Abstract

Lista figurilor

Lista tabelelor

Cuprins

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 comportamenului 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[1], și-au imaginat un viitor în care casele vor fi interactive. În scurta povestire a lui Bradbury, „There Will Come Soft Rain"[2], 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.

[1] Ray Douglas Bradbury (n. 22 August 1920 – d. 5 Iunie 2012): scriitor de romane stiintifico-fantastice, fantezie, horror și mister.

[2] „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[3] 2, Model B, iar dispozitivele controlate de utilizator vor fi module arduino[4].

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[5] care are drept scop interacțiunea eficientă și facilă cu utilizatorul; partea de back-end[6] 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 managementul 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” [7] 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.

[3] Raspberry Pi: https://www.raspberrypi.org/
[4] Modul arduino: este un circuit hardware complex, adus la o formă compactă pentru utilizare, cu o interfață de conexiune simplistă, ce necesită doar atasarea unor fire între placă și circuit.
[5] Front-end: Parte a site-ului pe care o putem vedea și cu care interacționează vizitatorii.
[6] Back-end: Locul unde se administrează informația. De obicei este format din server și bază de date.
[7] Smart Home: Echivalentul acestui termen în limba romană 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 sa 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 speriindu-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 apasarea unei taste) distanță. Fie că este vorba de telefonul tău inteligent, tabletă, calculator personal sau un terminal monat în perete**.**

Când te pregătești să dormi, puteți opri orice lumină din casă doar prin apăsarea unui buton. Puteți să vă uitați la ecranul dumneavoastă 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. Dimineața 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 rotile 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[8] 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,

[8] Alzheimer: tip de demență care cauzează probleme cu memoria, gândirea și comportamentul.

dacă nu aveti 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[9]. 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, acestă 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]

[9] Gadget: obiect tehnologic mic care îndeplinește o anumită funcție, de obicei fiind ceva nou.

Contribuții

1. Arhitectura sistemului

1.1. Proiectare

În ceea ce urmează o să descriu planul abordat de către mine pentru a pune în practică ideea de smart home.

Sistemul trebuie să fie accesibil din orice locație din lume și din această cauză am ales ca interfața cu utilizatorul 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 alese.

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

- un mediu persistent de stocare a datelor;
- înregistrarea, autentificarea[10] și managementul utilizatorilor;
- înregistrarea și controlul dispozitivelor expuse de utilizator;
- managementul conecțiunilor cu dispozitivele: 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 de programare trebuie folosit pentru a programa cât mai multe tipuri de clienți, daca nu chiar toți clienții posibili? 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[11], portabilitate.

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

[10] Autentificarea: Are drept scop stabilirea identității actorilor care doresc să comunice sigur în rețea.
[11] 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”[12]. O să descriu în cotinuare 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 un astfel de problemă care dovedește sistemul a fi inutilizabil, am să folosesc protocolul criptografic SSL[13] între server și client. SSL este un protocol critpografic care poate asigura confidențialitate, integritatea mesajelor și autentificarea părților. SSL acționează asupra unui flux TCP[14] și oferă servicii nivelurilor superioare.

În Figura 1 de mai jos se poate observa o schiță a design-ului sistemului.

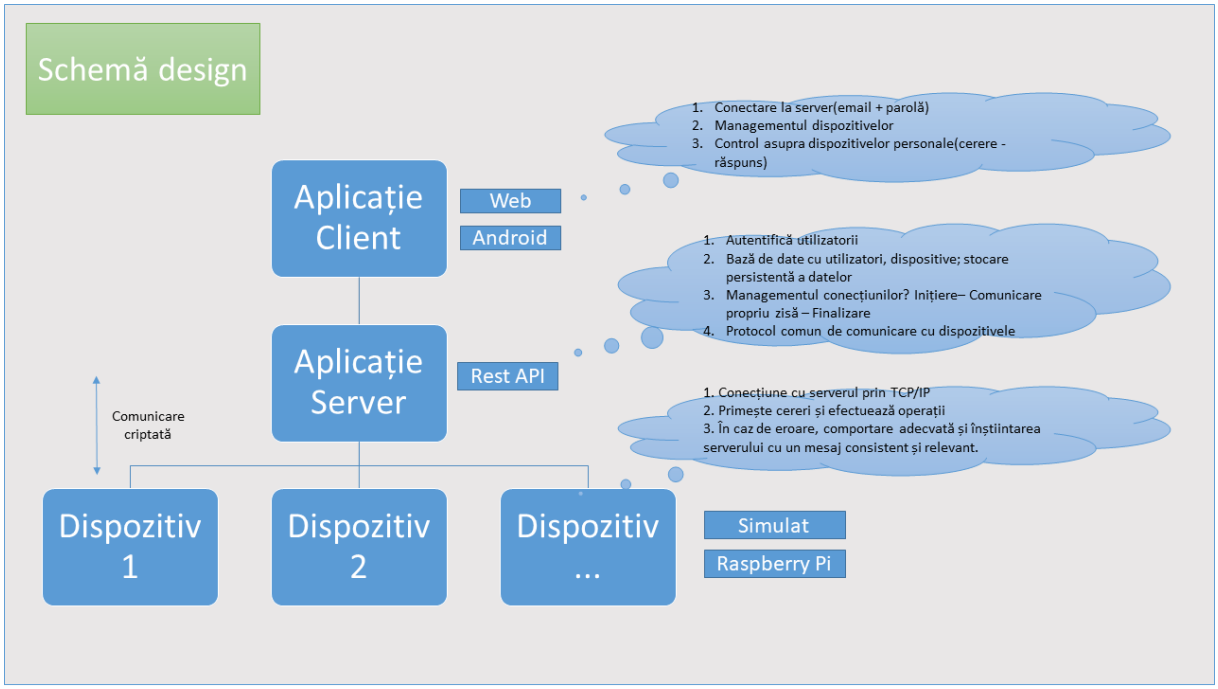

Figura 1: Schemă planificare design

[12] Man-in-the-middle: detalii aici: https://arxiv.org/ftp/arxiv/papers/1504/1504.02115.pdf
[13] SSL: Secure Sockets Layer
[14] TCP: Transport Control Protocol

1.2. Detalii arhitecturale

Rezulatul proiectării a dus la 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 intracționează cu sistemul prin intermediul primului modul, adică aplicația web. Aplicația structurează informațiile necesare utilizatorului astfel încât interacțiunea cu dispozivele personale să fie cât mai simplă și facilă.

