

|  |
| --- |
| **Departamentul de Electronică și Calculatoare**  **Programul de studiu: Calculatoare/ Electronică Aplicată/ Tehnologii și Sisteme de Telecomunicații**  **EFICIENTIZAREA CONSUMULUI DE ENERGIE ELECTRICĂ A UNEI CASE INTELIGENTE** |
|  |
| **LUCRARE DE LICENȚĂ** |
| **Coordonator ştiințific:**  **Profesor dr. ing. CAȚARON Angel Doru** |
| **Absolvent:**  **RAICU Tiberiu**  **BRAȘOV, 2019**  Cuprins  [1 Introducere 2](#_Toc11357111)  [1.1 Motivație 2](#_Toc11357112)  [1.2 Casa inteligentă 3](#_Toc11357113)  [*1.2.1 Ce activități pot fi efectuate de o casă inteligentă?* 3](#_Toc11357114)  [*1.2.2 Cum ar trebui să arate o casa inteligentă?* 3](#_Toc11357115)  [*1.2.3 Cum poate fi securizată o casă inteligentă?* 4](#_Toc11357116)  [1.3 Raspberry PI 4](#_Toc11357117)  [1.4 Obiectivele lucrării 5](#_Toc11357118)  [1.4.1 Scenarii propuse 5](#_Toc11357119)  [1.4.2 Monitorizarea dispozitivelor din sfera IoT 5](#_Toc11357120)  [1.4.3  Procesarea evenimentelor complexe 6](#_Toc11357121)  [1.4.4 Controlul dispozitivelor din sfera IoT 7](#_Toc11357122)  [2 Studiu de caz 8](#_Toc11357123)  [2.1 Descriere studiu de caz 8](#_Toc11357124)  [2.2 Arhitectura. 12](#_Toc11357125)  [2.2.1 Deployment diagram 12](#_Toc11357126)  [2.2.2 Modul de interactiune a componentelor software bazat pe scenari 26](#_Toc11357127)  [3 Tehnologii utlilizate 29](#_Toc11357128)  [3.1 Java 29](#_Toc11357129)  [3.1.1 Scurtă caracterizare 29](#_Toc11357130)  [3.1.2 Caracteristici Java : 30](#_Toc11357131)  [3.1.3 Principiile OOP (eng. Object-oriented programming) în Java 30](#_Toc11357132)  [3.1.4 Apache Maven 31](#_Toc11357133)  [3.2 Spring Framework 32](#_Toc11357134)  [3.2.1 Caracterizare 32](#_Toc11357135)  [3.2.2 Avantajele Spring Framework : 32](#_Toc11357136)  [3.2.3 Injectarea Dependințelor 33](#_Toc11357137)  [3.2.4 Spring Boot 33](#_Toc11357138)  [3.3 RabbitMQ 34](#_Toc11357139)  [3.4 CEP (eng. Complex Event Processing) 36](#_Toc11357140)  [3.4.1 Modelul evenimentelor 37](#_Toc11357141)  [3.5 Javascript 37](#_Toc11357142)  [3.6 HTML5 39](#_Toc11357143)  [3.7 Node.js 39](#_Toc11357144)  [3.8 CSS 40](#_Toc11357145)  [3.8.1 Sass (Syntactically awesome style sheets) 41](#_Toc11357146)  [3.9 Bootstrap 41](#_Toc11357147)  [3.9.1 Avantaje 42](#_Toc11357148)  [3.9.2 Dezavantaje 42](#_Toc11357149)  [3.10 ChartJS 43](#_Toc11357150)  [3.11 Angular 43](#_Toc11357151)  [3.11.1 Avantaje 43](#_Toc11357152)  [3.11.2 TypeScript 44](#_Toc11357153)  [3.11.3 RxJS 44](#_Toc11357154)  [3.11.4 Asistență Google 45](#_Toc11357155)  [3.12 REST (REpresentational State Transfer) 45](#_Toc11357156)  [3.12.1 Comunicarea între client și server. 46](#_Toc11357157)  [3.13 MySQL 47](#_Toc11357158)  [3.14 Spring Data JPA (eng. Java Persistence API) 48](#_Toc11357159)  [3.15 JWT (JSON Web Token) 48](#_Toc11357160)  [3.15.1 JSON (JavaScript Object Notation) 49](#_Toc11357161)  [4 Concluzii 50](#_Toc11357162)  [4.1 Posibilități de dezvoltare ulterioară 50](#_Toc11357163)  [5 Bibliografie 51](#_Toc11357164) |

# 1 Introducere

## 1.1 Motivație

În ziua de astăzi fiecare persoană este foarte ocupată și în același timp într-o grabă continuă de a rezolva cât mai multe lucruri într-un timp cât mai scurt.

