarea unei funcții pentru răspuns automat	8
2.3.4	Dezvoltarea unui client mobil Android	8
2.3.5	Control granular asupra datelor stocate	8
2.3.6	Criptarea comunicațiilor cu serviciile web	8
2.3.7	Oferirea și revocarea accesului la sistem	8
2.3.8	Expunerea unui flux duplex audio prin tehnologia VoIP	9
<b>3</b>	<b>Stadiul actual în domeniu și selectarea soluției tehnice</b>	<b>10</b>
3.1	Stadiul actual al tehnologiilor utilizate pentru dezvoltarea soluției	10
3.1.1	Apple, Amazon, Google	10
3.1.2	Espressif	11
3.1.3	Soluția hobbyista	11
3.2	Prezentarea tehnologiilor și platformelor de dezvoltare alese	11
3.2.1	Hardware	11
3.2.2	Backend	12
3.2.3	Baza de date	12
3.2.4	Client	13

<b>4</b>	<b>Considerente legate de implementarea soluției tehnice</b>	<b>14</b>
4.1	Arhitectura sistemului . . . . .	14
4.1.1	Raspberry Pi HUT . . . . .	14
4.1.2	Webserver NodeJS . . . . .	15
4.1.3	Android . . . . .	18
4.2	Implementarea sistemului . . . . .	18
4.3	Testarea sistemului . . . . .	18
<b>5</b>	<b>Studiu de caz</b>	<b>19</b>
5.1	Raspuns automat . . . . .	19
5.2	Raspuns de la distanta . . . . .	19
<b>6</b>	<b>Concluzii</b>	<b>20</b>
<b>7</b>	<b>Bibliografie</b>	<b>23</b>

# Capitolul 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 dezavantaje 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 inteligent, dar nu pot deoarece asta ar presupune schimbarea sistemului din intreaga cladire.

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

Pentru a putea oferi functiile inteligente unei audiente cat mai large, sistemul propus in aceasta lucrare se poate conecta la retea 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 Istoric

Interesul in conectarea locuintelor pentru a obtine functionalitate aditionala dateaza inca din anii 60, majoritatea fiind concepute 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 urmasori [10].

Trecerea de la o noutate 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 corpuri de iluminat.

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

Pentru consumatori, dezvoltarile din urmasorii ani au adus usi automate pentru garaje, termostate programabile si sisteme de securitate in cadrul monden, concomitent 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 Curent

Solutiile de tip "Smart Home" din prezent se integreaza in general cu o retea precum Espresso, 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

# Capitolul 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 (e corporatist) își dorește augmentarea funcționalităților 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".

### 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 rețeauă. Ele necesită doar o sursă de curent externă, o antenă și o cartelă Subscriber Identity Module (SIM) pentru a opera.



Figura 2.1: Sistem interfon Videx GSM [8]

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
- Oferă aplicație Android și iOS pentru programat unitatea

Dezavantaje:

- Nu oferă integrare cu servicii din rețeaua IoT

### 2.2.2 Google Nest x Yale Lock

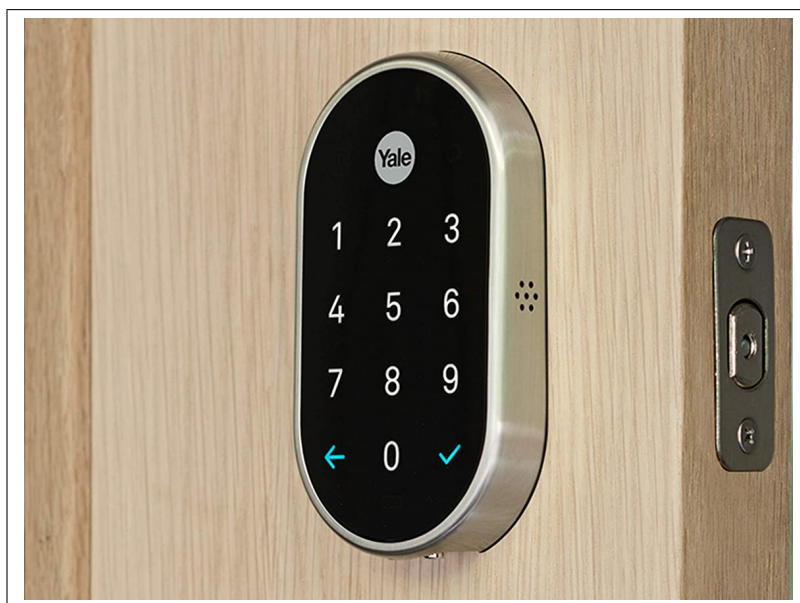


Figura 2.2: Next x Yale Lock [9]