Modulul al doilea este responsabil în oferirea de raspunsuri pentru toate cererile ce vor veni de la utilizator, prin intermediul aplicației web. Paradigma de programare a serverului este REST[15]. 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[16] 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] Motivul pentru care am ales acestă paradigma de programare deoarece oferă o separare între client și server, vizibilitate, scalabilitate, api-ul[17] REST este 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, așa 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. Funcționalitatea pinilor de ieșire și modul lor de conexiune este explicată de producător. Am ales această organizare deoarece în primul rând este ieftin din punct de vedere al prețului; module arduino m-au ajutat să am o interacțiune minimală cu dispozitivele hardware; iar există o multitudine de sisteme de operare ce pot fi instalte pe un Raspberry Pi. Pentru programarea dispozitivelor arduino am folosit o librarie pentru controlarea fluxului de intrare ieșire a unui Raspberry Pi, Pi4J [6].

[15] REST: Representational state transfer
[16] MIME: Multipurpose Internet Mail Extensions
[17] 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 de mai sus. Comunicarea între niveluri se realizează asfel:

- aplicația client comunică cu aplicația server via verbe HTTP[18], schimbând resurse în format MIME.
- aplicația server comunică cu dispozitivele conectate la Raspberry Pi printr-un protocol definit peste TCP, ce urmărește patternul cerere-răspuns.

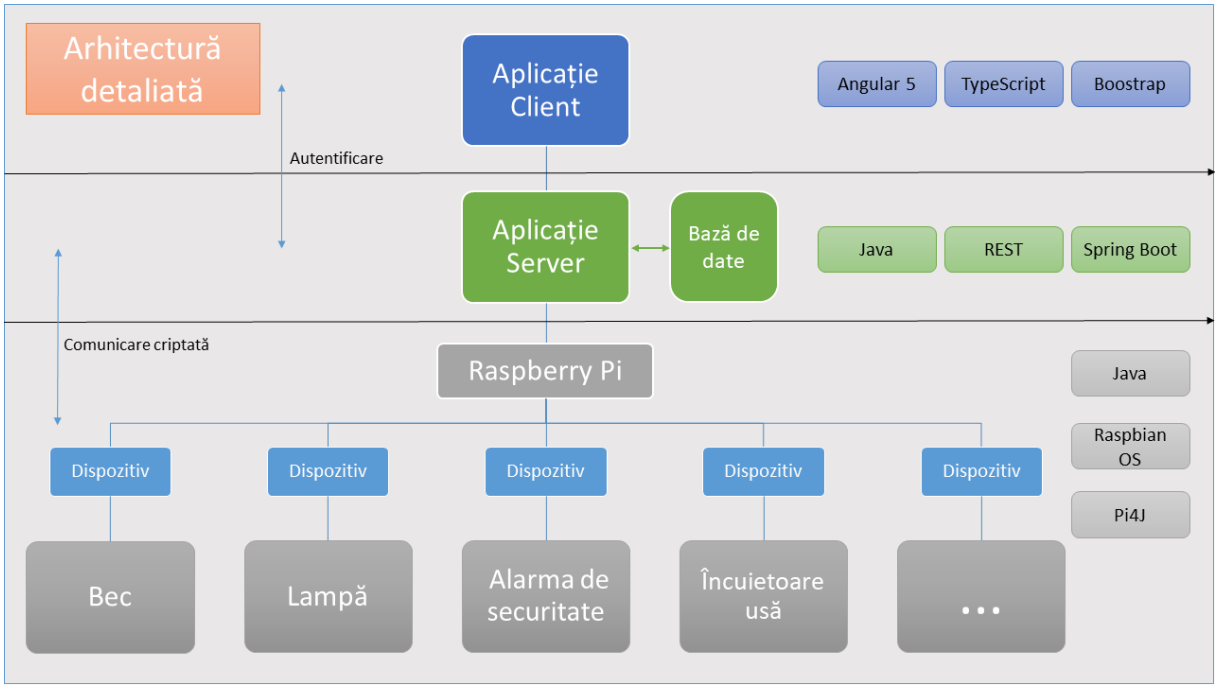

Figura 2: Arhitectura detaliată a sistemului

Protocolul de comunicare urmărește satisfacerea următoarelor funcționalități:

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

Dispozitivele vor avea definite, de la contrucție, parametrii personalizați. Parametrii unui dispozitiv nu vor fi cunoscuți apriori de către serverul Java. După ce se face conexiunea

[18] HTTP: Hyper Text Transfer Protocol

cu dispozitivul se vor afla parametrii interogare aplicată dispozitivului. Acest lucru face protocolul de comunicare mai dinamic, o eventuală schimbare în parametrii nu va afecta în niciun fel comunicare dintre server și client. Este drept că aplicația web va fi afectată, deoarece acolo este necesară cunoșterea exactă a numelui parametrului ce este primit.

