

# Lista figurilor

Figura 1: Schemă design.....	9
Figura 2: Arhitectura detaliată a sistemului .....	11
Figura 3: Interacțiunea utilizatorului cu sistemul.....	12
Figura 4: Diagrama de activitate a interacțiunii utilizatorului cu sistemul .....	13
Figura 5: Structura unei componente angular .....	16
Figura 6: Structura aplicației web .....	17
Figura 7: Reprezentarea lampei .....	19
Figura 8: Structura serverului Java .....	21
Figura 9: Captură a comunicației dintre server și un dispozitiv .....	29
Figura 10: Documentația interactivă a server-ului.....	30
Figura 11: Documentarea unei metode .....	30
Figura 12: Observarea traficului HTTP dintre aplicația web și server .....	31
Figura 13: KY-008 Modulul senzor laser.....	34
Figura 14: KY-016 Modul LED cu 3 culori .....	35
Figura 15: KY-019 Modul releu 5V.....	36
Figura 16: KY-011 Modul LED cu 2 culori .....	36
Figura 17: Deschiderea aplicației web .....	37
Figura 18: Rularea serverului java .....	39

# Lista tabelelor

Tabelul 1: Codul de rutare.....	18
Tabelul 2: Dependența la baza de date, definită pom.xml .....	21
Tabelul 3: Procedura responsabilă de conectarea la un dispozitiv.....	23
Tabelul 4: Comenzile protocolului de comunicare.....	25
Tabelul 5: Definirea protocolului de comunicare.....	26
Tabelul 6: Comanda pentru generarea certificatului SSL/TLS.....	28
Tabelul 7: Makefile-ul dispozitivului lampă .....	33
Tabelul 8: Scriptul ce pornește toate dispozitivele simultan .....	33
Tabelul 9: Definirea portului în fișierul .angular-cli.json .....	38

# Cuprins

Lista figurilor.....	i
Lista tabelelor.....	i
Cuprins	ii
Introducere.....	1
Context .....	1
Motivație .....	1
Ce este „Smart Home”? .....	2
Facilitățile traiului într-o casă inteligentă.....	3
Provocări .....	5
1. Contribuții .....	7
2. Arhitectura sistemului.....	8
2.1. Proiectare .....	8
2.2. Detalii arhitecturale.....	10
2.3. Exemplificare scenarii, cazuri de utilizare și fluxuri de activități.....	12
3. Implementare.....	15
3.1. Modulul 1 – Aplicația web.....	15
3.1.1. Structura proiectului .....	16
3.1.2. Rutarea .....	17
3.1.3. Serviciul de autentificare.....	19
3.1.4. Librării folosite.....	19
3.2. Modulul 2 – Server REST Java.....	20
3.2.1. Structura proiectului .....	20
3.2.2. Server REST.....	22
3.2.3. Managerul de dispozitive și protocolul de comunicare .....	24
3.2.4. Securitatea serverului .....	26
3.2.5. Unelte folosite .....	29
3.3. Modulul 3 – Dispozitivele inteligente.....	32
3.3.1. Interacțiunea cu Raspberry Pi .....	32
3.3.2. Dispozitive implementate .....	34
3.3.2.1. Laser de securitate .....	34
3.3.2.2. Lampă.....	35
3.3.2.3. Încuietoarea ușii.....	35
3.3.2.4. Bec.....	36
4. Instalarea aplicație .....	37

4.1. Aplicația web .....	37
4.2. Serverul Rest Java.....	38
4.3. Dispozitive inteligente .....	39
Concluziile lucrării și direcții de dezvoltare .....	40
Concluziile lucrării .....	40
Direcții de dezvoltare .....	40
Bibliografie .....	41

# Introducere

## Context

Persoanele își petrec cea mai mare parte a timpului la domiciliu sau la locul de muncă; pentru mulți, aceste locuri sunt sanctuarele lor. Pe parcursul secolului al XX-lea, progresele tehnologice au contribuit la sporirea confortului și protecției oferite de casele noastre. Observațiile asupra mediului casnic și modelarea comportamentului său sunt utile în a face mediile mai inteligente și mai receptive la nevoile noastre. Progresele recente au adus o astfel de „inteligență ambientală” mai aproape de realitate. Chiar dacă idea de casă inteligentă există de ceva vreme, case inteligente reale există doar de puțin timp.

De la miniaturizarea microprocesoarelor, puterea de calcul a fost încorporată în obiecte familiare, cum ar fi aparatele de uz casnic și dispozitivele mobile, regăsindu-se aproape la toate nivelurile societății. Inteligența ambientală extinde noțiunea de calcul pentru a oferi suport personalizat și automatizat în viețile noastre.

Ideea [de casă inteligentă] este următoarea: software-ul de calculator joacă rolul unui agent inteligent ce percepe starea mediul fizic și a rezidenților utilizând senzori, raționează în legătură cu starea acestora folosind tehnici de inteligență artificială și apoi ia măsuri pentru a atinge obiectivele specificate, cum ar fi maximizarea confortului locuitorilor, minimizarea consumul de resurse și menținerea sănătății și siguranței locuinței și a locuitorilor. [1]

## Motivație

Primele case inteligente au fost idei, nu construcții reale. Timp de decenii, literatura științifico-fantastică a explorat ideea automatizării locuințelor. Producători prolifici, cum ar fi Ray Bradbury<sup>1</sup>, și-au imaginat un viitor în care casele vor fi interactive. În scurta povestire a lui Bradbury, „There Will Come Soft Rain”<sup>2</sup>, se descrie o casă automată care continuă să funcționeze chiar și după ce oamenii au dispărut. Sună un pic înfricoșător, până când luați în considerare beneficiile reale ale automatizării locuințelor, iar apoi ideea devine mai mult confortabilă decât descurajatoare.

---

<sup>1</sup> Ray Douglas Bradbury (n. 22 August 1920 – d. 5 Iunie 2012): scriitor de romane științifico-fantastice, fantezie, horror și mister.

<sup>2</sup> „Va veni ploaie ușoară”: povestire scurta publicată pe data de 6 Mai 1950.

Atunci când nu sunteți acasă, aveți tot felul de suspiciuni care vă îngrijorează? Am oprit cafetiera? Am închis ușa de la intrare? Oare am lăsat apă aprinsă în baie? Cu o locuință inteligentă, aceste îngrijorări nu-și mai au locul. Puteți afla informații despre locuința dumneavoastră folosindu-vă de tabletă, telefon sau calculator personal. Puteți conecta dispozitivele și aparatele de la domiciliu pentru a comunica cu ele. De altfel, dispozitivele pot fi programate pentru a comunica între ele.

Lucrare de față își propune implementarea ideii de casă inteligentă și aducerea la cunoștiință, prin exemple practice, a avantajelor folosirii unui astfel de sistem. În exemplele oferite, voi folosi drept locuință inteligentă un Raspberry Pi<sup>3</sup> 2, Model B, iar dispozitivele controlate de utilizator vor fi module arduino<sup>4</sup>.

Inovația acestei lucrări constă în implementarea unui sistem de la zero, cu o arhitectură unică, folosind tehnologii de ultimă generație. Arhitectura sistemului este formată din 3 componente: partea de front-end<sup>5</sup> care are drept scop interacțiunea eficientă și facilă cu utilizatorul; partea de back-end<sup>6</sup> ce constă într-un server java care se ocupă de managementul utilizatorilor și al dispozitivelor, oferind siguranță, încredere și eficiență; partea de dispozitive ale utilizatorului ce folosește un protocol de comunicare securizată cu partea de back-end. Protocolul de comunicare oferă un management al erorilor foarte strict pentru a ține utilizatorul la curent cu starea dispozitivelor chiar și în cazuri de excepție.

## Ce este „Smart Home”?

„Smart Home”<sup>7</sup> este termenul utilizat în mod obișnuit pentru a defini o reședință care are iluminat, încălzire, aer condiționat, televizoare, calculatoare, sisteme audio și video de divertisment, sisteme de securitate și camere de luat vederi capabile să comunice între ele și pot fi controlate de la distanță: din orice cameră din casă, precum și din orice locație din lume, prin telefon sau prin internet.

Smart home este un sistem care oferă deținătorilor locuinței confort, securitate, eficiență energetică (costuri de operare scăzute) și comoditate în orice moment, indiferent dacă este cineva acasă sau nu.

---

<sup>3</sup> Raspberry Pi: <https://www.raspberrypi.org/>

<sup>4</sup> Modul arduino: este un circuit hardware complex, adus la o formă compactă pentru utilizare, cu o interfață de conexiune simplă, ce necesită doar atasarea unor fire între placă și circuit.

<sup>5</sup> Front-end: Parte a site-ului pe care o putem vedea și cu care interacționează vizitatorii.

<sup>6</sup> Back-end: Locul unde se administrează informația. De obicei este format din server și bază de date.

<sup>7</sup> Smart Home: Echivalentul acestui termen în limba română este „Casă Inteligentă”.

Instalarea de produse inteligente oferă locuinței și ocupanților săi diverse beneficii - aceleași avantaje pe care tehnologia și computerele personale le-au adus în ultimii 30 de ani - confort și economii de timp, bani și energie. [2]

## Facilitățile traiului într-o casă inteligentă

Multora nu le place ideea de „Smart Home” considerând că va fi prea costisitoare, greu de folosit, sau nu vor avea suficient control asupra propriului mediu. Cu toate acestea, casele inteligente devin foarte accesibile, datorită opțiunii de a construi puțin câte puțin, înglobând părțile într-un proiect mare, lăsându-se loc pentru îmbunătățiri viitoare. Fiecare dispozitiv vă stă la dispoziție pentru a vă face viața mai confortabilă.

În continuare voi enumera câteva motive pentru care ar trebui luați în considerare un smart home: [3]

### 1. Confort

Una dintre cele mai bune părți ale locuinței inteligente este că o puteți configura pentru a corespunde cu exactitate necesităților dumneavoastră. Lampa trebuie să fie configurată la o luminozitate corectă, nu prea luminoasă, dar nici prea întunecată. Puteți chiar să setați aprinderea luminilor treptat, evitând astfel blițul brusc de lumină orbitoare.

Puteți să setați o melodie liniștită atunci când alarmele se sting dimineața, astfel încât să vă treziți într-o atmosferă plăcută. Puteți amplasa difuzoarele în mai multe încăperi pentru a fi programate să difuzeze diferite melodii la diferite volume, astfel încât să puteți asculta mereu ceea ce doriți în orice loc din casă.

De asemenea, aveți posibilitatea ca temperatura casei să fie menținută la valoarea optimă. Instalația de încălzire și climatizare poate fi controlată de la distanță printr-un termostat, astfel încât în casă să nu fie niciodată prea rece sau prea cald.

### 2. Securitate

Există nenumărate motive pentru care o casă inteligentă vă poate menține în siguranță. Puteți avea senzori specifici pentru a detecta imediat mișcarea, cum ar fi noaptea sau în timp ce vă aflați la serviciu. Dacă detectorii de mișcare sunt avertizați, vi se poate trimite o notificare telefonului dumneavoastră și/sau se pot aprinde luminile pentru a da impresia că sunteți treaz, acest lucru sperându-l pe potențialul spărgător.

Puteți avea senzori pe ferestre și uși, astfel încât să știți dacă au fost deschise sau dacă au fost distruse.

O mulțime de case inteligente au camere video instalate care vă permit să monitorizați orice cameră utilizând dispozitivul inteligent sau terminalul de perete. Puteți seta secțiuni ale ecranului pentru a detecta mișcarea, astfel încât un animal de companie sau un copac în vânt să nu declanșeze alarma.

Dacă vă montați alarme „inteligente” de incendiu, puteți primi notificări despre potențiale incendii, indiferent dacă vă aflați în casă sau la locul de muncă. Dacă știți că este o alarmă falsă, puteți opri alarma printr-o simplă atingere de ecran, în loc să urcați pe un scaun și să vă întindeți pentru a ajunge la butonul de alarmă.

Mai mult, dacă plecați vreodată în vacanță puteți alege între a obține personal notificări despre cazurile excepționale sau a fi transmise unui vecin ori unui membru al familiei.

### 3. Ușor de utilizat

Partea cea mai convenabilă a unei locuințe inteligente este că fiecare parte a casei ar putea fi doar la o atingere de ecran (sau apăsarea unei taste) distanță. Fie că este vorba de telefonul tău inteligent, tabletă, calculator personal sau un terminal montat în perete.

Când te pregătești să dormi, puteți opri orice lumină din casă doar prin apăsarea unui buton. Puteți să vă uitați la ecranul dumneavoastră și să vedeți că toate ușile și fereastrele sunt blocate, astfel încât să nu trebuiască să mergeți și să le verificați pe fiecare în parte.