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 cartela
- Nu are versiune rezistentă



Figura 2.3: Level Lock [5]

### 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.

Avantaje:

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

### 2.2.4 Comparatii

Produsele de mai sus adreseaza probleme usor diferite, dar incerca 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 rețeaua 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 agnostica 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:

[4]



## **2.3 Stabilirea cerințelor funcționale si nefuncționale ale sistemului**

### **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)

## Capitolul 3

# Stadiul actual in domeniu si selectarea solutiei tehnice

### 3.1 Stadiul actual al tehnologiilor utilizate pentru dezvoltarea solutiei

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, agregate, 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 [2], 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 layere din stiva Open Systems Interconnection (OSI) intr-unul singur, reducand latenta cauzata de pierderea pachetelor in retele congestionate. Fiind mai simplu, iroieste 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.

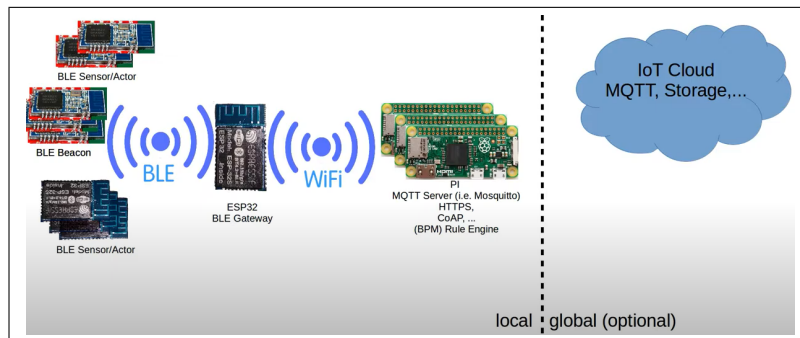


Figura 3.1: Sistem interconectare actori Espressif la internet [6]

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 microcontrollere 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 protoalelor 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. Acesta 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 interactiunea 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 interactiunea 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 librarie 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. [7]

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 compunere.

### 3.2.3 Baza de date

Din punct de vedere al scalabilitatii, paradigma 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 latentă mai ridicată în comunicarea cu baza de date și retransmiterea comenzilor în cazul în 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.

# Capitolul 4

## Considerente legate de implementarea soluției tehnice

### 4.1 Arhitectura sistemului

Sistemul prezentat presupune atât o parte 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.

#### 4.1.1 Raspberry Pi HUT

Pentru a proiecta un Printed Circuit Board (PCB) am folosit softwareul Fritzing. Acesta permite proiectarea schemei electrice și ulterior trasarea conexiunilor pe layoutul fizic al plăcii.