1.3. Exemplificare scenarii, cazuri de utilizare și fluxuri de activități

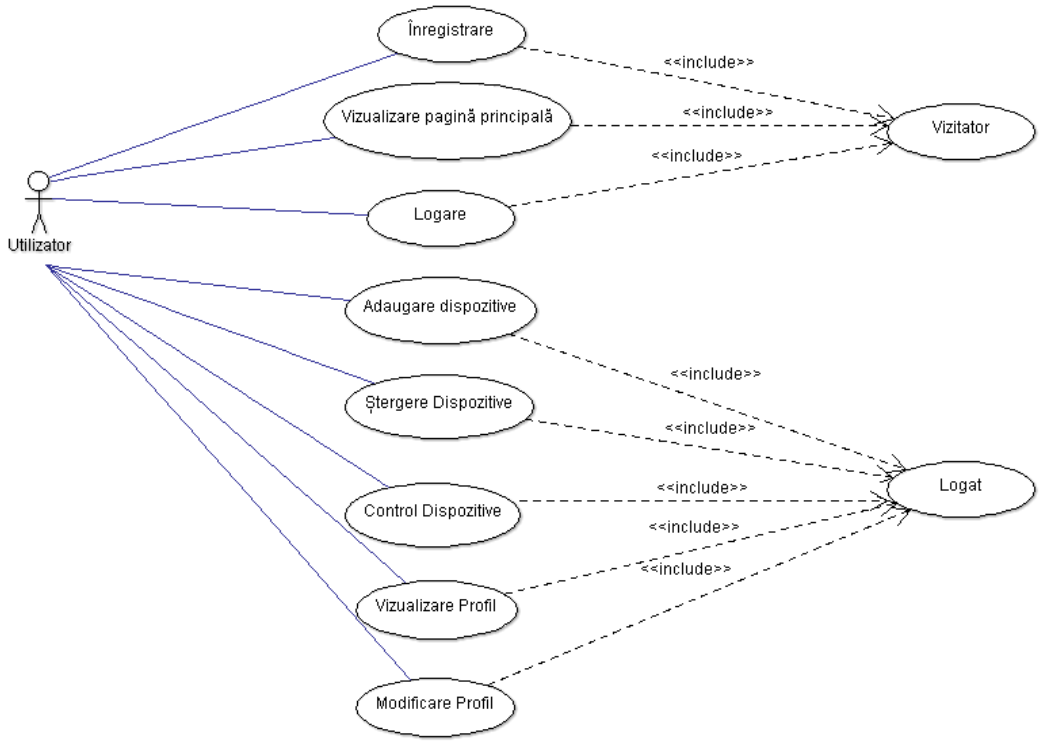

Figura 3: Interacțiunea utilizatorului cu sistemul

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

Utilizatorul logat poate să adauge, șteargă și să programeze dispozitivele. În plus poate să își vizualizeze profilul personal și să îl modifice după bunul plac.

În Figura 4, localizată mai jos, reprezintă o diagramă de activitate. Această diagramă intenționează să arate fluxul de activitate rezultat din interacțiunea utilizatorului cu sistemul. Fluxul diagramei este de sus în jos. De exemplu conectarea la dispozitiv se află pozitionată la baza diagramei deoarece o secvență de pași trebuiesc efectuați până la posibilitatea conectării

la dispozitiv cum ar fi: logarea utilizatorului, vizualizarea listei de dispozitive, alegerea unui dispozitiv din listă, selectarea opțiunii de management al dispozitivului, iar apoi conectarea efectivă la dispozitiv.

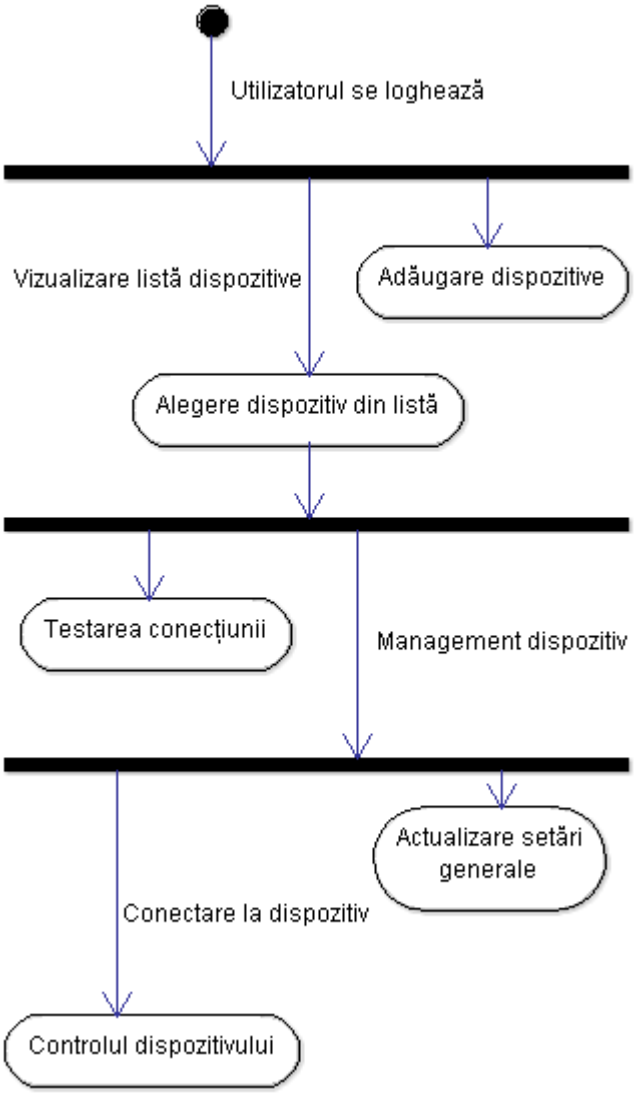

Figura 4: Diagrama de activitate interacțiunii utilizatorului cu sistemul

Un scenariu important este managementul conexiunilor cu dispozitivele. Conectarea la dispozitive o face utilizatorul manual, prin apasarea butonului „Connect” de pe interfața grafică. Pănă acum nu este nici o problemă. La deconectare deja încep să apară unele întrebări. Dacă utilizatorul apasă pe butonul de deconectarea înainte să închidă pagina sau să se delogheze, este foarte bine, s-a asigurat el că dispozitivul este închis. Dar dacă se conectează la dispozitiv și apoi închide tab-ul din browser? Sau se deloghează? Sau chiar închide browser-ul. Aceste cazuri trebuie tratate corespunzător pentru a nu consuma resurse inutil sau defecta dispozitivele. În primul rând o modalitatea de a trata anumite cazuri este folosind interceptarea de evenimente cum ar fi închiderea tabului, delogarea, închiderea browser-ului.

Cum unele evenimente sunt dependente de mediul de lucru, de exemplu browser-ul folosit de client în accesarea site-ul web, o soluție independentă de mediu de lucru ar fi folosirea unui mecanism de expirare a unei unitați timp prestabilite. Un astfel de mecanism va acționa pe partea de server, pentru a avea control total. Mecanismul va funcționa în următorul mod: Când un utilizator va face o se ca conecta la un dispozitiv sau v-a face o stare dispozitivului, se va porni un timer. Dacă timer-ul este deja pronit și utilizatorul face o acțiune, atunci timer-ul va începe număratoarea de la început. Timer-ul va avea o valoare de început prestabilită, de exemplu: 20 de minute. Prin urmare, comportamenul în urma aplicării acestui mecanism va fi următorul: 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, neafectând în nici un fel starea pe care o avea înainte de deconectare. Astfel s-a ajuns la un comportament sigur, independent de browser, ce asigură eliberarea corespunzătoare de resurse.