Puteți chiar crea setări pentru grupuri de dispozitive, de exemplu, când porniți televizorul, luminile se diminuează automat sau sunetul se aprinde odată cu televizorul. Dimineata puteți seta încălzirea, luminile, muzica pentru a se aprinde când vă ridicați din pat, fără să trebuiască să faceți nimic.

### 4. Accesibilitate

Casele inteligente oferă un ajutor imens pentru persoanele cu diverse dizabilități. Oamenii care nu văd pot avea interfață vocală cu care pot controla televizoarele, luminile, încălzirea, orice dispozitiv ce este conectat la electricitate și internet. Pentru persoanele cu dizabilități, precum deficiențe musculare, activarea comutatoarelor de lumină sau alarmei de incendiu se poate dovedi a fi un lucru dificil, dacă nu chiar imposibil. Acum, astfel de dispozitive, se pot controla folosind un telefon sau o tabletă. Pentru cineva într-un scaun cu roțile se poate configura deschiderea și închiderea ușilor automat folosind senzori.

Programele pot avea setări personalizate în funcție de utilizatori, astfel scutindu-i de preocupări triviale, cum ar fi setarea zilnică a încălzirii sau luminii.

Familiile care au un membru în vârstă sau poate pe cineva ce suferă de o boală precum Alzheimer<sup>8</sup> pot avea senzori instalați în casă pentru a fi mai simplă monitorizarea de la distanță a persoanelor în nevoie. Se pot crea alerte care să informeze în caz că ușa frontală este deschisă pe timp de noapte, pot fi atașați senzori la chei pentru a afla dacă se rătăcesc și nu sunt unde vă așteptați să fie. De altfel, se pot crea alerte de inundații în camerele de bucătărie, baie, chiar și monitorizarea casetelor de pastile pentru a vă asigura că sunt luate în fiecare zi.

De asemenea, pot fi instalate cu ușurință butoane de panică, în cazul în care a existat un accident în casă, iar familia sau personalul de îngrijire poate fi anunțat imediat, scutind persoana în nevoie să telefoneze pentru ajutor.

## Provocări

O casă inteligentă poate fi un coșmar pentru acei oameni ce nu se simt confortabil cu tehnologia de ultimă generație.

Unul dintre blocajele esențiale în instalarea unui sistem smart home este menținerea echilibrului dintre complexitatea și gradul de dificultate al utilizării lui. Dacă este exasperant de utilizat, atunci o să vă facă viața mai grea în loc să o facă mai ușoară. Când planificați sistemul, este important să luați în considerare câțiva factori:

- Ce tipuri de componente fac parte din sistem? Sunt de bază, de dimensiuni mici sau impunătoare, cum ar fi un sistem de alarmă sau o cameră video?
- Cât de intuitiv va fi sistemul pentru un non-utilizator?
- Dispozitivul îndeplinește o nevoie sau este doar o fantezie și potențial o jucărie frustrantă?
- Câți oameni vor fi nevoiți să utilizeze sistemul?
- Cine va ști cum să opereze sistemul? Cine va ști cum să mențină sistemul și să remedieze eșecurile?
- Cât de ușor este să faceți schimbări în interfață? De exemplu, dacă casa ta este programată să te trezească la 7 dimineața, cum o vei lăsa să știe că ești deplasat peste noapte la birou sau că dorești să dormi mai mult într-o sâmbătă?

Din aceste motive, ar putea fi mai ușor să începeți cu o rețea foarte simplă și apoi să o extindeți atunci când sunt necesare sau dorite îmbunătățiri. Ca multe dintre noile tehnologii, casele inteligente necesită o investiție semnificativă atât în bani, cât și în timp. În caz contrar,

---

<sup>8</sup> Alzheimer: tip de demență care cauzează probleme cu memoria, gândirea și comportamentul.



dacă nu aveți nici bani, nici timp, ați putea dori să rămâneți la casa dumneavoastră „veche” și „neinteligentă”.

Înainte de a cumpăra, verificați recenziile despre produse. Există o mulțime de produse care fac promisiuni înalte, dar în lumea reală nu au succes. Și dacă sunteți un utilizator de smartphone, luați în considerare produsele care apar și care au construite pentru ele o aplicație smartphone bine revizuită. Unele aplicații sunt atât de greoaie sau complicate încât provoacă mai multe dureri de cap decât vă ușurează viața.

Casele inteligente vin, de asemenea, cu unele probleme de securitate. Hackerii care găsesc o modalitate de a accesa rețeaua internă pot să dezactiveze sistemele de alarmă, luminile, lăsând locuința vulnerabilă la o spargere. De asemenea, ar putea provoca neplăceri, cum ar fi aprinderea și închiderea rapidă a dispozitivelor electronice, ceea ce ar putea dăuna funcționării sau, într-un caz extrem, ar putea provoca un incendiu.

Producătorii produselor electronice de uz casnic își îmbunătățesc liniile de producție, în speranța că automatizarea locuințelor va ajunge în sfârșit să se realizeze în masă. Mulțumită smartphone-urilor, tabletelor și numeroaselor aplicații de automatizare a locuințelor, disponibile acum, există o șansă că trendul va atrage mai mulți utilizatori.

Asta pentru că, în ciuda atâtor progrese tehnologice, nu există încă un sistem standard pentru automatizarea tuturor acestor gadgeturi<sup>9</sup>. Fără un astfel de standard, mulți consumatori sunt supuși riscului de a cheltui sute sau mii de dolari pe produse care vor sfârși depășite sau inutilizabile într-un scurt timp.

Desigur, se pune problema dacă o persoană are nevoie de toată această tehnologie. Societatea noastră este într-adevăr atât de leneșă și comodă încât nu putem apăsa pe un comutator de lumină? Indiferent de răspunsul acestei întrebări, această tehnologie are drept scop satisfacerea altelor puncte de interes, și anume: datorită timpului pe care îl vom economisi din automatizarea locuinței, vom avea timp de mai multe activități mai puțin triviale. [4]

---

<sup>9</sup> Gadget: obiect tehnologic mic care îndeplinește o anumită funcție, de obicei fiind ceva nou.

# 1. Contribuții

## 2. Arhitectura sistemului

### 2.1. Proiectare

În ceea ce urmează o să descriu planul pe care l-am abordat pentru a pune în practică ideea de Smart Home.

Sistemul trebuie să fie accesibil din orice locație din lume, din această cauză am ales ca interfața pentru utilizator să fie o aplicație web. Aplicația poate fi accesată de pe telefon, tabletă, calculator personal atât timp cât există conexiune la internet; ceea ce nu ar fi fost la fel de accesibil cu o aplicație Android sau iOS.

Pentru ca dispozitivele din casa dumneavoastră să nu fie accesate de persoane neautorizate, fiecare utilizator va avea un cont personal, ce va fi accesat pe baza unor credențiale.

Partea de back-end, altfel spus serverul de back-end, trebuie să ofere:

- un mediu persistent de stocare a datelor;
- înregistrarea, autentificarea<sup>10</sup> și managementul utilizatorilor;
- înregistrarea și controlul dispozitivelor expuse de utilizator;
- managementul conexiunilor dispozitivelor: inițiere, comunicare propriu zisă, finalizare;
- stabilirea unui protocol de comunicare cu dispozitivele „smart”.

Odată stabilite responsabilitățile serverului de back-end, mai rămâne de stabilit modalitatea de conectare a dispozitivelor smart cu partea de back-end și implementarea lor propriu-zisă.

Ce limbaj trebuie folosit pentru a programa cât mai multe tipuri de dispozitive, dacă nu chiar toate dispozitivele posibile? Am ales Java deoarece este un limbaj independent de platforma de lucru, aceeași aplicație rulând fără nicio modificare și fără a necesita recompilarea ei pe sisteme de operare diferite cum ar fi Windows, Linux, Mac OS, după cum se înțelege și din sloganul Java „*Write once, run everywhere*”. Alte calități sunt: simplitate, robustețe, garbage collector<sup>11</sup>, portabilitate.

Prin urmare dispozitivele trebuie să poată rula cod sursă scris în Java și ar trebui să aibă o conexiune la internet, având în vedere că acestea trebuie controlate de la distanță.

---

<sup>10</sup> Autentificarea: Are drept scop stabilirea identității actorilor care doresc să comunice sigur în rețea.

<sup>11</sup> Garbage collector: Program care face managementul memoriei interne automat.

Comunicarea cu serverul de back-end ar trebui să fie criptată astfel încât un posibil atacator să nu poată descifra nimic din ceea ce se transmite pe rețea. În plus, se pune și problema de certificare a serverului și evitare a unui posibil atac de tip „Man-in-the-Middle”<sup>12</sup>. O să descriu în continuare acest tip de atac aplicat pe acest sistem. Un atacator a aflat protocolul de comunicare a serverului cu dispozitivele. Un dispozitiv al unui utilizator este conectat la internet și primește o cerere de conectare de la atacator, care pretinde că este serverul de back-end. Dacă nu ar fi nici o metodă de identificare a serverului real, atunci întreg sistemul este compromis, atacatorul având posibilitatea sa controleze orice dispozitiv disponibil.

Prin urmare, pentru a evita o astfel de problemă, ce dovedește sistemul a fi inutilizabil, am să folosesc protocolul criptografic SSL (Secure Sockets Layer)/TLS (Transport Layer Security)<sup>13</sup> între server și client. SSL/TLS este un protocol criptografic care poate asigura confidențialitate, integritatea mesajelor și autentificarea părților. SSL/TLS acționează asupra unui flux TCP (Transport Control Protocol) și oferă servicii nivelurilor superioare.

În Figura 1 de mai jos este reprezentată o schiță pentru design-ului sistemului.

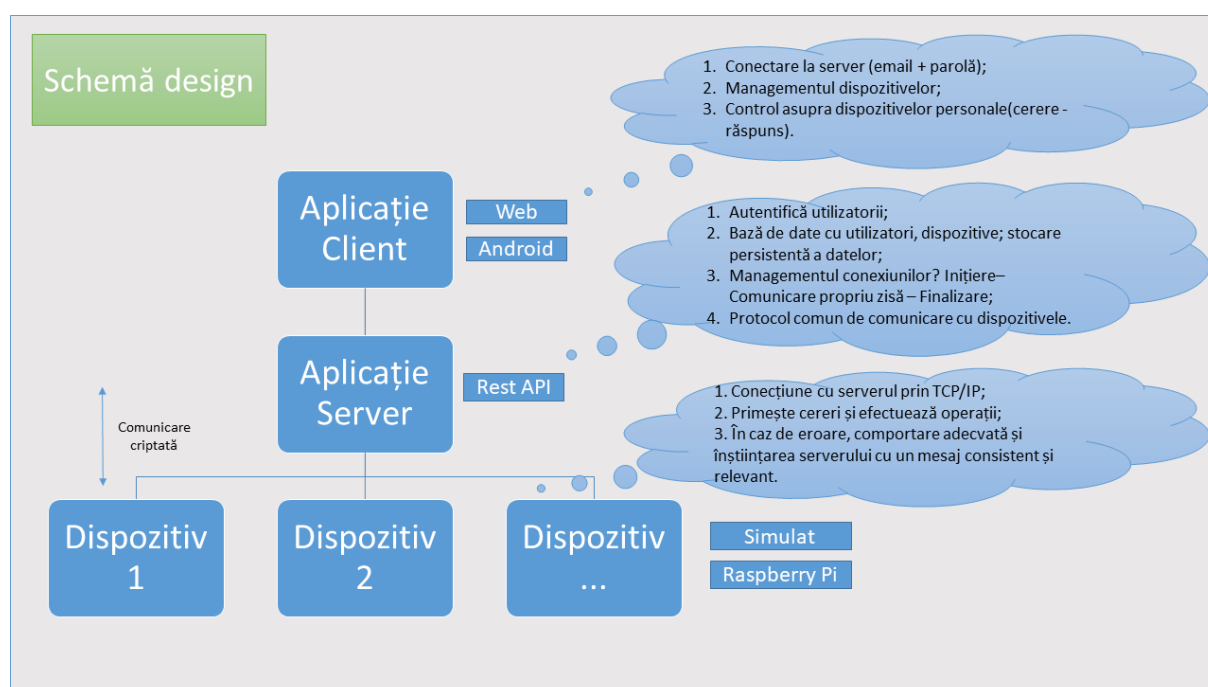


Figura 1: Schemă design

<sup>12</sup> Man-in-the-middle: detalii aici: <https://arxiv.org/ftp/arxiv/papers/1504/1504.02115.pdf>

<sup>13</sup> SSL/TLS: Când protocolul SSL a fost standardizat de către IETF, a fost redenumit în TLS.