Ca și consecință a acestui fapt lucrurile pe care trebuie să le ținem minte pentru a duce la bun sfârșit, multitudinea de activități în care suntem implicați zilnic se mărește considerabil, iar șansele ca, din aceste informații, una sau mai multe să ne scape, să fie uitate sunt destul de mari. Sigur, am putea folosi aplicații care să ne aducă aminte de anumite lucruri cum ar fi: ședință la ora 1 sau astăzi este ziua de naștere a prietenului X, dar cele de rutină cum ar fi, stingerea unui bec când s-a părăsit locuința sau scoaterea unui încărcător din priză în momentul în care dispozitivul a fost încărcat în întregime nu pot fi programate cu exactitate.

Toate aceste lucruri le-am observat la mai mulți oameni, dar în special la persoana pe care am putut-o studia toată viața, căreia îi știu modul de gândire și faptul că din când în când , unele lucruri care par triviale, nesemnificative îi scapă, persoana mea. De multe ori uit să fac chiar lucrurile menționate mai sus, iar în consecință banii pe care îi datorez în fiecare lună furnizorului de energie electrică care, de altfel, pot fi economisiți și investiți în alte activități, alte lucruri care să îmi fie de folos, sunt cu câteva procente mai mulți decât este necesar. Pățind aceleași lucruri în fiecare zi, m-am gândit să creez o aplicație care să mă ajute să automatizez aceste activități de rutină pentru a nu trebui să țin minte fiecare lucru mărunt ce poate fi realizat de către o aplicație care poate fi programată în acest scop și să îmi pot folosi mintea la lucruri ce necesită creativitatea, aceste lucruri neputând fi automatizate.

Un alt motiv pentru care am creat această aplicație este faptul că din ce în ce mai multe persoane vorbesc de energie regenerabilă și majoritatea acestora folosesc energia eoliană sau cea solară pe lângă cea oferită de distribuitorii de energie electrică. Atunci când se folosește o astfel de sursă de energie apare următoarea problemă : energia generată nu este suficientă pentru a alimenta o întreagă locuința. Sigur, soluția ar fi simplă, se achiziționează mai multe panouri solare sau mai multe eoliene, dar pentru majoritatea oamenilor efortul financiar este ridicat și probabil nu și-l vor putea permite. Presupunând că avem un panou solar ce generează o anumită energie electrică, insuficientă pentru a alimenta toată casa, cum am putea folosi energia generată cât mai eficient? O soluție ar fi sa distribuim alimentarea circuitelor electrice ale casei în funcție de consumul acestora pe energia neconvențională și alimentarea obișnuită. Atât timp cât consumul de energie de la consumatorii casnici este mai mic sau egal cu cel generat de către panoul solar toți consumatorii vor fi alimentați de la sursa de energie neconvențională. Situația excepțională apare în momentul în care se mai adaugă un consumator la circuitul alimentat de către panoul solar. În acel moment energia generată nu este suficientă și este necesară alimentarea unuia sau a mai multor circuite la rețeaua de distribuție convențională a energiei electrice.

Programul creat ne scapă de această problemă monitorizând fiecare consumator în parte și schimbând automat alimentarea circuitelor astfel încât consumatorii casnici să fie alimentați cu prioritate la energia neconvențională și doar în cazul în care aceasta este insuficientă să se utilizeze rețeaua de distribuție a energiei convenționale.

## 1.2 Casa inteligentă [1]

O casă inteligentă este definită ca spațiul (fie el locuință, birou, casă de vacanță) care utilizează tehnologii moderne pentru automatizarea sistemelor și a aparatelor aflate în acesta. Într-o casă inteligentă aparatele și dispozitivele pot fi controlate automat de la distanță din oricare locație de pe planetă ce este conectată la internet, utilizând un dispozitiv mobil sau alt dispozitiv conectat la rețea. Noile tehnologii vin cu abilități de auto-învățare, prin care pot învăța orarul proprietarilor și se pot ajusta după necesitățile acestora. Locuințele inteligente ce au în dotare controlul iluminatului permit proprietarilor să reducă consumul de energie și, prin urmare, să reducă banii datorați furnizorului de curent electric. Unele sisteme avertizează proprietarul în cazul în care mișcarea este detectată în casă, iar unele pot apela departamentul de pompieri în caz de pericol iminent. Casele inteligente pot fi bazate pe sisteme cu sau fără fir. Sistemele fără fir sunt ieftine și mai ușor de instalat, în timp ce sistemele cu fir sunt considerate mai fiabile și, în general, oferă o securitate crescută. Chiar dacă sistemele cablate sunt mai scumpe decât opțiunile fără fir, instalarea unui sistem cablat poate crește valoarea monetară a unei case.