2. Implementare

2.1. Modulul 1 – Aplicația web

Modulul de aplicație web este foarte important, deoarece utilizatorul va interacționa cu sistemul folosind interfața web. Pentru a crea confort utilizatorului atunci când interacționează cu sistemul mi-am stabilit următoarele obiective de design:

- *intuitiv*. Atunci când utilizatorul folosește aplicația să nu fie nevoit să citească pagini întregi cu manualul aplicației pentru a ști cum se efectuează o operație simplă.
- *modern*. Folosirea de tehnologii de ultimă generație oferă o experiență mai bună utilizatorului, noile tehnologii fiind 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 paginile pot fi prea încărcate, operațiunile ce se doresc a fi efectuate pot fi complicat de configurat.
- *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 hotarât să folosesc pentru acest modul două dintre cele mai populare tehnologii web: Angular[19] și Boostrap[20].

Am ales Angular pentru că este o bibliotecă ce ajută la crearea aplicațiilor SPA[21]. Datorită acestui fapt, pagina este fragmentată în mai multe componente ce se încarcă dinamic atunci cand utilizatorul interacționează cu aplicația. Această abordare bazată pe componente face ca aplicația să se încarce foarte rapid, să fie modularizată, evitând duplicarea de cod.

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

[19] Angular: https://cli.angular.io/
[20] Boostrap: https://getbootstrap.com/
[21] Single-page application: Aplicație ce interacționează cu utilizatorul rescriind dinamic pagina curentă, în loc să încarce pagini întregi primite de la server.

2.1.1. Structura proiectului

În Figura 5 se poate observa structura unei componente Angular. O componentă conține un fișier .css[22], unul .html[23] și altul .ts[24]. Acest mod de structurare al unei componente îi permite să nu depindă de altă componentă, astfel reducând cuplajul din interiorul aplicației.

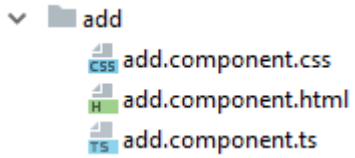

Figura 5: Structura unei componente angular

Prin definiție, cuplajul este o masură a gradului de dependență a unei clase de alte clase. Reformulând această definiție pentru contextul curent, cuplajul este gradul de dependență a unei componente de altă componentă. O componentă are un cuplaj redus daca nu depinde de multe alte componente.

În Figura 6 se poate observa structura aplicației web, ce conține urmatoarele 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, încuietoarea 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 logarea utilizatorului.
- *notfound*: componentă ce va apare de fiecare dată când utilizatorul va încerca să acceseze o resursă ce nu este disponibilă.
- *profile*: componentă ce se ocupă cu managentul profilului utilizatorului.
- *register*: componenta cu care interacționează utilizatorul când se înregistrează.

[22] Cascade Style Sheets(foaie de stiluri CSS): standard pentru formatarea elementelor unui document html.
[23] Hyper Text Markup Language: limbajul principal al Web-ului pentru crearea de conținut ce poate fi utilizat oriunde.
[24] TypeScript: limbaj puternic tipizat, orientat obiect, compilat, superset al lui JavaScript.

2.1.2. Rutarea

Pentru a explica ce este rutarea, mai întăi trebuie sa explic ce este un view. O grupare de elemente prezentate utilizatorului ce se creează și distrug împreună formează un view.

Rutarea constă în navigarea de pe un view pe următorul view, pe masură ce utilizatorii execută operații în aplicație.

Figura 7 conține codul corespunzător care efectuează rutarea. Se exportă un router[25] 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[26]-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 daca se poate sau nu accesa ruta.