## 2.2. Detalii arhitecturale

În urma proiectării s-a conturat o arhitectură formată din trei module principale. Primul modul constă într-o aplicație web care are drept scop principal comunicarea și interacțiunea cu utilizatorul. Al doilea modul este o aplicație Java, ce servește pe post de server, fiind „creierul sistemului”. Am numit acest modul astfel deoarece face legătura între ceea ce vede utilizatorul și cum se desfășoară firul logic al acțiunilor în spate. Al treilea modul este format din aplicații client ce vor juca rolul de dispozitive din cadrul unei case inteligente.

Utilizatorul interacționează cu sistemul prin intermediul primului modul, aplicația web. Aplicația structurează informațiile necesare utilizatorului astfel încât interacțiunea cu dispozitivele personale să fie cât mai simplă și facilă.

Modulul al doilea este responsabil în oferirea de răspunsuri pentru toate cererile ce vor veni de la utilizator, prin intermediul aplicației web. Paradigma de programare a serverului este REST (Representational State Transfer). REST este un stil arhitectural de dezvoltare al aplicațiilor Web cu focalizare asupra reprezentării datelor. Rezultatul unei procesări conduce la obținerea unei reprezentări a unei resurse. Formatul reprezentării e desemnat de tipuri MIME (Multipurpose Internet Mail Extensions) text/html, text/xml, text/csv, application/json, image/png, etc. Clienții (e.g., navigatoare Web, roboți, player-e etc.) interacționează cu reprezentările resurselor via verbe: „accesează” – GET, „modifică” – POST, „șterge” – DELETE, ș.a.m.d. . [5] Am ales această paradigma de programare deoarece oferă o separare între client și server, vizibilitate, scalabilitate, api-ul<sup>14</sup> REST fiind mereu independent de tipul de platformă sau limbajul de programare folosit.

Modulul al treilea, responsabil de dispozitivele inteligente, va fi simulat folosind un Raspberry Pi și module arduino. Un Raspberry Pi este, după cum îl prezintă producătorii, un microcomputer de dimensiunile unui card de credit, fiind cel mai mic și ieftin computer din lume. Un modul arduino este un circuit hardware complex, adus la o formă compactă pentru utilizare, ce are o interfață de conexiune simplistă necesitând doar atașarea unor fire între placă și circuit. Producătorul modulului oferă informații despre funcționalitatea pinilor de ieșire și modul lor de conexiune. Am ales aceste componente pentru că, în primul rând, este ieftin din punct de vedere financiar. În al doilea rând, module arduino oferă o interacțiune minimală cu dispozitivele hardware. În al treilea rând, Raspberry Pi oferă un mediu pentru crearea dispozitivelor foarte ușor și plăcut de utilizat. Pentru programarea dispozitivelor arduino am folosit o bibliotecă pentru controlarea fluxului de intrare ieșire a unui Raspberry Pi, Pi4J [6].

---

<sup>14</sup> API: Application Programming Interface

Figura 2, conține arhitectura detaliată a sistemului. După cum se poate observa, sistemul folosește o arhitectură bazată pe niveluri. Fiecare nivel efectuează anumite operații și oferă funcționalități nivelelor superioare. Comunicarea între niveluri se realizează astfel:

- aplicația client comunică cu aplicația server via verbe HTTP (Hyper Text Transfer Protocol), schimbând resurse în format MIME.
- aplicația server comunică cu dispozitivele conectate la Raspberry Pi printr-un protocol definit peste TCP, ce urmărește pattern-ul cerere-răspuns.

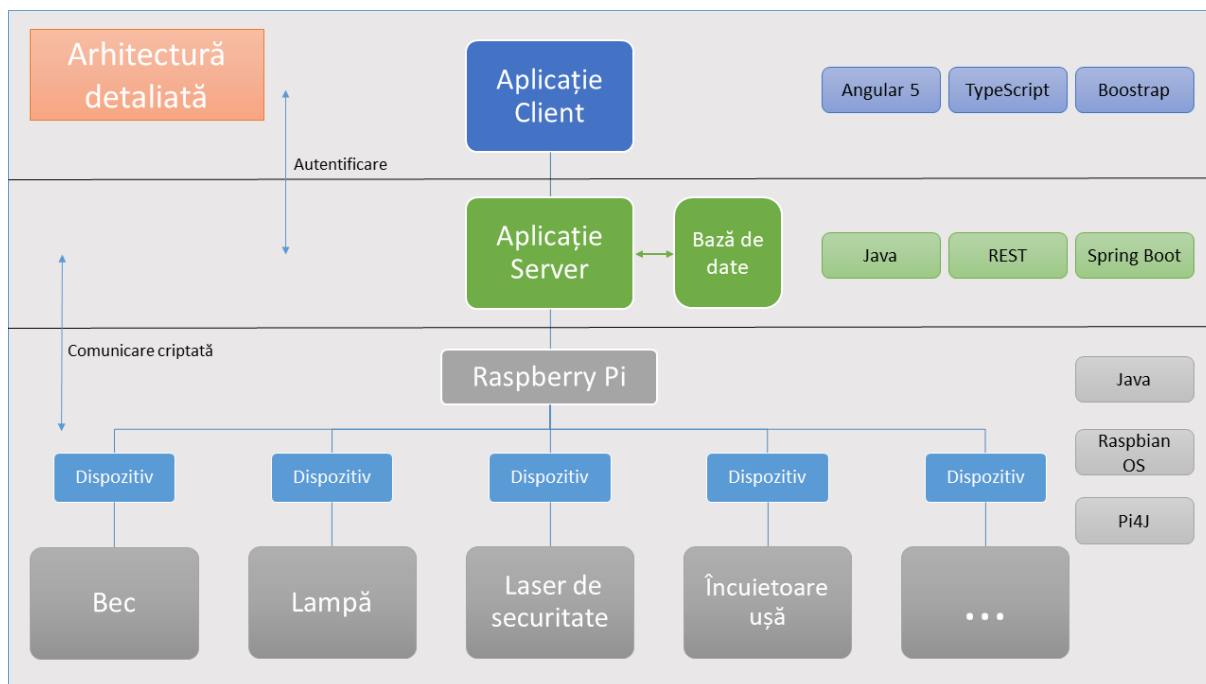


Figura 2: Arhitectura detaliată a sistemului

Scopul protocolul de comunicare este de a îndeplini următoarele funcționalități:

- Crearea și finalizarea unei conexiuni cu dispozitivul;
- Testarea conexiunii;
- Aflarea statusului dispozitivului;
- Aflarea tipului de dispozitiv;
- Interogarea parametrilor pe care dispozitivul îi acceptă;
- Setarea anumitor configurări;
- Interogarea valorilor curente.

Atunci când se construiește un dispozitiv trebuie stabiliți parametri personalizați. Parametrii unui dispozitiv nu vor fi cunoscuți apriori de către serverul Java. După ce se face conexiunea cu dispozitivul se vor afla parametrii acestuia în urma unei interogări aplicate dispozitivului. Acest lucru face protocolul de comunicare dinamic, o eventuală schimbare a

parametrilor nu va afecta în niciun fel comunicarea dintre server și client. Totuși, aplicația web va fi afectată, deoarece acolo este necesară cunoșterea exactă a numelui parametrului ce este primit.

## 2.3. Exemplificare scenarii, cazuri de utilizare și fluxuri de activități

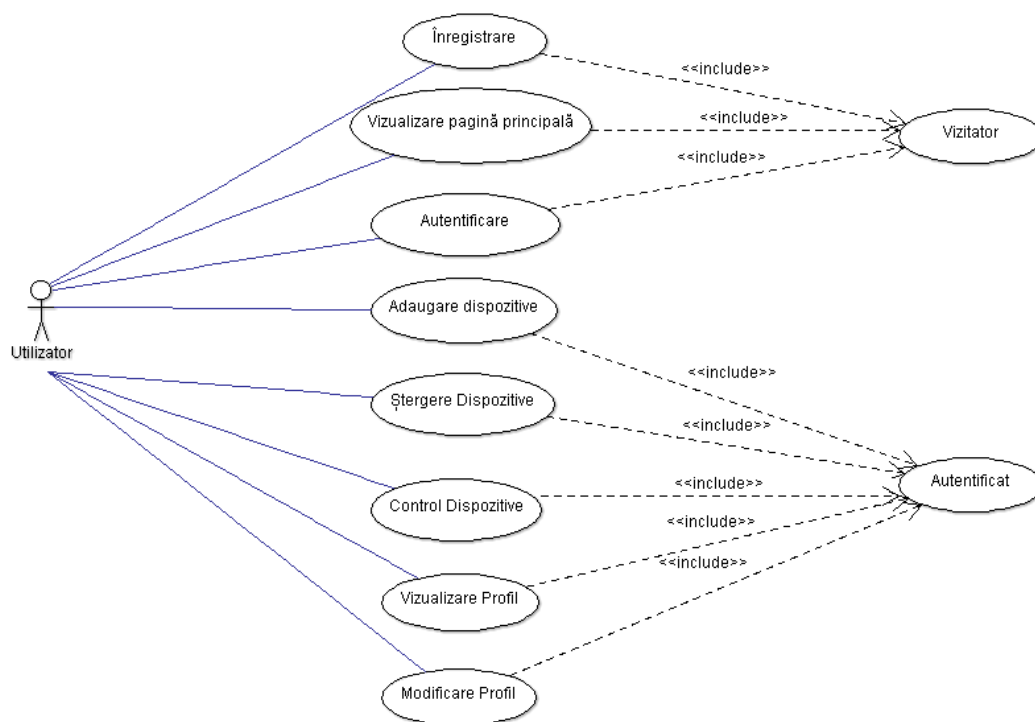


Figura 3: Interacțiunea utilizatorului cu sistemul

Figura 3, de mai sus, conține o diagramă cu cazurile de utilizare a sistemului. Utilizatorul poate să se înregistreze, să vizualizeze pagina principală și să se autentifice (login) dacă este un utilizator vizitator. El nu o să poată accesa alte funcționalități ale sistemului.

Utilizatorul autentificat poate să adauge, să șteargă și să programeze dispozitivele. În plus poate să își vizualizeze profilul și să îl modifice.

Figura 4, localizată mai jos, reprezintă o diagramă de activitate. Această diagramă arată fluxul de activitate rezultat din interacțiunea utilizatorului cu sistemul. Fluxul diagramei este de sus în jos. De exemplu, conectarea la un dispozitiv este poziționată la baza diagramei deoarece trebuie efectuați o secvență de pași până ajunge la aceasta. Evenimentele sunt aranjate cronologic în felul următor: logarea utilizatorului, vizualizarea listei de dispozitive, alegerea

unui dispozitiv din listă, selectarea opțiunii de management al dispozitivului și în cele din urmă conectarea efectivă la un dispozitiv.

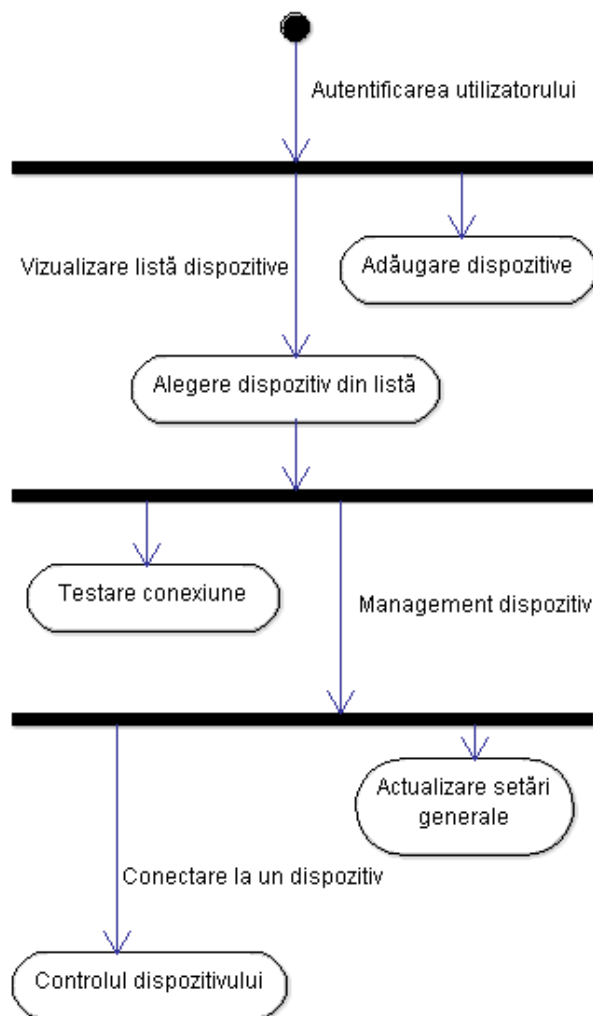


Figura 4: Diagrama de activitate a interacțiunii utilizatorului cu sistemul

Un scenariu important ține de managementul conexiunilor dispozitivelor. Utilizatorul efectuează manual conectarea la dispozitive, prin apăsarea butonului „Connect” de pe interfața grafică. La deconectare deja încep să apară unele întrebări. Cazul ideal este când utilizatorul apasă pe butonul de deconectare înainte să închidă pagina sau să iasă din cont, asigurându-se astfel că dispozitivul este închis. Ce se întâmplă atunci când se conectează la dispozitiv și apoi închide tab-ul din browser? Sau iese din cont? Sau chiar închide browser-ul? Aceste cazuri trebuie tratate corespunzător pentru a nu consuma resurse inutile sau a defecta dispozitivele. În primul rând o modalitate de a trata anumite cazuri este folosind interceptarea de evenimente cum ar fi închiderea tabului, ieșirea din cont, închiderea browser-ului.