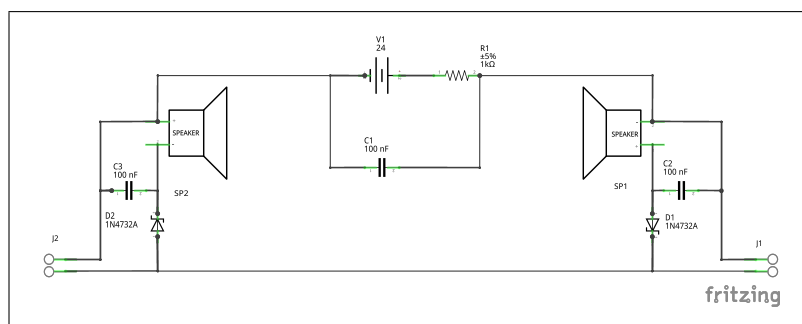


Figura 4.1: Ilustrare resursa disputata intre doi utilizatori

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

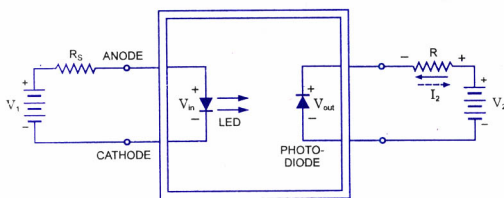
Detectarea unui apel este realizată prin legarea unui Metal Oxide Semiconductor Field Effect Transistor (MOSFET) la bornele difuzorului terminalului POTS și inserierea cu un amplificator operational în regim de comparator cu referința de 0.1V. Am folosit de asemenea și un Filtru Trece Jos deoarece terminalul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

## Optocuploare homemade

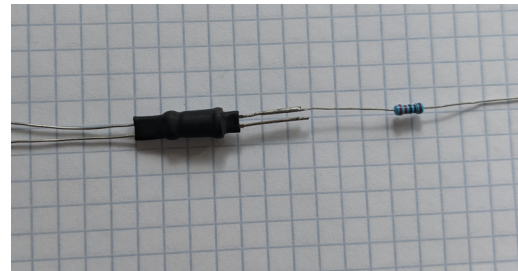
Datorita crizei globale de semiconductori, o placuta breakout care include doua optocuploare costa aproximativ 50 RON. Considerand simplitatea functionarii acestui circuit, am decis sa construiesc propria solutie, folosind componentele de baza: un rezistor pentru limitat curentul, un LED si un fotorezistor. 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:

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

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



(a) Schema optocuplor [3]



(b) Ansanblu realizat manual

Figura 4.2: Schema electrica optocuplor (a) si rezultat ansablu (b)

Deoarece aveam nevoie sa scurtcircuitez un contactor pe partea terminalului POTS am omis rezistenta legata fotorezistorului. Atunci cand ledul este aprins, rezistenta fotorezistorului scade, comportandu-se aproape ca un conductor ideal, atingand lamelele contactorului corespunzator receptorului.

### 4.1.2 Webserver NodeJS

NodeJS este un

#### 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 folosi acest token in comunicatiile ulterioare cu serverul, acesta folosindu-se de mesaj, semnatura acestuia 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 aceluasi 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 framework pentru aplicatii web disponibil pentru Node.js, NestJS imbunatateste experienta dezvoltarii serviciilor 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 intretinute 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.

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

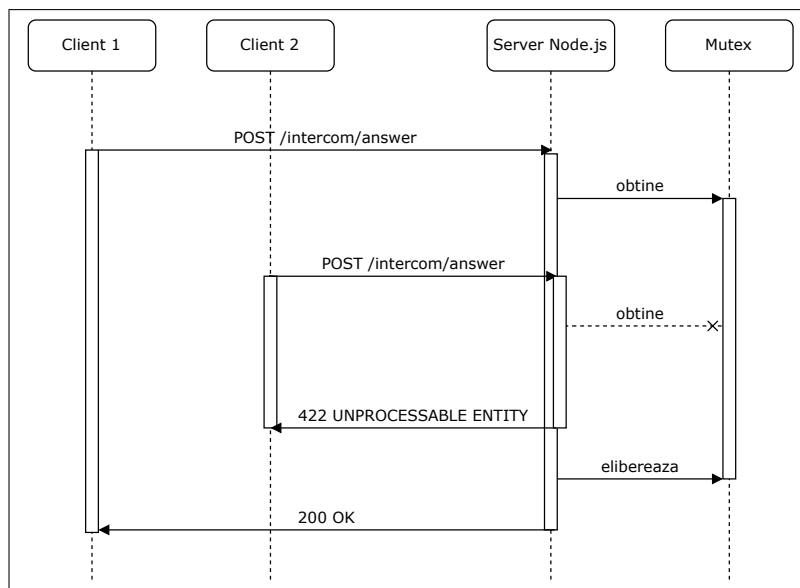


Figura 4.3: Ilustrare resursa disputata intre doi utilizatori

## 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 (reprezentarea unui utilizator nu va avea campul pentru parola).