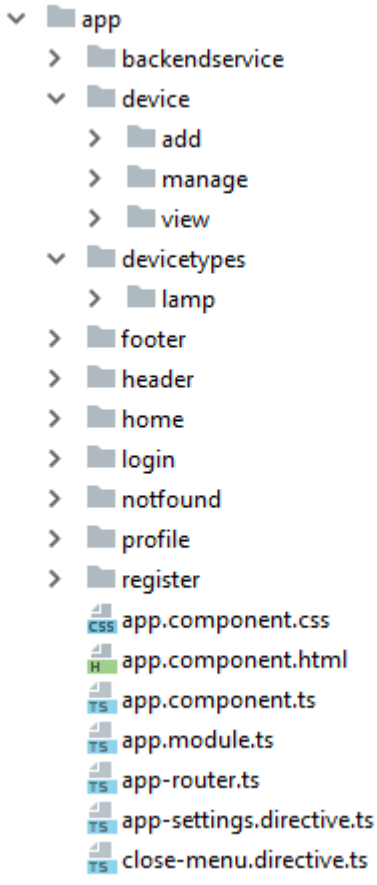

Figura 6: Structura aplicației web

[25] Obiect responsabil cu navigarea de pe un view pe următorul view.
[26] 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);
```

Figura 7: 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 de NotFound, indicându-i, într-o manieră elegantă, utilizatorului că a introdus un link greșit. „devices/manage/:id” să capteze 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 hostul local, la portul 4200, și utilizatorul introduce în browser următorul url: *http://localhost:4200/home*, atunci, conform ruter-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, dar AuthGuard va decide dacă se poate sau nu accesa. O să explic în următoarea secțiune ce este AuthGuard și care este rolul lui.

2.1.3. Serviciul de logare

Într-o aplicație web, există rute publice și rute ce pot fi accesate dacă se îndeplinesc anumite condiții. Una dintre aceste condiții este starea de logare a utilizatorului. Acesta este și cazul de față, AuthGuard fiind un serviciu care verifică starea de logarea a utilizatorului când accesează anumite rute.

Paginile publice din această aplicație sunt: pagina principală, pagina de login, pagina de register. Paginile verificate de serviciu sunt: pagina de profil și paginile ce țin de dispozitive. Este necesară verificarea stării de logare, deoarece utilizatorul vizitator (ne-logat) nu are acces la principalele funcționalități oferite de aplicație, pentru aceasta fiind nevoie de un profil înregistrat, însoțit de logarea propriu-zisă.

2.1.4. Librării folosite

În creearea aplicației am folosit următoarele librării:

- ngx-toastr [7]
- highcharts [8]
- fontawesome: [9]

Ngx-toatr este o librărie ce am folosit-o pentru a afișa mesaje toast utilizatorilor. Un mesaj toast este un mesaj scurt într-un popup. Toast-ul dispare dacă se apasă pe el sau automat, după un anumit timp. 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 reușeste din diverite motive tehnice.

HighCharts oferă suport pentru grafice scrise în javascript. Am integrat un grafic în design-ul interfeței obiectului lampă, mai precis la 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 a fi cea mai aproape de realitate, în ceea ce privește interacțiunea cu o lampă.

FontAwesome l-am folosit pentru diverse pictograme pe care le oferă. DETALIERE

2.2. 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[27], un framework de Java foarte popular în construirea de aplicatii web și enterprise. Nu este prima dată când interacționez cu acest framework, rămânând cu un gust plăcut după fiecare proiect scris utilizând acest framework.

Paradigma de programare a serverului este REST[28]. 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[29] 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, șamd. [5] Motivul pentru care am ales acestă paradigma de programare deoarece oferă o separare între client și server, vizibilitate, scalabilitate, api-ul REST este mereu independent de tipul de platformă sau limbajul de programare folosit.

2.2.1. Structura proiectului

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

- *controllers*: o colecție de controllere sunt responsabile cu preluarea cererilor de la client gestionănd resursele necesare satisfacerii cererilor.
- *dtos*: acronim pentru „Data Transfer Object”, folosite pentru a incapsula 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 intependent.
- *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 în accesul la date.

[27] Spring Boot: https://spring.io/projects/spring-boot
[28] REST: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
[29] MIME: https://www.iana.org/assignments/media-types/media-types.xhtml

- *security*: pachet responsabil cu 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 cu utilizatorii.
- SwaggerConfig: clasă responsabilă cu configurarea Swagger-ului[30].
- Application.properties: fișier responsabil cu configurări ale aplicație. Aici se găsesc configurări cum ar fi: portul la care rulează serverul, șirul de caractere ce reprezintă conexiunea la baza de date, numele utilizatorului ce poate accesa baza de date, ș.a.m.d.

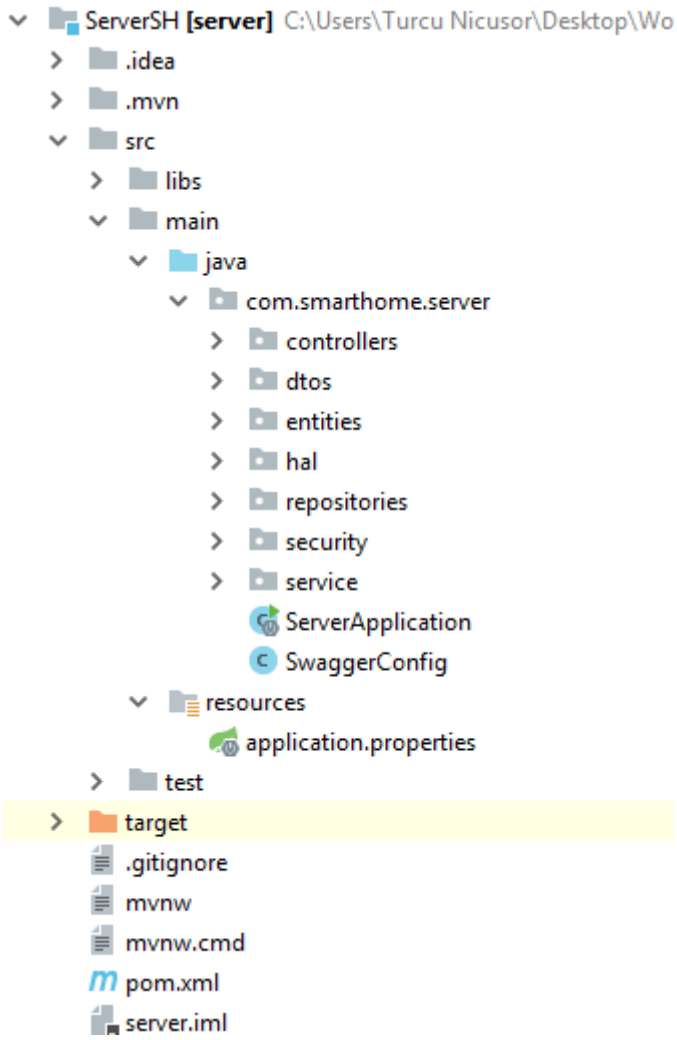

Figura 8: Structura serverului Java

[30] Swagger: https://swagger.io

Fișierul pom.xml conține dependențele din cadrul server-ului. Ele sunt specificate în format xml, după cum se vede și în Figura 9. Atunci când se specifică dependențele, se caută 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>
```

Figura 9: Dependința la baza de date, definită pom.xml

2.2.2. 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 este organizarea unei colecții de date, indiferent de tipul lor. Mai teoretic, 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 incapsulează 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: angajat, student, profesor. 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. Permite utilizarea obiectele, fără a avea nevoie să știi cum se face managementul lor în spate. Două dintre avantajele folosirii sunt: existența un singur loc unde se fac modificări în accesul datelor precum și existența unui singur loc responsabil pentru mai multe tabele.

Un controller face de obicei, doar un lucru: primește o intrare și generează rezultat. Mai detaliat, un controller primește cererile utilizatorului, apoi deleagă de obicei un serviciu în a rezolva cererea, iar cand cererea se rezolvă, el răspunde cu mesajul corespunzător.

Acum că am terminat definirea de termeni, voi continua explicarea contrucției serverului.