Din cauza dependențelor dintre evenimente și mediul de lucru, cum ar fi browser-ul folosit de client în accesarea site-ului web, o soluție independentă de mediu de lucru ar fi folosirea unui mecanism de expirare a unei unități de timp prestabilite. Un astfel de mecanism va acționa pe partea de server, pentru a avea control complet. Mecanismul va funcționa în următorul mod: atunci când un utilizator se va conecta la un dispozitiv sau va face o setare dispozitivului, un timer (cronometru) va fi activat. Dacă utilizatorul face o acțiune în timp ce timer-ul este activ, atunci se va relua număratoarea de la început. Timer-ul va avea o valoare inițială prestabilită, de exemplu – 20 de minute. Sumarizând, comportamentul rezultat este: dacă utilizatorul nu face nici o operație asupra unui dispozitiv, într-un interval mai mare de 20 de minute, atunci conexiunea cu dispozitivul este încheiată, fără a afecta starea pe care o avea înainte de deconectare. Astfel, am obținut un comportament sigur, independent de browser, ce asigură managementul eficient al resurselor.

## 3. Implementare

### 3.1. Modulul 1 – Aplicația web

Modulul de aplicație web este foarte important, deoarece utilizatorul va interacționa cu sistemul folosind interfața web. Pentru ca experiența utilizatorului să fie plăcută și agreabilă, atunci când interacționează cu sistemul, am contruit un design bazat pe următoarele principii:

- *intuitiv*. Atunci când utilizatorul folosește aplicația să nu fie nevoit să citească pagini întregi din manualul aplicației pentru a ști cum se efectuează o operație simplă.
- *modern*. Folosirea tehnologiilor de ultimă generație oferă o experiență plăcută utilizatorului, datorită faptului că noile tehnologii sunt construite în funcție de feedback-ul utilizatorilor.
- *simplist*. Un design poate să înglobeze diverse funcționalități, însă poate eșua în a fi simplu de utilizat. De exemplu, dacă paginile sunt prea încărcate, setarea anumitor opțiuni devine un proces mult prea complicat.
- *responsive*. Design responsive înseamnă capacitatea unui site web de a își adapta conținutul în funcție de dimensiunea dispozitivului de pe care este accesat. Acest lucru duce la atragerea utilizatorilor ce folosesc dispozitive mobile.

Prin urmare, am hotărât să folosesc pentru acest modul două dintre cele mai populare tehnologii web: Angular<sup>15</sup> versiunea 5 și Bootstrap<sup>16</sup> versiunea 4.

Am ales Angular pentru că este o platformă ce ajută la crearea aplicațiilor SPA<sup>17</sup>. Datorită acestui fapt, pagina este fragmentată în mai multe componente ce se încarcă dinamic atunci când utilizatorul interacționează cu aplicația. Această abordare bazată pe componente face ca aplicația să se încarce foarte rapid, să fie modularizată, evitând duplicarea codului.

Apoi, am ales Bootstrap deoarece oferă un suport foarte bun în crearea de site-uri responsive, o documentație foarte bine structurată, cu exemple relevante, ceea ce face developmentul foarte rapid.

---

<sup>15</sup> Angular: <https://cli.angular.io/>

<sup>16</sup> Bootstrap: <https://getbootstrap.com/>

<sup>17</sup> Single-page application: Aplicație ce interacționează cu utilizatorul rescriind dinamic pagina curentă, în loc să încarce pagini întregi primite de la server.

### 3.1.1. Structura proiectului

În Figura 5 se poate observa structura unei componente Angular. O componentă conține un fișier .css<sup>18</sup>, unul .html<sup>19</sup> și altul .ts<sup>20</sup>. Acest mod de structurare permite unei componente să nu depindă de altă componentă, astfel reducând cuplajul din interiorul aplicației.

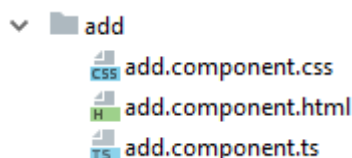


Figura 5: Structura unei componente angular

Prin definiție, cuplajul este măsura gradului de interdependență a claselor. Reformulând această definiție pentru contextul curent, cuplajul este gradul de dependență a unei componente de altă componentă. O componentă are un cuplaj redus dacă nu depinde de multe alte componente.

În Figura 6 se poate observa structura aplicației web, ce conține următoarele module:

- *backendservice*: conține un serviciu responsabil de cererile asincrone către serverul Java și alt serviciu ce este responsabil de autentificarea utilizatorului când se încearcă accesarea anumitor funcționalități;
- *device*: conține componente ce descriu operațiile ce pot fi aplicate asupra dispozitivelor: add – adăugare, manage – management, view – vizualizare;
- *deviceTypes*: conține mai multe componente ce descriu interfețele specifice în funcție de dispozitive: lampă, laser de securitate, încuietorea ușii, bec, etc;
- *footer*: definește subsolul site-ului;
- *header*: definește antetul site-ului. Antetul conține meniul principal;
- *home*: componentă responsabilă de pagina principală;
- *login*: componentă ce se ocupă de autentificarea utilizatorului;
- *notfound*: componentă ce va apărea de fiecare dată când utilizatorul va încerca să acceseze o resursă ce nu este disponibilă;
- *profile*: componentă ce se ocupă de managementul profilului utilizatorului;
- *register*: componenta cu care interacționează utilizatorul când se înregistrează.

<sup>18</sup> Cascade Style Sheets(foaie de stiluri CSS): standard pentru formatarea elementelor unui document html.

<sup>19</sup> Hyper Text Markup Language: limbajul principal al Web-ului pentru crearea de conținut ce poate fi utilizat oriunde.

<sup>20</sup> TypeScript: limbaj puternic tipizat, orientat obiect, compilat, superset al lui JavaScript.

### 3.1.2. Rutarea

Pentru a explica ce este rutarea este necesară definirea termenului „view”. O grupare de elemente prezentate utilizatorului ce se creează și se distrug împreună formează un view.

Rutarea constă în navigarea între view-uri, pe măsură ce utilizatorii execută operații în aplicație.

Tabelul 1 conține codul responsabil de rutare. Ideea principală constă în definirea unui router<sup>21</sup> care va fi inclus în modulul aplicației. El va efectua rutarea pe componenta specificată. Obiectul ce conține rutele este o listă, iar în interior conține dicționare ce au următorul format: *path*, *component*, *canActivate*. *Path* conține un string cu URL<sup>22</sup>-ul relativ la aplicație. *Component* definește ce clasă o să fie încărcată, iar *canActivate* definește dacă se poate accesa sau nu ruta respectivă. Dacă *canActivate* lipsește, atunci se poate accesa ruta fără nici o restricție, altfel se va decide dacă se poate sau nu accesa ruta.

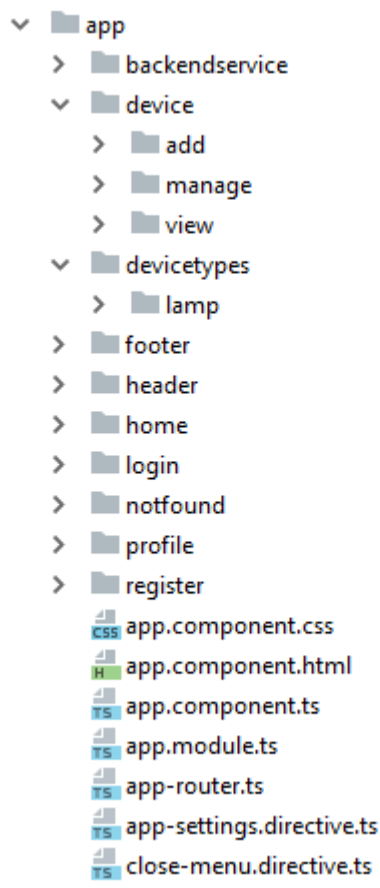


Figura 6: Structura aplicației web

<sup>21</sup> Obiect responsabil cu navigarea de pe un view pe următorul view.

<sup>22</sup> URL (Uniform Resource Locator): localizator uniform de resurse.

```

export const router: Routes = [
  {path: '', component: HomeComponent},
  {path: 'home', component: HomeComponent},
  {path: 'login', component: LoginComponent},
  {path: 'register', component: RegisterComponent},
  {path: 'profile', component: ProfileComponent, canActivate: [AuthGuard]},
  {path: 'devices', component: ViewComponent, canActivate: [AuthGuard]},
  {path: 'devices/add', component: AddComponent, canActivate: [AuthGuard]},
  {path: 'devices/manage/:id', component: ManageComponent, canActivate: [AuthGuard]},
  {path: '**', component: NotFoundComponent},
];

export const RouterGlobal: ModuleWithProviders =
RouterModule.forRoot(router);

```

Tabelul 1: Codul de rutare

Consider oportună explicarea path-urilor „\*\*”, „devices/manage/:id” întrucât sunt path-uri speciale. „\*\*” este folosit pentru a controla rutele invalide, ceea ce o să declanșeze componenta NotFound, indicându-i utilizatorului, într-o manieră elegantă, faptul că a introdus un link greșit. „devices/manage/:id” are scopul de a capta rutele de forma: <http://localhost:4200/devices/manage/7018dea9-8cb6-4677-a5ea-cfb743ee4dca>, unde „7018dea9-8cb6-4677-a5ea-cfb743ee4dca” este id-ul unui dispozitiv. Se folosește „:id” pentru a avea acces direct la id, evitând astfel parsarea url-ului.

De exemplu, dacă aplicația rulează pe host-ul local, la portul 4200, și utilizatorul introduce în browser următorul url: <http://localhost:4200/home>, atunci, conform router-ului definit, se va încărca componenta HomeComponent, fără nicio restricție de accesare. Dacă se introduce în browser url-ul: <http://localhost:4200/profile>, atunci ar trebui să se încarce ProfileComponent. Totuși, AuthGuard va lua decizia finală dacă se poate sau nu accesa componenta. O să explic în următoarea secțiune ce este AuthGuard și care este rolul lui.

### 3.1.3. Serviciul de autentificare

Într-o aplicație web, există rute publice și rute ce pot fi accesate doar dacă sunt satisfăcute anumite condiții. Una dintre aceste condiții este ca utilizatorul să fie autentificat. Acesta este și cazul de față, AuthGuard fiind un serviciu care verifică utilizatorul este autentificat atunci când accesează anumite rute.

Paginile publice din aceasă aplicație sunt: pagina principală, pagina de autentificare, pagina de înregistrare. Paginile verificate de serviciu sunt: pagina de profil și paginile ce țin de dispozitive. Este necesară verificarea stării utilizatorului, deoarece utilizatorul vizitator (neautentificat) nu are acces la principalele funcționalități oferite de aplicație, pentru aceasta fiind nevoie de un profil înregistrat, însoțit de autentificarea propriu-zisă.

### 3.1.4. Librării folosite

În crearea aplicației am folosit următoarele librării:

- ngx-toastr [7]
- highcharts [8]

Ngx-toastr este o librărie ce am folosit-o pentru a afișa mesaje toast utilizatorilor. Un mesaj toast este un mesaj scurt afișat într-un pop-up. Toast-ul dispare automat după un anumit timp sau poate fi închis manual de către utilizator. Am folosit toast-uri pentru a afișa mesaje de diferite tipuri utilizatorilor. Mesajele afișate pot fi de următoarele tipuri: eroare, informare, success, avertizare. Un exemplu de mesaj afișat utilizatorului este un mesaj de eroare, atunci când se încearcă conectarea la un dispozitiv dar nu se reușește din diverse motive tehnice.