Campurile unui Data Transfer Object (DTO) sunt adnotate 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

Urmarind "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 comuna. In cazul vederilor pentru aplicatia de admin dormin sa restrictionam afisarea sa userilor normali, asadar inlantuim "Guard-urile" 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

### **4.1.3 Android**

Android este o platforma mobile 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. De asemenea am folosit un framework pentru injectarea automata a dependintelor.

#### **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 nenecesare sau frecvente.

## **4.2 Implementarea sistemului**

## **4.3 Testarea sistemului**

# Capitolul 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 numarul apartamentului meu si voi folosi aplicatia mobila pentru a raspunde la propriul apel.

# Capitolul 6

## Concluzii

concluzia domnuleee?

# Acronime

**ADC** Analog to Digital Convertor. 9

**ARPANet** Advanced Research Projects Agency Network. 4

**AWS** Amazon Web Services. 10

**CAD** Computer Assisted Design. 12

**DAC** Digital to Analog Convertor. 9

**DTO** Data Transfer Object. 17

**FP** Functional Programming. 12

**FRP** Functional Reactive Programming. 12

**GPIO** General-Purpose Input/Output. 12, 15

**GSM** Global System for Mobile Communications. 5, 7

**HS256** HMAC SHA-256. 16

**IoT** Internet of Things. 3, 6, 7, 10, 11

**IP** Internet Protocol. 10, 11

**JWT** JSON Web Token. 15, 16

**MOSFET** Metal Oxide Semiconductor Field Effect Transistor. 14

**MVC** Model View Controller. 12, 16, 18

**OOP** Object Oriented Programming. 12

**OSI** Open Systems Interconnection. 11

**PCB** Printed Circuit Board. 12, 14

**POTS** Plain Old Telephone Service. 3, 5, 8, 12, 14, 15

**REST** Representational State Transfer. 8, 11, 17

**RFID** Radio-Frequency Identification. 5, 19

**SIM** Subscriber Identity Module. 5

**SSL** Secure Sockets Layer. 8

**VoIP** Voice Over IP. 9

# Capitolul 7

## Bibliografie

- [1] Frances K Aldrich. "Smart homes: past, present and future". in *Inside the smart home*: Springer, 2003, **pages** 18–18.
- [2] Daniel Bryants. *Apple Rebuilds Siri Backend Services Using Apache Mesos*. 2015. URL: <https://www.infoq.com/news/2015/05/mesos-powers-apple-siri/> (**urlseen** 13/06/2022).
- [3] CircuitsToday. *Optocoupler*. 2009. URL: <https://www.circuitstoday.com/wp-content/uploads/2009/08/optocoupler.jpg> (**urlseen** 17/06/2022).
- [4] Alexandra Gheorghe. *The Internet of Things: Risks in the connected home*. Research Paper BD-NGZ-86847. Bitdefender, 2016.
- [5] Level Home. *Level Lock: The Smallest and Most Advanced Smart Lock Ever*. 2019. URL: <https://level.co/products/lock> (**urlseen** 29/04/2022).
- [6] Nina. *Why use mqtt server for BLE gateway?* 2021. URL: <https://stackoverflow.com/questions/68195682/why-use-mqtt-server-for-ble-gateway> (**urlseen** 05/06/2022).
- [7] Stack Overflow. *Stack Overflow Developer Survey 2021*. 2021. URL: <https://insights.stackoverflow.com/survey/2021#section-most-popular-technologies-programming-scripting-and-markup-languages> (**urlseen** 05/06/2022).
- [8] Videx Security. *GSM Intercoms*. 2016. URL: <https://www.videxuk.com/system/gsm-intercoms> (**urlseen** 28/04/2022).
- [9] 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) (**urlseen** 28/04/2022).
- [10] Zeus Integrated Systems. *A Brief History of Smart Home Automation*. 2019. URL: <https://zeusintegrated.com/blog/item/a-brief-history-of-smart-home-automation> (**urlseen** 02/06/2022).