Atunci când un utilizator efectuează o operație pe partea de aplicație web, atunci un cerere va fi trimisă serverului. Prima componentă întâlnită va fi filtrul http. Filtrul http este descris în subcapitolul 3.2.4 Securitate. Apoi cererea trece de la filtrele http la controller-ul corespunzător. În aplicație există trei controllere: pentru dispositive, utilizatori, și pentru autentificarea utilizatorilor. Fiecare controller are o responsabilitate exactă.

Figura 10, de mai jos, conține procedura responsabilă de conectarea la un dispozitiv. „@PutMapping" este o adnotare ce are următoarea semnificație: metoda se apelează atunci 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 identificare 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.

```
@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("");
    }
```

Figura 10: Procedura responsabilă de conectarea la un dispozitiv

Metoda verifică mai întâi ca dispozitivul să existe și ca utilizatorul să aibe drept să îl acceseze, iar apoi se delegă resposabilitatea managerului de dispozitive ce va efectua operația de conectare. Metoda checkDevice nu este inclusă in blocul try catch pentru că face o verificare a condițiilor minime de continuare 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.

Repository-urile le-am folosit în 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[31], toate setările de conectare fiind definite în fișierul „application.properties”.

Un exemplu de dto folosite este dto-ul de înregistrare. El are următorii membrii privați: *firstName, lastName, email, password*. Fiecare membru are validare corespunzătoare, de exemplu, la câmpul de email se verifică formatul email-ului să fie valid. Acest obiect se completează după ce utilizatorul completează formularul de înregistrare, apoi se trimite serverului care procesează corespunzător cererea.

2.2.3. 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 și 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 loghează, toate dispozitivele lui, existente în baza de date, se vor încărca într-un HashMap intern. Cache-ul este necesar deoarece o conecțiune trebuie inițiată cu dispozitivul atunci când clientul o solicită. Această stare a conecțiunii trebuie menținută corespunzător.

După ce utilizatorul se loghează, 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 comunicarea cu toate dispozitive la care utilizatorul este conectat, iar apoi se vor scoate toate dispozitivele utilizatorului din cache.

[31] MySQL: https://www.mysql.com/

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 întăi se trimite antetul reprezentat de un octet ce va indica comanda, iar apoi conținutul cererii. Protocolul de comunicare are comenzile definite în Figura 11. Primele două comenzi reprezintă răspunsurile posibile: succes sau eroare, următoarele șapte reprezentând cererile ce se pot efectua.

În caz de eroare, dupa octetul pentru protocol, 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.

Celelalte comenzi sunt implementate identic, primul octet este identificatorul comenzii, octeții următori având semnificații diferite, în funcție 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 QUERRY_DATA = 108;

}
```

Figura 11: Comenzile protocolului de comunicare

Metodă	**Descriere**	**Cerere**	**Răspuns**
Open	Comandă ce va deschide dispozitivul.	Doar octetul de comandă.	Octet de success. În caz contrar, octet de eroare urmat de mesaj.
Close	Comandă ce va închide dispozivul.	Doar octetul de comandă.	Octet de success. În caz contrar, octet de eroare urmat de mesaj.

IsOpen	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.
GetType	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.
GetAcceptedParams	Cere parametrii configurabili ai dispozitibului. 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.
Command	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.
QuerryData	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 1: Definirea protocolului de comunicare

Tabelul 1, de mai sus, conține definirea protocolului de comunicare. Coloanele cerere și răspuns se referă la ce se trimite pe rețea atunci când se solicită operație, respectv răspunsul corespunzător cererii.

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

2.2.4. Securitatea serverului

Din punct de vedere al securității între server și client folosec 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 raspunde la cerere. Un token este o secvență de informație care nu are nici o însemnătate singură, dar combinată cu sistemul de tokenizare joacă un rol vital în securitatea serverului.

Atunci când un utilizator dorește să se logheze, 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.

Din cauză că după logare aplicația web trimite la fiecare cerere token-ul primit, am creat un filtru http care interceptează toate cererile de la aplicația web, în afară de logare 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 este invalid, același răspuns se va trimite. 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[32].

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 ca momentan se folosește protocolul HTTP, unde antetele http sunt trimise în clar peste retea.

Dacă se folosește HTTPS[33], 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, dar este în esență același lucru ca traficul HTTP, dar cu avantaje de securitae. [10]

Referitor la securitatea între server și client, așa cum am precizat și in capitolul de ahitectură a sistemului, folosesc protocolul criptorgrafic SSL. Pentru a folosi protocolul criptografic SSL este nevoie de un certificat SSL. Certificatul este un fișier de dimensiune redusă ce leagă cheia criptografică de informațiile unei organizații. Avantajele folosirii acestui protocol sunt [11]:

- criptează datele sensibile. Datele trimise folosind acest protocol sunt criptate, fiind ilizibile pentru cei ce nu au cheia.

[32] HMAC-SHA-512: https://tools.ietf.org/html/rfc4868
[33] HTTPS: Secure Hyper Text Transfer Protocol

- asigură că informația este trimisă la destinatarul real. Certificatul SSL 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.
- apără împotriva escrocherilor. Escrocherii au foarte mari dificultăți în crearea unei replici autentice a unui certificat SSL.
- oferă confidențialiate și încredere.

Pentru conectarea SSL 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 se specifice locația certificatul SSL folosit. În plus, dispozitivele trebuie să specifice și parola certificatului SSL.

Figura 12 conține comanda prin care am generat certificatul SSL. Când comanda se rulează următoarele informații for fi cerute: numele și prenumele, numele unității organizationale, 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
```

Figura 12: Comanda pentru generarea certificatului SSL

DETALIERE CAPTURI ECRAN WIRESHARK

2.2.5. Unelte folosite

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

Swagger

Swagger permite descrierea structurei API-ului dorit astfel încăt și masinile să poată le poată citi. Prin citirea structurii API-ului se poate contrui automat o documentație interactivă și

frumoasă. De asemenea se pot genera automat biblioteci, în mai multe limbi, pentru a explora și alte posibilități cum ar fi testarea automată. [12]

Am folosit Swagger pentru a genera o documentație interactivă a server-ului, și mai ales pentru a putea testa serverul interactiv fară nevoia existenței modulului de aplicație web. Figura 13 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 pot accesa fiecare metodă sunt marcate corespunzător.