HighCharts oferă suport pentru grafice create în javascript. Am integrat un grafic, prezent în Figura 7, în design-ul interfeței obiectului lampă, mai precis pentru funcționalitatea de a seta o anumită culoare. Motivul pentru care am ales această librărie este de a oferi utilizatorului o experiență mai bună, întrucât această reprezentare mi se pare cea mai aproape de realitate, în ceea ce privește interacțiunea cu o lampă.

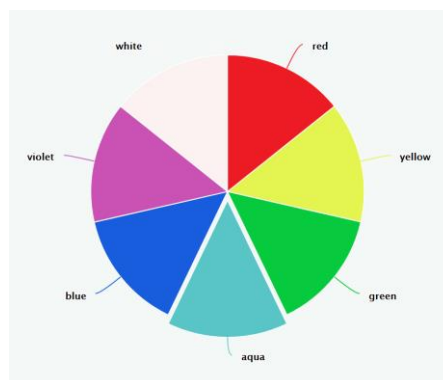


Figura 7: Reprezentarea lampii

## 3.2. Modulul 2 – Server REST Java

După cum am menționat și la începutul subcapitolului „Detalii arhitecturale”, acest modul este „creierul sistemului” deoarece face legătura între ceea ce vede utilizatorul și cum se desfășoară firul logic al acțiunilor în spate.

Am ales să folosesc pentru acest server Spring Boot<sup>23</sup>, un framework de Java foarte popular în construirea de aplicații web și enterprise. Nu este prima dată când interacționez cu acest framework, rămânând cu o impresie bună după fiecare proiect în care l-am utilizat.

Paradigma de programare a serverului este REST<sup>24</sup>. Această paradigmă este descrisă în subcapitolul „Detalii arhitecturale”.

### 3.2.1. Structura proiectului

În Figura 8 se poate observa structura serverului Java, ce este formată din:

- *controllers*: o colecție de controllere ce sunt responsabile de preluarea cererilor de la client gestionând resursele necesare satisfacerii cererilor;
- *dtos*: acronim pentru „Data Transfer Object”, folosite pentru a încapsula date ce vor fi trimise între server și aplicația web;
- *entities*: pachet ce conține o colecție de entități din baza de date. Entitățile sunt obiecte ce pot exista independent;
- *hal*: acronim pentru „Hardware Abstraction Layer”. Acest pachet este folosit pentru comunicarea cu dispozitivele inteligente;
- *repositories*: pachet ce conține mai multe colecții de date. Am folosit această abordare pentru a evita dublare logicii accesului la date;
- *security*: pachet responsabil de securitatea serverului, conține filtre http și partea de autentificare;
- *service*: pachet ce conține serviciile utilizate în server cum ar fi managerul de dispozitive sau serviciul responsabil de utilizatori;
- *SwaggerConfig*: clasă responsabilă de configurarea Swagger-ului<sup>25</sup>;
- *Application.properties*: fișier responsabil de configurările aplicației. Aici se găsesc configurări cum ar fi: portul la care rulează serverul, șirul de caractere ce

---

<sup>23</sup> Spring Boot: <https://spring.io/projects/spring-boot>

<sup>24</sup> REST: [https://www.ics.uci.edu/~fielding/pubs/dissertation/rest\\_arch\\_style.htm](https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm)

<sup>25</sup> Swagger: <https://swagger.io>

reprezintă conexiunea la baza de date, numele utilizatorului ce poate accesa baza de date, ș.a.m.d.

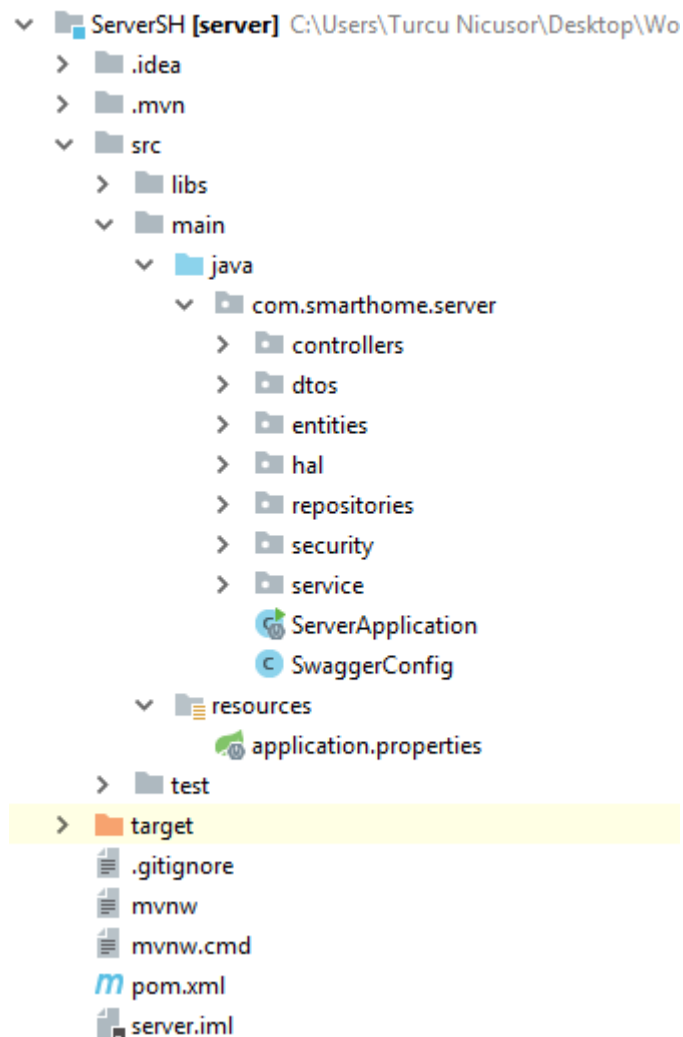


Figura 8: Structura serverului Java

Fișierul pom.xml conține dependențele din cadrul server-ului. Ele sunt specificate în format xml, după cum se vede și în Tabelul 2. Atunci când dependențele se adaugă în fișierul pom.xml, acestea sunt cautate online; după ce au fost găsite, se salvează și la nivel local.

```
<dependency>

    <groupId>mysql</groupId>

    <artifactId>mysql-connector-java</artifactId>

    <scope>runtime</scope>

</dependency>
```

Tabelul 2: Dependența la baza de date, definită pom.xml



### 3.2.2. Server REST

Programarea la nivel de server necesită cunoașterea unei anumite terminologii. Termenii pe care îi voi explica vor fi următorii: repository, entități, DTO (data transfer object), controller, bază de date.

O bază de date se poate defini în 2 moduri. În general, o bază de date reprezintă organizarea unei colecții de date, indiferent de tipul lor. În termeni tehnici, o bază de date este un sistem electronic care permite ca datele să fie ușor accesate, modificate și actualizate.

DTO (data transfer object), în traducere obiect de transfer de date, este termenul desemnat pentru a defini un obiect ce încapsulează date ce sunt trimise de la un subsistem la altul. În cazul meu, un DTO este folosit pentru a transfera informații între server și aplicația web.

O entitate este orice obiect pe care dorim să îl păstrăm într-o bază de date. Entitățile pot fi: concepte concrete, abstracte, recunoscute; evenimente, lucruri, locuri, etc. Exemple clasice ce sunt folosite deseori și în exemplificări sunt: student, cursuri, profesori. Din punct de vedere informatic, o entitate este o tabelă din baza de date relațională.

Un repository este o abstractizare asupra unei baze de date. El se ocupă cu managementul obiectelor și expune diverse funcționalități ce coordonează obiectele și relațiile dintre ele pentru serviciile ce îl vor folosi. Avantajul principal constă în distribuirea responsabilităților urmărind astfel principiile „separation of concerns” și „single responsibility principle”<sup>26</sup>.

Un controller face de obicei, doar un lucru: primește o intrare și generează rezultat. Mai exact, un controller primește cererile utilizatorului, apoi delegă, de obicei, un serviciu pentru a rezolva cererea, iar când cererea este rezolvată, el răspunde cu mesajul corespunzător.

Acum că am terminat definirea termenilor, voi continua explicarea construcției serverului.

Atunci când un utilizator efectuează o operație pe partea de aplicație web o cerere va fi trimisă serverului. Prima componentă întâlnită va fi filtrul HTTP. Filtrul HTTP este descris în subcapitolul „Securitatea serverului”. Cererea trimisă serverului trece de la filtrele HTTP la controller-ul corespunzător. În aplicație există trei controllere: pentru dispozitive, pentru utilizatori și pentru autentificarea utilizatorilor. Fiecare controller are o responsabilitate exactă.

Tabelul 3, de mai jos, conține procedura responsabilă de conectarea la un dispozitiv. „@PostMapping” este o anotație ce are următoarea semnificație: metoda se apelează atunci

---

<sup>26</sup> Mai multe detalii aici: <https://deviq.com/separation-of-concerns/>

când verbul HTTP din cerere este PUT. Calea „/connect” reprezintă url-ul relativ la metodă. Această metodă necesită în antetul HTTP token-ul de autorizare, pentru identificarea utilizatorului ce a făcut cererea, iar ca parametru de interogare, id-ul dispozitivului, pentru identificarea dispozitivului la care de dorește conectarea. Metoda aparține controller-ului „DevicesController” de aceea, url-ul complet prin care va fi accesată această metoda este: [http://localhost:9000/device/connect?device=hash dispozitiv](http://localhost:9000/device/connect?device=hash_dispozitiv).

```
@PutMapping("/connect")

    ResponseEntity connect(@RequestHeader("Authorization") String token,
    @RequestParam("device") String hash) {

        ResponseEntity response = checkDevice(token, hash);

        if (response.getStatusCode() != HttpStatus.OK) return response;

        try {

            deviceManager.getHalDevice(hash).connect();

        } catch (Exception e) {

            return
ResponseEntity.status(HttpStatus.EXPECTATION_FAILED).body("Connect failed.
Reason: " + e.getMessage());

        }

        return ResponseEntity.status(HttpStatus.OK).body("");

    }
```

Tabelul 3: Procedura responsabilă de conectarea la un dispozitiv

Metoda verifică mai întâi dacă dispozitivul există și dacă utilizatorul are drept să îl acceseze, iar apoi se delegă responsabilitatea managerului de dispozitive ce va efectua operația de conectare. Metoda *checkDevice* nu este inclusă în blocul *try catch* pentru că face o verificare a condițiilor minime de continuare a execuției metodei. Blocul *try catch* tratează cazurile de excepție, returnând aplicației web un mesaj corespunzător. În cazul în care totul merge conform planului, atunci se va apela ultima linie din metodă, care va întoarce un status HTTP 200, adică totul s-a realizat cu succes, conectarea la dispozitiv fiind reușită.

Am folosit repository-urile pentru accesarea datelor corespunzătoare utilizatorilor și dispozitivelor necesare. Datele preluate sunt sub formă de entități, definite corespunzător, fiind

extrase dintr-o bază de date. Pentru baza de date am folosit MySQL<sup>27</sup>, toate setările pentru conectare fiind definite în fișierul „application.properties”.

Un exemplu de DTO folosit este cel pentru înregistrare. El are următorii membrii privați: *firstName*, *lastName*, *email*, *password*. Fiecare membru are validare corespunzătoare, de exemplu, la câmpul de email se verifică dacă formatul email-ului este valid. Acest obiect se completează după ce utilizatorul completează formularul de înregistrare, apoi se trimite serverului care procesează cererea.

### 3.2.3. Managerul de dispozitive și protocolul de comunicare

Pentru interacțiunea facilă cu dispozitivele, am creat un manager de dispozitive. Așa cum îi spune și numele, el are rolul de administrare a dispozitivelor. Acest manager o să interacționeze cu toate obiectele de tip dispozitiv. Cu ajutorul lui dispozitivele vor fi programate.

Managerul de dispozitive folosește un mecanism de cache, descris în ceea ce urmează. Atunci când un utilizator se autentifică, toate dispozitivele lui, existente în baza de date, se vor încărca într-un HashMap intern. Cache-ul este necesar deoarece o conexiune cu dispozitivul trebuie inițiată atunci când clientul o solicită. Această stare a conexiunii trebuie să nu fie alterată.

După ce utilizatorul se autentifică, orice operație asupra unui dispozitiv este delegată managerului de dispozitive, ce are acces la toate operațiile expuse de clasa HalDevice.

La deconectarea utilizatorului, se va încheia conexiunea cu toate dispozitive la care utilizatorul este conectat, iar apoi se vor scoate toate dispozitivele utilizatorului din cache.

HalDevice este clasa ce implementează protocolul de comunicare cu dispozitivele. Protocolul este de tip cerere răspuns. Mesajele transmise sunt fluxuri de octeți. Pentru a identifica comanda, mai întâi se trimite antetul reprezentat de un octet ce va indica comanda, iar apoi conținutul cererii. Protocolul de comunicare are comenzile definite în Tabelul 4. Primele două comenzi reprezintă răspunsurile posibile: succes sau eroare, următoarele șapte reprezentând cererile ce se pot efectua.