Figura 14 surprinde o altă din interfața swagger, ce conține documentarea metodei de ștergere a unui dispozitiv. Metoda necesită 2 parametrii: Authorization și device. În partea dreaptă apare tipul de parametru necesitat, antet http respectiv interogarea a resursei, dar și tipul de dată necesar. În secțiunea „Curl" se afișează request-ul corespunzător cu metoda: ce antete http se pun, care este url-ul corespunzător.

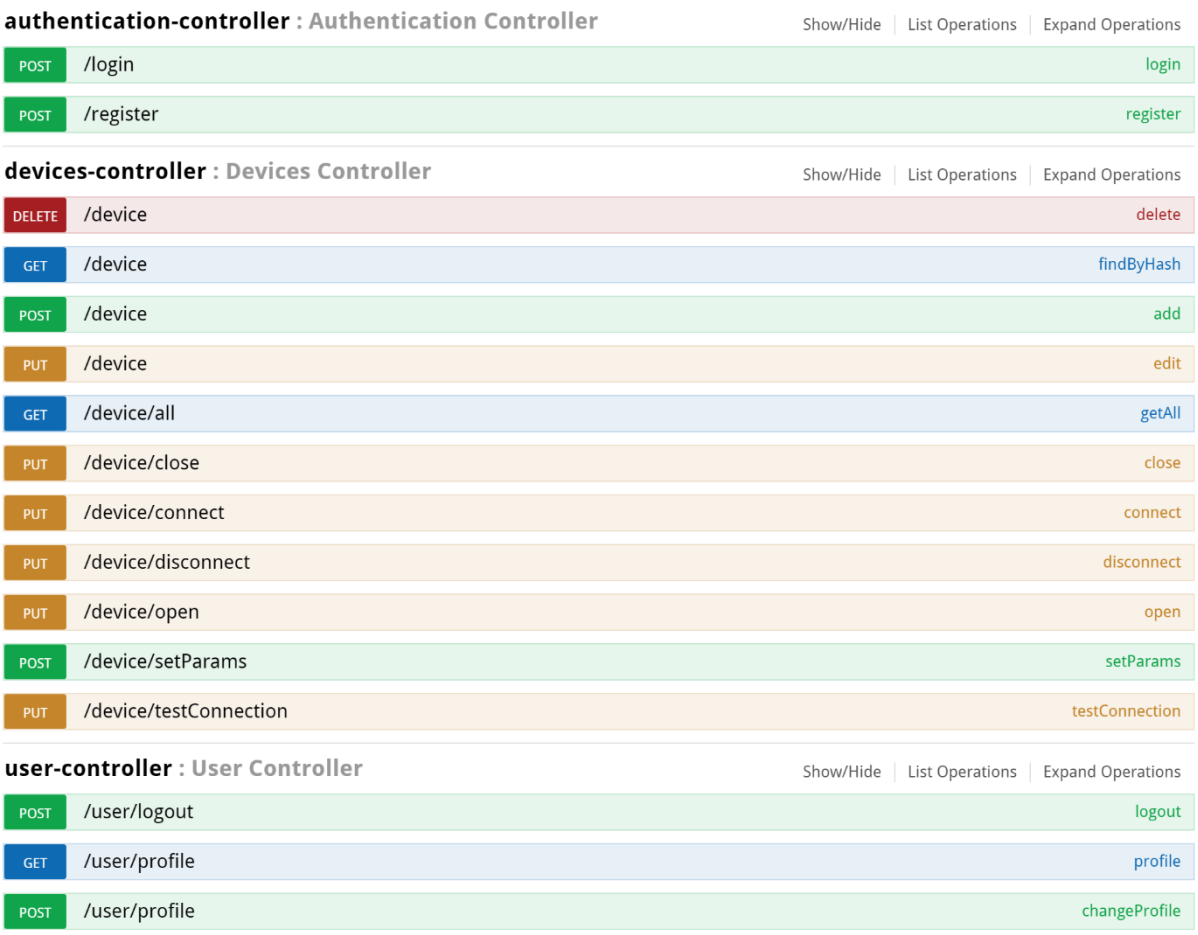

Figura 13: Documentația interactivă a server-ului

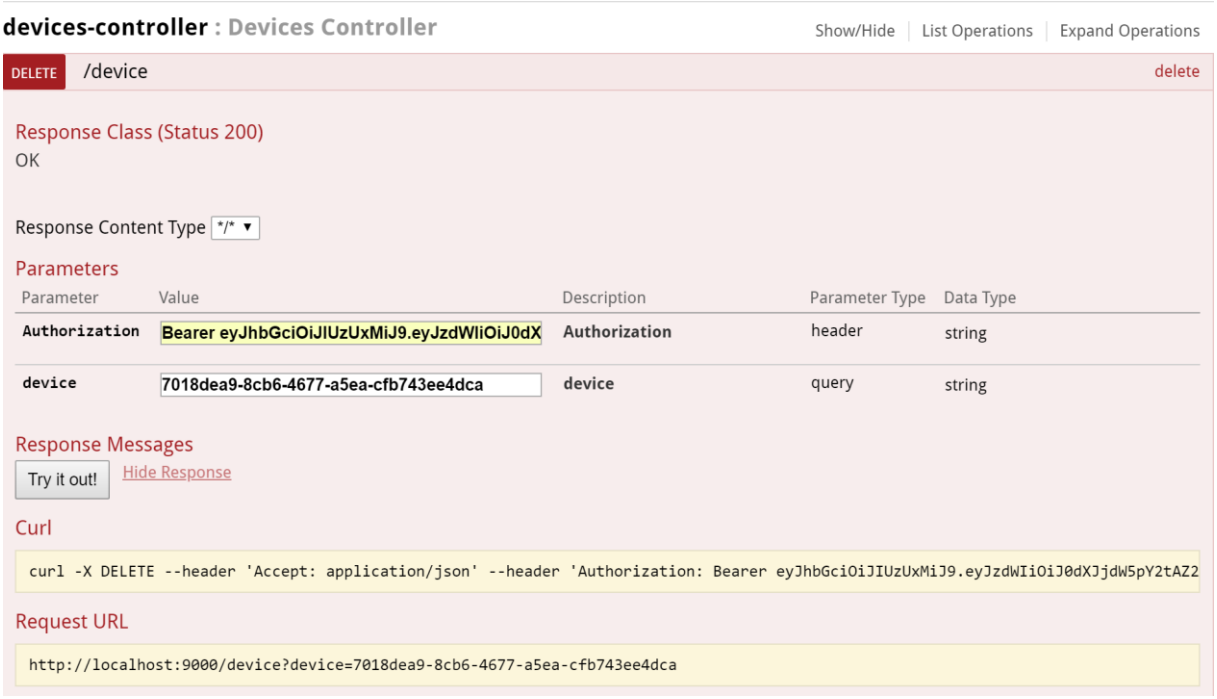

Figura 14: Documentarea unei metode

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

Ngrok

Este un software ce permite dezvoltatorilor să expună la 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. [13]