În caz de eroare, după antet, se trimite și mesajul de eroare corespunzător. Atunci când acest mesaj ajunge pe partea de server se va arunca o excepție creată din mesajul primit, ce va fi tratată corespunzător din logica serverului.

---

<sup>27</sup> MySQL: <https://www.mysql.com/>

Celelalte comenzi sunt implementate identic, primul octet este identificatorul comenzii, octeții următori având semnificații diferite, în funcție de comandă.

```
public class Protocol {

    public static final byte EXCEPTION = 101;

    public static final byte SUCCESS = 109;

    public static final byte OPEN = 102;

    public static final byte CLOSE = 103;

    public static final byte IS_OPENED = 104;

    public static final byte GET_TYPE = 105;

    public static final byte GET_PARAMS = 106;

    public static final byte COMMAND = 107;

    public static final byte QUERY_DATA = 108;

}
```

Tabelul 4: Comenzile protocolului de comunicare

<b>Metodă</b>	<b>Descriere</b>	<b>Cerere</b>	<b>Răspuns</b>
<i>Open</i>	Comandă ce va deschide dispozitivul.	Doar octetul de comandă.	Octet de success. În caz contrar, octet de eroare urmat de mesaj.
<i>Close</i>	Comandă ce va închide dispozitivul.	Doar octetul de comandă.	Octet de success. În caz contrar, octet de eroare urmat de mesaj.
<i>IsOpen</i>	Verifică dacă dispozitivul este conectat.	Doar octetul de comandă.	Octet de succes urmat de o valoare booleană. În caz contrar, octet de eroare urmat de mesaj.
<i>GetType</i>	Cere tipul dispozitivului.	Doar octetul de comandă.	Octet de succes urmat de un șir de caractere ce semnifică tipul dispozitivului. În caz contrar, octet de eroare urmat de mesaj.

<i>GetAcceptedParams</i>	Cere parametrii configurabili ai dispozitivului. Ei vor fi de forma: nume, tipul parametrului și dacă este de citire sau de scriere.	Doar octetul de comandă.	Octet de succes urmat de un șir de parametrii separați prin „;”. Un parametru este de forma: „cheie=boolean;tip”. Valoarea booleană semnifică dacă parametrul este doar de citire sau nu. În caz contrar, octet de eroare urmat de mesaj.
<i>Command</i>	Programează dispozitivul.	Octetul de comandă urmat de o secvență de perechi cheie-valoare separate prin „;”.	Octet de success. În caz contrar, octet de eroare urmat de mesaj.
<i>QueryData</i>	Cere valorile parametrilor dispozitivului.	Octetul de comandă urmat de numele parametrilor separați prin „;”.	Octet de success urmat de un șir valori de forma „cheie=valoare”, separate prin „;”. În caz contrar, octet de eroare urmat de mesaj.

Tabelul 5: Definirea protocolului de comunicare

Tabelul 5, de mai sus, conține definirea protocolului de comunicare. Coloanele cerere și răspuns se referă la ce se trimite pe rețea atunci când se solicită o operație, respectiv răspunsul corespunzător cererii.

Securitatea dintre server și dispozitive va fi prezentată în capitolul următor.

### 3.2.4. Securitatea serverului

Din punct de vedere al securității între server și client folosesc autorizarea bazată pe token-uri de access. În autentificarea bazată pe token-uri, se asigură că fiecare cerere către server conține un token semnat, pe care serverul îl va verifica pentru autenticitate, iar doar apoi va răspunde la cerere. Un token este o secvență de informație care de unul singur nu are nicio însemnătate, dar combinat cu sistemul de tokenizare joacă un rol vital în securitatea serverului.

Atunci când un utilizator dorește să se autentifice, cererea este trimisă de la aplicația web la server. Serverul primește cererea, verifică dacă utilizatorul este înregistrat, apoi verifică dacă parola se potrivește. Dacă toți acești pași se termină cu succes, atunci utilizatorul este autorizat, prin urmare serverul generează un token ce este trimis drept răspuns aplicației web. Din acest punct aplicația web are tokenul de autorizare și o să îl folosească la fiecare cerere către server. Un token expiră la un timp presabilit de către server.

După autentificare aplicația web trimite la fiecare cerere token-ul primit. Din acest motiv am creat un filtru HTTP care interceptează toate cererile de la aplicația web, în afară de autentificare și de înregistrare, și verifică existența token-ului în header-ul HTTP. Dacă se efectuează cererea și se trimite fără token, atunci serverul răspunde că utilizatorul nu este autorizat în efectuarea operațiunii dorite. Dacă se trimite token dar după verificarea serverului acesta este invalid, se va trimite același răspuns. Doar în cazul în care tokenul este valid, cererea trece la metoda din controller ce o să execute acțiunea dorită.

Tokenul creat de server este semnat folosind HMAC-SHA-512<sup>28</sup>.

Din cauza faptului că token-ul se trimite ca antet HTTP există posibilitatea ca el să fie recepționat de un atacator. Acest lucru este posibil pentru că momentan se folosește protocolul HTTP, unde antetele HTTP sunt trimise în clar peste rețea.

Dacă se folosește HTTPS (Secure Hyper Text Transfer Protocol), nu vor mai fi astfel de probleme. HTTPS are o fază de inițiere unde se negociază criptarea între serverul web și browser-ul web. Apoi tot traficul este trimis criptat, în loc să fie trimis în clar. În esență același lucru ca traficul HTTP, dar cu avantaje de securitate. [9]

Referitor la securitatea între server și client, așa cum am precizat și în capitolul „Arhitectura sistemului”, folosesc protocolul criptografic SSL/TLS. Pentru a folosi protocolul criptografic SSL/TLS este nevoie de un certificat SSL/TLS. Certificatul este un fișier de dimensiune redusă ce leagă cheia criptografică de informațiile unei organizații. Avantajele folosirii acestui protocol sunt [10]:

- criptează datele sensibile. Datele trimise folosind acest protocol sunt criptate, fiind nedescifrabile pentru cei ce nu au cheia.
- asigură că informația este trimisă la destinatarul real. Certificatul SSL/TLS acționează sub forma unui handshake între browser-ele care comunică. Handshake-ul reprezintă un termen criptografic ce semnifică un proces automat de negociere între doi participanți pentru stabilirea protocolului de comunicare înainte de începerea comunicării propriu-zise.
- protejează de escrocheri. Escrocherii au foarte mari dificultăți în crearea unei replici autentice a unui certificat SSL/TLS.
- oferă confidențialitate și încredere.

Pentru conectarea SSL/TLS am folosit pachetul javax.net.ssl. Pe partea de dispozitive am folosit clasa SSLServerSocketFactory pentru a crea un socket. Serverul obține socket-urile pentru conectarea la dispozitive din clasa SSLSocketFactory. Serverul și dispozitivele trebuie

---

<sup>28</sup> HMAC-SHA-512: <https://tools.ietf.org/html/rfc4868>

se specifice locația certificatului SSL/TLS folosit. În plus, dispozitivele trebuie să specifice și parola certificatului SSL/TLS.

Tabelul 6 conține comanda prin care am generat certificatul SSL/TLS. Când comanda se rulează următoarele informații vor fi cerute: numele și prenumele, numele unității organizaționale, numele organizației, numele orașului/localității, numele țării, prescurtarea țării. Apoi se va cere introducerea unei parole ce mai apoi trebuie confirmată.

```
keytool -genkey -keyalg RSA -alias smarthome -keystore secret_key -storepass  
„*****” -validity 360 -keysize 2048
```

Tabelul 6: Comanda pentru generarea certificatului SSL/TLS

Figura 9 reprezintă o captură de ecran dintre comunicația dintre server și un dispozitiv. Această captură a fost realizată folosind programul Wireshark<sup>29</sup>. Un dispozitiv este online fiind conectat la Raspberry P. Traficul din captură este filtrat după ip-ul *169.254.207.167* și port-ul *8000*, ce aparțin dispozitivului. Ip-ul *169.254.207.167* este ip-ul de conexiune al serverului.

Captura indică toate mesajele ce se transmit pe rețea atunci când utilizatorul apasă butonul *Connect* în interfața web.

În figură se pot observa 4 secțiuni:

- Secțiunea 1: stabilirea conexiunii TCP/IP între server și dispozitiv.
- Secțiunea 2: negocierea protocolului SSL/TLS.
- Secțiunea 3: comunicarea inițială cu dispozitivul. Serverul cere tipul dispozitivului, ce parametri acceptă și valorile curente ale parametrilor.
- Secțiunea 4: conține datele în format hexazecimal în partea stângă, respectiv datele în format ASCII în partea dreaptă, referitoare la mesajul cu numărul *518*.

După cum se observă nu se înțelege nimic din ceea ce se transmite pe rețea.

---

<sup>29</sup> Wireshark: <https://www.wireshark.org/>

No.	Source	Destination	Protocol	Info
1.	493 169.254.48.113	169.254.207.167	TCP	51776 → 8000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
	494 169.254.207.167	169.254.48.113	TCP	8000 → 51776 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=128
	495 169.254.48.113	169.254.207.167	TCP	51776 → 8000 [ACK] Seq=1 Ack=1 Win=65536 Len=0
	496 169.254.48.113	169.254.207.167	TLSv1.2	Client Hello
	497 169.254.207.167	169.254.48.113	TCP	8000 → 51776 [ACK] Seq=1 Ack=199 Win=30336 Len=0
	501 169.254.207.167	169.254.48.113	TLSv1.2	Server Hello, Certificate, Server Key Exchange, Server Hello Done
2.	502 169.254.48.113	169.254.207.167	TLSv1.2	Client Key Exchange
	503 169.254.207.167	169.254.48.113	TCP	8000 → 51776 [ACK] Seq=1204 Ack=338 Win=31360 Len=0
	504 169.254.48.113	169.254.207.167	TLSv1.2	Change Cipher Spec
	505 169.254.207.167	169.254.48.113	TCP	8000 → 51776 [ACK] Seq=1204 Ack=344 Win=31360 Len=0
	506 169.254.48.113	169.254.207.167	TLSv1.2	Encrypted Handshake Message
	507 169.254.207.167	169.254.48.113	TCP	8000 → 51776 [ACK] Seq=1204 Ack=429 Win=31360 Len=0
	508 169.254.207.167	169.254.48.113	TLSv1.2	Change Cipher Spec
	509 169.254.48.113	169.254.207.167	TCP	51776 → 8000 [ACK] Seq=429 Ack=1210 Win=64256 Len=0
	510 169.254.207.167	169.254.48.113	TLSv1.2	Encrypted Handshake Message
3.	511 169.254.48.113	169.254.207.167	TLSv1.2	Application Data
	512 169.254.207.167	169.254.48.113	TCP	8000 → 51776 [ACK] Seq=1295 Ack=498 Win=31360 Len=0
	513 169.254.207.167	169.254.48.113	TLSv1.2	Application Data
	517 169.254.48.113	169.254.207.167	TLSv1.2	Application Data
	518 169.254.207.167	169.254.48.113	TLSv1.2	Application Data
	519 169.254.48.113	169.254.207.167	TLSv1.2	Application Data
	520 169.254.207.167	169.254.48.113	TLSv1.2	Application Data
	521 169.254.48.113	169.254.207.167	TLSv1.2	Application Data
	522 169.254.207.167	169.254.48.113	TLSv1.2	Application Data
	528 169.254.48.113	169.254.207.167	TCP	51776 → 8000 [ACK] Seq=721 Ack=1651 Win=65280 Len=0
> Frame 518: 171 bytes on wire (1368 bits), 171 bytes captured (1368 bits) on interface 0 > Ethernet II, Src: Raspberr_0a:f0:6b (b8:27:eb:0a:f0:6b), Dst: TanakaS/_0c:27:1d (00:05:0f:0c:27:1d) > Internet Protocol Version 4, Src: 169.254.207.167, Dst: 169.254.48.113 > Transmission Control Protocol, Src Port: 8000, Dst Port: 51776, Seq: 1364, Ack: 567, Len: 117 > Secure Sockets Layer				
	0000	00 05 0f 0c 27 1d b8 27 eb 0a f0 6b 08 00 45 00	...	...k...E
	0010	00 9d 98 b7 40 00 40 06 4d 8e a9 fe cf a7 a9 fe	...	...@. M.....
	0020	30 71 1f 40 ca 40 af 77 0d f5 ba d1 d9 4a 50 18	0q @-w	...JP...
	0030	00 f5 1b 49 00 00 17 03 03 00 70 e0 3c 04 f7 04	...I...	...p<...
	0040	ff a0 c9 80 5c 59 b8 25 b4 8f 74 67 b0 b2 21 8a	...V-%	...tg..!

Figura 9: Captură a comunicației dintre server și un dispozitiv

### 3.2.5. Unelte folosite

În dezvoltarea aplicației server am folosit două unelte ce m-au ajutat foarte mult: swagger și ngrok.

Swagger permite descrierea structurii API-ului dorit astfel încât și mașinile să le poată citi. Prin citirea structurii API-ului se poate construi automat o documentație interactivă. De asemenea se pot genera automat biblioteci, în mai multe limbi, pentru a explora și alte posibilități cum ar fi testarea automată. [11]

Am folosit Swagger pentru a genera o documentație interactivă a server-ului și, mai ales, pentru a putea testa serverul dinamic fără nevoia modulului de aplicație web. Figura 10 surprinde o parte din interfața expusă de swagger unde se observă documentația interactivă a server-ului. Se pot observa ce controllere există, iar în interiorul fiecărui controller se pot vedea metodele ce pot fi accesate. Verbele HTTP prin care se accesează fiecare metodă sunt marcate corespunzător.

Figura 11 surprinde o altă din interfața swagger, ce conține documentarea metodei de ștergere a unui dispozitiv. Metoda necesită 2 parametri: authorization și device. În partea dreaptă apare tipul de parametru necesar, antetul HTTP și tipul de dată necesar. În secțiunea „Curl” se afișează request-ul corespunzător metodei: ce antete HTTP se pun, care este url-ul corespunzător.



## authentication-controller : Authentication Controller

Show/Hide | List Operations | Expand Operations

POST	/login	login
POST	/register	register

## devices-controller : Devices Controller

Show/Hide | List Operations | Expand Operations

DELETE	/device	delete
GET	/device	findByHash
POST	/device	add
PUT	/device	edit
GET	/device/all	getAll
PUT	/device/close	close
PUT	/device/connect	connect
PUT	/device/disconnect	disconnect
PUT	/device/open	open
POST	/device/setParams	setParams
PUT	/device/testConnection	testConnection

## user-controller : User Controller

Show/Hide | List Operations | Expand Operations

POST	/user/logout	logout
GET	/user/profile	profile
POST	/user/profile	changeProfile

Figura 10: Documentația interactivă a server-ului

## devices-controller : Devices Controller

Show/Hide | List Operations | Expand Operations

DELETE	/device	delete
--------	---------	--------

### Response Class (Status 200)

OK

Response Content Type

### Parameters

Parameter	Value	Description	Parameter Type	Data Type
Authorization	Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJ0dXJjdW5pY2tAZ2	Authorization	header	string
device	7018dea9-8cb6-4677-a5ea-cfb743ee4dca	device	query	string

### Response Messages

Try it out!

[Hide Response](#)

### Curl

```
curl -X DELETE --header 'Accept: application/json' --header 'Authorization: Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJ0dXJjdW5pY2tAZ2
```

### Request URL

```
http://localhost:9000/device?device=7018dea9-8cb6-4677-a5ea-cfb743ee4dca
```

Figura 11: Documentarea unei metode

Aceste informații sunt folosite la definirea API-ului deoarece o metodă poate să fie definită greșit. Drept urmare, atunci când se va folosi swagger, să se testeze dacă metoda merge, se va vedea ușor dacă metoda este greșit definită sau nu, sau dacă nu funcționează corespunzător.

Ngrok este un software ce permite dezvoltatorilor să expună pe internet un server web ce rulează pe mașina locală. Oferă o interfață web în timp real unde poate observa tot traficul HTTP care rulează peste tunelurile folosite. [12]

Motivul pentru care am folosit ngrok este pentru a valida cererile făcute de aplicația web, referitor la accesarea resurselor oferite de server. Am avut cazuri când cererea către server pe care o făceam era greșită, deși eu credeam că este corectă, și comunicarea între ei nu funcționa.

Figura 12 reprezintă o porțiune a unei capturi de ecran în interfața web oferită de ngrok. În partea stângă se pot vedea cererile ce s-au făcut serverului. Cererea „GET /device/all” este selectată, de aceea în partea dreaptă apare în format text conținutul ei.

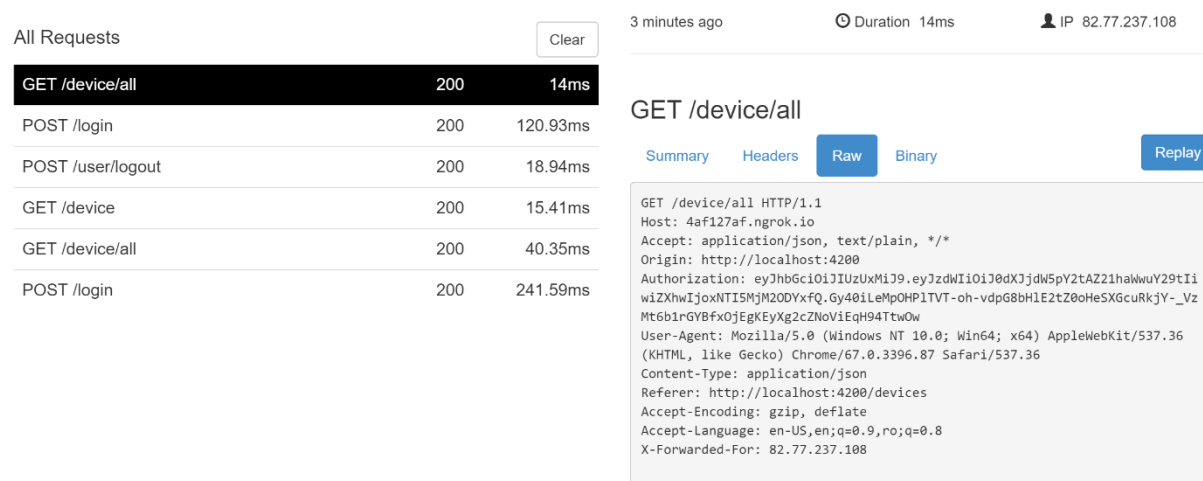


Figura 12: Observarea traficului HTTP dintre aplicația web și server

### 3.3. Modulul 3 – Dispozitivele inteligente

Din punct de vedere software dispozitivele inteligente trebuie să folosească un limbaj puternic, ce poate programa cât mai multe tipuri de clienți posibili. Din aceste considerente am ales Java, ce este un limbaj independent de platforma de lucru, portabil, simplu, robust.

Dispozitivele o să comunice cu serverul Java prin TCP, folosind protocolul criptografic SSL/TLS. Protocol SSL/TLS necesită un certificat SSL/TLS, de aceea am generat unul și l-am inclus în serverul Java și în aplicațiile dispozitivelor. Apoi, comunicarea efectivă între server și dispozitive este stabilită folosind un protocolul de comunicare definit în capitolul „3.2.3. Managerul de dispozitive și protocolul de comunicare”.

Pentru a simula dispozitivele inteligente am folosit un Raspberry Pi 2, Model B. El este de dimensiunile unui card de credit, ieftin, cu o varietate de sisteme de operare ce pot fi instalate. La Raspberry am conectat module arduino, acele module reprezentând efectiv dispozitivele inteligente.

#### 3.3.1. Interacțiunea cu Raspberry Pi

Primul lucru pe care a trebuit să îl fac, a fost instalare de sistem de operare. Am instalat sistemul de operare oficial pentru toate modelele de Raspberry, Raspbian. Acest lucru l-am realizat folosind un monitor alternativ ce l-am conectat la portul de HDMI (High-Definition Multimedia Interface). Dar, din cauză că monitor nu îmi aparținea, a trebuit să găsesc o modalitate de conectare direct din calculatorul personal. Așa că am conectat un cablu din portul de ethernet al calculatorului la portul de ethernet al raspberry-ului. Apoi am scanat rețeaua internă a laptopului, am găsit hostname-ul „raspberrypi” cu ip-ul corespunzător.

După, am configurat VNC<sup>30</sup>-ul, un sistem de împărtășire a desktop-ului către alt computer. Am făcut acest lucru pentru a putea avea o interfață grafică atunci când mă conectez la Raspberry. Apoi am deconectat monitorul și am realizat prima conexiune folosind prin VNC, știind ip-ul static. Raspberry-ul era conectat prin ethernet la calculator, dar nu aveam conexiune la internet din cadrul acestuia. Acest lucru nu m-a incomodat până nu a trebuit să scriu primul dispozitiv. Atunci mi-am dat seama că am nevoie de o librărie java, ce oferă suport în controlul pinilor de pe Raspberry. Dar nu puteam să configurez librăria deoarece nu aveam conexiune la internet.

---

<sup>30</sup> VNC: Virtual Network Computing

Din această cauză am configurat partajarea unei conexiuni wifi de la computer la Raspberry Pi folosind un cablu ethernet. Am reușit să fac această comunicare apelând la articolul [13]. După această configurare, Raspberry-ul își va lua un ip dinamic, de aceea înainte de conectare trebuie scanată rețeaua calculatorului între anumite clase de ip-uri. După, având acces la internet, am făcut update la sistemul de operare și am instalat librăria Java Pi4J.

Pentru a automatiza cât de mult posibil compilarea și rularea dispozitivelor am definit makefile-uri pentru fiecare dintre ele. Apoi am scris un script care ce folosește aceste makefile-uri pentru a porni fiecare dispozitiv în câte un terminal, în paralel.

```
compile:

    javac -classpath .:classes:/opt/pi4j/lib/'*' -d . Lamp.java

run:

    java -classpath .:classes:/opt/pi4j/lib/'*' Lamp lamp $(ip) 8000 0 1 2
```

Tabelul 7: Makefile-ul dispozitivului lampă

Tabelul 7 reprezintă makefile-ul dispozitivului lampă. El are două părți: compilare și rulare. Din cauză că folosesc librăria Pi4J trebuie să adaug în classpath, la rulare și la compilare, JAR-urile librăriei Pi4J.

```
#!/bin/bash

lxterm -e 'bash -c "cd DoorLock/src && make compile; make ip=$(hostname -I) run"' &
lxterm -e 'bash -c "cd Lamp/src && make compile; make ip=$(hostname -I) run"' &
lxterm -e 'bash -c "cd SecurityLaser/src && make compile; make ip=$(hostname -I) run"' &
lxterm -e 'bash -c "cd LightBulb/src && make compile; make ip=$(hostname -I) run"'
```

Tabelul 8: Scriptul ce pornește toate dispozitivele simultan

În Tabelul 8 se găsește conținutul scriptului ce pornește toate dispozitivele simultan. În urma rulării acestui script se vor deschide în paralel 4 terminale ce vor compila codul corespunzător dispozitivului, apoi vor pornește dispozitivele.

### 3.3.2. Dispozitive implementate

În contruirea dispozitivelor, partea cea mai importantă este definirea parametrilor personalizați. Parametrii unui dispozitiv nu vor fi cunoscuți apriori de către serverul Java. Vor fi aflați abia după ce se realizează conexiunea cu dispozitivul. Ei trebuie definiți cu atenție pentru fiecare dispozitiv, fiind sugestivi în denumire și îndeplinind o operație clară.

Am ales această idee de design deoarece protocolul de comunicare devine astfel mai dinamic, numele și numărul parametrilor unui dispozitiv fiind cunoscut la momentul execuției programului. Un alt motiv, mai practic, constă în faptul că dispozitivele nu sunt neapărat contruite de aceeași producător și în plus implementările nu o să fie livrate toate odată. Unele o să fi gata mai devreme, altele mai târziu. Dacă în logica serverului s-ar ține cont de tipul de dispozitiv și de parametrii lui, atunci ar fi prea mare cuplajul între server și aplicațiile client, și la orice dispozitiv nou ar trebui modificat și recompilat serverul.

Pentru dispozitivele implementate am folosit module arduino la partea de hardware și librăria Java Pi4J pe partea de software. Această librărie pune la dispoziție o schemă ce ilustrează tipul pinilor folosind schema de numerotare<sup>31</sup> GPIO Pi4J/WiringPi.

Ip-urile la care vor rula dispozitivele vor fi aceleași ca și ip-ul Raspberry-ului. În continuare voi descrie ce dispozitive am implementat.

#### 3.3.2.1. Laser de securitate



Figura 13: KY-008 Modulul senzor laser<sup>32</sup>

Dispozitivul hardware folosit este modulul senzor laser, din Figura 13. Pentru controlul acestui dispozitiv se folosește un singur pin GPIO (General-Purpose Input/Output), mai precis GPIO 4. Portul la care va fi găsit dispozitivul este 8002. Dispozitivul are un parametru cu numele *laser* de tip boolean. Valoarea *true* înseamnă că laserul este pornit, valoarea *false* însemnând că laserul este oprit.