Motivul pentru care am folosit ngrok este de a verifica aplicația web ca face cererile protrivite în 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 15 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 cererii.

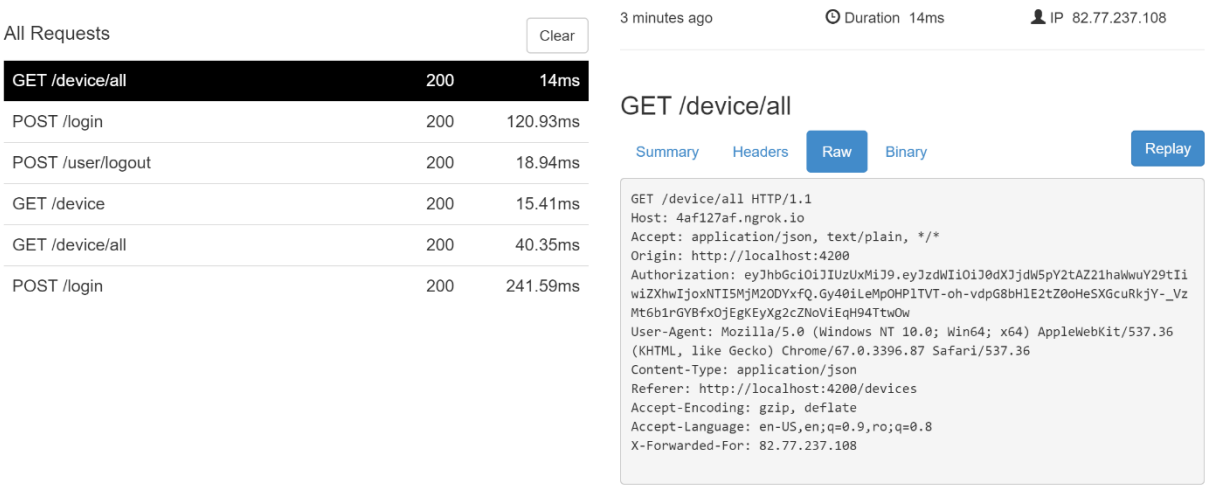

Figura 15: Observarea traficului HTTP dintre aplicația web și server

2.3. Modulul 3 – Dispozitivele „inteligente”

Din punct de vede 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. Protocol SSL necesită un certificat SSL, de acea 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 si protocolul de comuncare”.

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 reprezentănd efectiv dispozitivele inteligente.

2.3.1. 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[34]. 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 eternet al calculatorului la portul de eternet al raspberry-ului. Apoi am scanat rețeaua internă a laptopului, am găsit hostname-ul „raspberrypi” cu ip-ul corespunzător.

DETALIERE – CHECK WITH PI

După, am configurat VNC[35]-ul, un sistem de împărtășire a desktop-ului catre 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

[34] HDMI: High-Definition Multimedia Interface
[35] VNC: Vritual Network Computing

pinilor de pe Raspberry. Dar nu puteam să configurez librăria deoarece nu aveam conexiune la internet.

Din această cauză am configurat partajarea unei conexiuni wifi de la computer la Raspberry Pi folosind un cablu ethernet. Acest tutorial mi-a fost de ajutor: https://www.hackster.io/Anwaarullah/sharing-wifi-with-raspberry-pi-using-a-lan-cable-ae1f44. După această configurare, Raspbery-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 facut update la sistemul de operare și am instalat libraria Java Pi4J.

2.3.2. 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 trebui e 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 numarul parametrilor unui dispozitiv fiind cunoscut la momentul execuției programului. Un alt motiv, mai practic, constă în faptul că dispozitivele nu sunt neaparat 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.

În continuare voi descrie ce dispozitive am implementat.

2.3.2.1. Laser de securitate

Figura 16: KY-008 Modulul senzor laser[36]

[36] Imagine preluată. Sursa: https://www.fasttech.com/product/1219301-keyes-ky-008-arduino-compatible-650nm-laser

2.3.2.2. Lampă

Figura 17: KY-016 Modul LED cu 3 culori[37]

2.3.2.3. Încuietoare a ușii

Figura 18: KY-019 Modul releu 5V[38]

2.3.2.4. Bec

Figura 19: KY-011 Modul LED cu 2 culori[39]

[37] Imagine preluată. Sursa: https://piandmore.wordpress.com/2016/02/10/nodemcu-ky-016-3-color-led
[38] Imagine preluată. Sursa: https://tkkrlab.nl/wiki/Arduino_KY-019_5V_relay_module
[39] Imagine preluată. Sursa: https://www.newegg.com/Product/Product.aspx?Item=1FS-002B-000B7

3. Direcții de viitor

Concluziile lucrării

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] Komei, „Angular5 Fontawesome," [Interactiv]. Available: https://github.com/travelist/angular2-fontawesome. [Accesat 25 05 2018].

[10] B. Pollard, „Secure websites with HTTPS," 25 02 2018. [Interactiv]. Available: https://www.tunetheweb.com/security/https/.

[11] **, „The importance of SSL - Top 6 advantages," Clear Vertical Ltd. 8259274, [Interactiv]. Available: https://www.clearvertical.co.uk/the-importance-and-advantages-of-ssl/.

[12] SmartBear Software, „Swagger," [Interactiv]. Available: https://swagger.io/.

[13] inconshreveable, „ngrok," [Interactiv]. Available: https://ngrok.com/.