<sup>31</sup> Schema de numerotare: <http://pi4j.com/images/j8header-2b-large.png>

<sup>32</sup> Imagine preluată. Sursa: <https://www.fasttech.com/product/1219301-keyes-ky-008-arduino-compatible-650nm-laser>

Am ales să simulez un sistem de securitate prin intermediul modulului senzor laser. Am făcut această alegere pentru că securitatea locuinței este foarte importantă. Prin intermediul unei locuințe inteligente nivelul de securitate poate atinge fără prea mari dificultăți nivelul protecției unei clădiri guvernamentale top secrete.

DETALIERE – Imagine aplicația web

### 3.3.2.2. Lampă



Figura 14: KY-016 Modul LED cu 3 culori<sup>33</sup>

Acest dispozitiv folosește modulul LED cu 3 culori senzor laser, din Figura 14. Pentru că este un led cu trei culori, acest dispozitiv necesită folosirea a trei pini GPIO. Pini aleși sunt: GPIO 0, GPIO 1, GPIO 2. Acest dispozitiv va folosi portul 8000 în comunicarea cu serverul. Parametrii dispozitivului sunt: *red*, *green*, *blue*. Valorile sunt de tip boolean, *true* înseamnă ca acea culoare este setată, *false* reprezentând opusul. Parametrii nu se setează individual, având astfel posibilitatea de a fi setați în grupuri. Pe scurt, culorile ce pot fi setate sunt: roșu, galben, verde, turcoaz, albastru, violet, alb.

Acest dispozitiv simulează o lampă reală, ce oferă o paletă de 7 culori. Am ales acest dispozitiv deoarece, contrar funcționalității simple, este foarte util în viața de zi cu zi. Pe lângă asta, posibilitatea de a seta mai multe culori unei lampe este deasemenea un element important în crearea unei ambianțe plăcute.

DETALIERE – Imagine aplicația web

### 3.3.2.3. Încuietoarea ușii

---

<sup>33</sup> Imagine preluată. Sursa: <https://piandmore.wordpress.com/2016/02/10/nodemcu-ky-016-3-color-led>



Figura 15: KY-019 Modul releu 5V<sup>34</sup>

Dispozitivul folosește modulul releu de 5V, din Figura 15. Modulul este controlat folosind un singur pin GPIO, mai exact GPIO 3. Dispozitivul folosește portul 8001 în comunicarea cu serverul. Ca parametrii necesită un singur parametru cu numele *lock*, de tip boolean. Dacă valoare este *true* atunci ușa este încuiată, *false* opusul.

Dispozitivul simulează o încuietoare reală a ușii. Am ales acest dispozitiv întrucât încuietoarea ușii este folosită zilnic. Controlul de la distanță al acesteia ne scutește de momentele de incertitudine în care ne întrebăm dacă am încuiat sau nu ușa.

DETALIERE – Imagine aplicația web

### 3.3.2.4. Bec



Figura 16: KY-011 Modul LED cu 2 culori<sup>35</sup>

Dispozitivul folosește modulul LED cu 2 culori, din Figura 16. Modulul este controlat folosind doi pini GPIO, mai precis GPIO 5 și GPIO 6. Dispozitivul folosește portul 8003 în comunicarea cu serverul. Necesită doi parametrii cu numele *red*, respectiv *green*. Atunci când parametrul are valoare *true* culoarea este setată, *false* reprezentând opusul.

<sup>34</sup> Imagine preluată. Sursa: [https://tkkrlab.nl/wiki/Arduino\\_KY-019\\_5V\\_relay\\_module](https://tkkrlab.nl/wiki/Arduino_KY-019_5V_relay_module)

<sup>35</sup> Imagine preluată. Sursa: <https://www.newegg.com/Product/Product.aspx?Item=1FS-002B-000B7>

## 4. Instalarea aplicație

### 4.1. Aplicația web

Următorii pași trebuie efectuați pentru instalarea aplicației web:

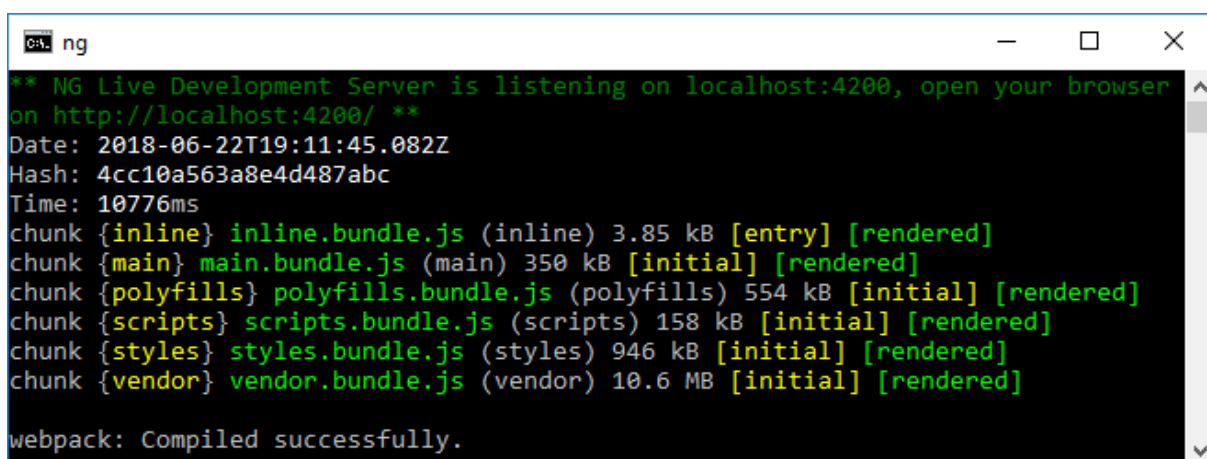
- instalare Node.js și npm<sup>36</sup>, dacă nu sunt deja pe mașină;
- instalare Angular CLI. Se deschide linia de comandă și se execută comanda „npm install -g @angular/cli”;
- în directorul proiectului, se deschide o linie de comandă și se execută comanda „npm install”.

Aplicația web poate fi acum rulată, dar mai întâi trebuie să instalăm niște dependențe. Ele se găsesc enumerate în fișierul README.md, afla în rădăcina proiectului, dar le găsiți și mai jos:

- npm install --save bootstrap@next
- npm install ngx-toastr --save
- npm install @angular/animations --save
- npm install --save highcharts && npm install --save-dev @types/highchartst

Ordinea în care se execută comenzile nu contează.

După ce acești pași au fost efectuați, aplicația web este gata pentru a fi rulată. Comanda „ng serve”, executată într-o linie de comandă deschisă din directorul proiectului v-a rula aplicația. La linia de comandă ar trebui să apară ceva asemănător ca în Figura 17.



```
ng
** NG Live Development Server is listening on localhost:4200, open your browser
on http://localhost:4200/ **
Date: 2018-06-22T19:11:45.082Z
Hash: 4cc10a563a8e4d487abc
Time: 10776ms
chunk {inline} inline.bundle.js (inline) 3.85 kB [entry] [rendered]
chunk {main} main.bundle.js (main) 350 kB [initial] [rendered]
chunk {polyfills} polyfills.bundle.js (polyfills) 554 kB [initial] [rendered]
chunk {scripts} scripts.bundle.js (scripts) 158 kB [initial] [rendered]
chunk {styles} styles.bundle.js (styles) 946 kB [initial] [rendered]
chunk {vendor} vendor.bundle.js (vendor) 10.6 MB [initial] [rendered]
webpack: Compiled successfully.
```

Figura 17: Deschiderea aplicației web

<sup>36</sup> Site-ul pentru descărcare: <https://nodejs.org/en/download/>



Aceasta înseamnă că serverul este pornit, și, după cum scrie și la linia de comandă, se găsește la adresa <http://localhost:4200>.

Dacă doriți să schimbați portul la care rulează aplicația, de exemplu din 4200 în 6000, aveți două opțiuni:

- editarea fișierului „packages.json”, navigând la cheia „start”, iar după, în interiorul ei, modificând la cheia start din: „ng serve” în „ng serve --port 6000”
- editarea fișierului „angular-cli.json” definind o proprietate ca în Tabelul 9.

```
{
  "defaults": {
    "serve": {
      "port": 6000
    }
  }
}
```

Tabelul 9: Definirea portului în fișierul .angular-cli.json

## 4.2. Serverul Rest Java

Pentru configurarea serverului Java trebuie urmați următorii pași:

- instalare java<sup>37</sup>, în caz că nu există pe mașină;
- instalare maven<sup>38</sup>;
- deschiderea unei linii de comandă, în care se va executa comanda „mvn -version”. Rolul execuției acestei comenzi este verificarea instalării corectă a maven-ului.
- în aceeași linie de comandă, trebuie rulată și comanda „mvn clean install -U”.

Aceasta instalează toate dependențele proiectului.

Serverul este instalat. Acum, pentru deschiderea lui trebuie navigat în directorul proiectului, deschisă o linie de comandă și rularea comenzii „mvn spring-boot:run”. În urma rulării comenzii, în conținutul liniei de comandă v-a fi ca cel din Figura 18.

Serverul se va porni la portul 9000. Dacă doriți să schimbați această configurare, trebuie să navigați în directorul „resources”, fișierul „application.properties”. În acest fișier, pe prima linie se specifică portul serverului.

<sup>37</sup> <http://www.oracle.com/technetwork/java/javase/downloads/index.html>

<sup>38</sup> <https://maven.apache.org/download.cgi#Installation>



# Concluziile lucrării și direcții de dezvoltare

## Concluziile lucrării

## Direcții de dezvoltare

Consider că acest sistem poate deveni mult mai complex.

Se poate extinde gama de dispozitive implementate, în funcție de necesitate. O dată cu creșterea numărului de dispozitive protocolul de comunicare va trebui să fie updatat, eventual să suporte mai multe comenzi.

De asemenea, se pot crea de grupuri, reprezentând membrii unei familii ce pot împărtăși aceleași dispozitive din casă. Accesul la dispozitive trebuie să se realizeze concurent, iar pentru operațiile atomice să se folosească mecanisme de sincronizare.

În opinia mea, această sistem reprezintă un start bun, având o implementare arată multe aspecte ale unui sistem de tip Smart Home.

# Bibliografie

- [1] D. J. Cook, „How Smart Is Your Home?,” *Science (New York, NY)*, vol. 335, nr. 6076, pp. 1579-1581, 2012.
- [2] \*\*\*, „What is a Smart Home,” n.d. n.d. c2018. [Interactiv]. Available: <https://www.smarthomeusa.com/smarthome/>. [Accesat 3 06 2018].
- [3] Homeoftech, „7 Benefits of living in a smart home,” 24 05 2016. [Interactiv]. Available: <http://homeoftechnologies.co.uk/7-benefits-of-living-in-a-smart-home/>. [Accesat 03 06 2018].
- [4] N. C. Molly Edmonds, „How Smart Homes Work,” 25 03 2008. [Interactiv]. Available: <https://home.howstuffworks.com/smart-home6.htm>. [Accesat 03 07 2018].
- [5] B. Sabin-Corneliu, „Dezvoltarea de aplicații Web prin REST,” [Interactiv]. Available: <https://profs.info.uaic.ro/~busaco/teach/courses/web/presentations/web11ServiciiWeb-REST.pdf>. [Accesat 13 06 2018].
- [6] D. S. Robert Savage, „The Pi4J Project,” [Interactiv]. Available: <http://pi4j.com/index.html>.
- [7] S. Cooper, „ngx-toastr,” [Interactiv]. Available: <https://github.com/scttcper/ngx-toastr>. [Accesat 04 05 2018].
- [8] „Highcharts,” [Interactiv]. Available: <https://www.highcharts.com/>.
- [9] B. Pollard, „Secure websites with HTTPS,” 25 02 2018. [Interactiv]. Available: <https://www.tunetheweb.com/security/https/>.
- [10] \*\*, „The importance of SSL - Top 6 advantages,” Clear Vertical Ltd. 8259274, [Interactiv]. Available: <https://www.clearvertical.co.uk/the-importance-and-advantages-of-ssl/>.
- [11] SmartBear Software, „Swagger,” [Interactiv]. Available: <https://swagger.io/>.
- [12] inconshreveable, „ngrok,” [Interactiv]. Available: <https://ngrok.com/>.
- [13] S. Anwaarullah, „Sharing WiFi with Raspberry Pi using a LAN Cable,” [Interactiv]. Available: <https://www.hackster.io/Anwaarullah/sharing-wifi-with-raspberry-pi-using-a-lan-cable-ae1f44>.