Practic, o casă inteligentă are ca scop principal confortul individului prin preluarea și ducerea la bun sfârșit a unor activități de rutină ce pot fi automatizate.

### *1.2.1 Ce activități pot fi efectuate de o casă inteligentă?*

Când ne gândim la o casă inteligentă, care este primul lucru ce ne trece prin minte? De obicei răspunsul la această întrebare este una din primele și cele mai simple tipuri de automatizări, controlarea consumatorilor electrici. Modalitățile de control sunt : aprinderea și stingerea becurilor prin intermediul unor sunete pe baza unui contor de timp sau prin folosirea unor senzori pentru a determina prezența în o anumită încăpere, astfel determinând starea de aprins sau închis a luminii. De-a lungul timpului au fost dezvoltate mai multe modalități de a face o locuință să fie monitorizată și controlată de la distanță. Cu o simplă atingere a ecranului unui telefon putem aprinde și stinge dispozitivele din locuință, putem afla și controla temperatura din fiecare cameră, putem afla dacă animalul de companie pe care l-am lăsat singur acasă ne-a dărâmat vaza preferată sau chiar și consumul pe putere electrică.

### *1.2.2 Cum ar trebui să arate o casa inteligentă?*

Într-o casă inteligentă luminile s-ar aprinde concomitent cu mașina care intră în curte, ușa garajului se deschide automat, în casă sunt 21 de grade Celsius, lumina de pe hol se aprindă odată cu deschiderea ușii, iar pe fundal se aude melodia ta preferată, dispozitivele din casă fiind programate de la distanță prin telefonul mobil sau de la locul de muncă prin intermediul calculatorului, chiar înainte de a pleca de la birou.

Datorită nivelului tehnologic ridicat, acum este posibil să visăm la o casă, cu adevărat, așa cum ne-am dorit-o, o locuință capabilă să ne ridice la un alt standard viața.

Noțiunea de casă cu inteligență („brain“) care să se adapteze cerințelor persoanelor care locuiesc într-un asemenea spațiu a apărut încă din 1920, concept introdus de Le Corbusier [2] arhitect modernist. Odată cu evoluția tehnologică, termenul de SMART BUILDING a căpătat noi definiții și utilizări.

Dacă în perioada anilor 1970, această noțiune era folosită pentru spațiile care încorporau tehnologii de eficientizare a energiei utilizate, mai târziu, odată cu dezvoltarea rapidă a PC-ului și a tehnologiilor moderne (perioada 1980), pentru locațiile care dețineau obiecte controlabile prin intermediul calculatorului, în zilele noastre acest concept întrunește caracteristicile unui sistem de management al clădirii și asimilează toate definițiile utilizate anterior.

O clădire inteligentă are componente active care acționează în cadrul subsistemelor din alcătuirea spațiului, mai multe astfel de subsisteme fiind capabile de interacțiuni în interiorul unui sistem central.

### *1.2.3 Cum poate fi securizată o casă inteligentă?*

Securitatea locuinței este unul din principalele criterii funcționale care definesc o casă inteligentă. Prin amplasarea de camere și senzori de mișcare în fața și în interiorul casei, intrușii pot fi ținuți la distanță.

Prin colaborarea funcțiilor expuse de mai multe dispozitive, odată cu detectarea mișcării se pot aprinde luminile interioare „mimând” faptul că cineva se află în casă în acel moment sau chiar pornirea difuzoarelor cu anumite sunete pentru a speria intrușii.

Pe lângă aceste facilități, pe telefonul mobil se pot primi notificări sau chiar mici filmulețe cu zonele în care s-a detectat mișcarea.

## 1.3 Raspberry PI [3]

Raspberry PI este numele unei serii SBC ([Single-board computer](https://ro.wikipedia.org/wiki/Single-board_computer)) realizate de către fundația Raspberry Pi, o organizație de caritate ce își are sediul în Anglia, ce are ca scop educarea oamenilor în domeniul calculatorului.

Raspberry Pi a apărut pe piață în anul 2012, iar de atunci au fost lansate câteva iterații și variații ale acestui produs. Pi original a avut un procesor single-core de 700MHz și doar 256MB RAM, iar cel mai recent model are un procesor quad-core de 1.4GHz cu 1 GB RAM. Prețul de plecare pentru un Raspberry Pi a fost întotdeauna 35 $, toate modelele ce au fost în continuare lansate având acest preț sau chiar unul mai scăzut, inclusiv Pi Zero, care costă doar 5 $.

În toată lumea oamenii folosesc Raspberry Pi pentru a învăța programare, a construi proiecte hardware, sau chiar pentru a face automatizarea unei case.

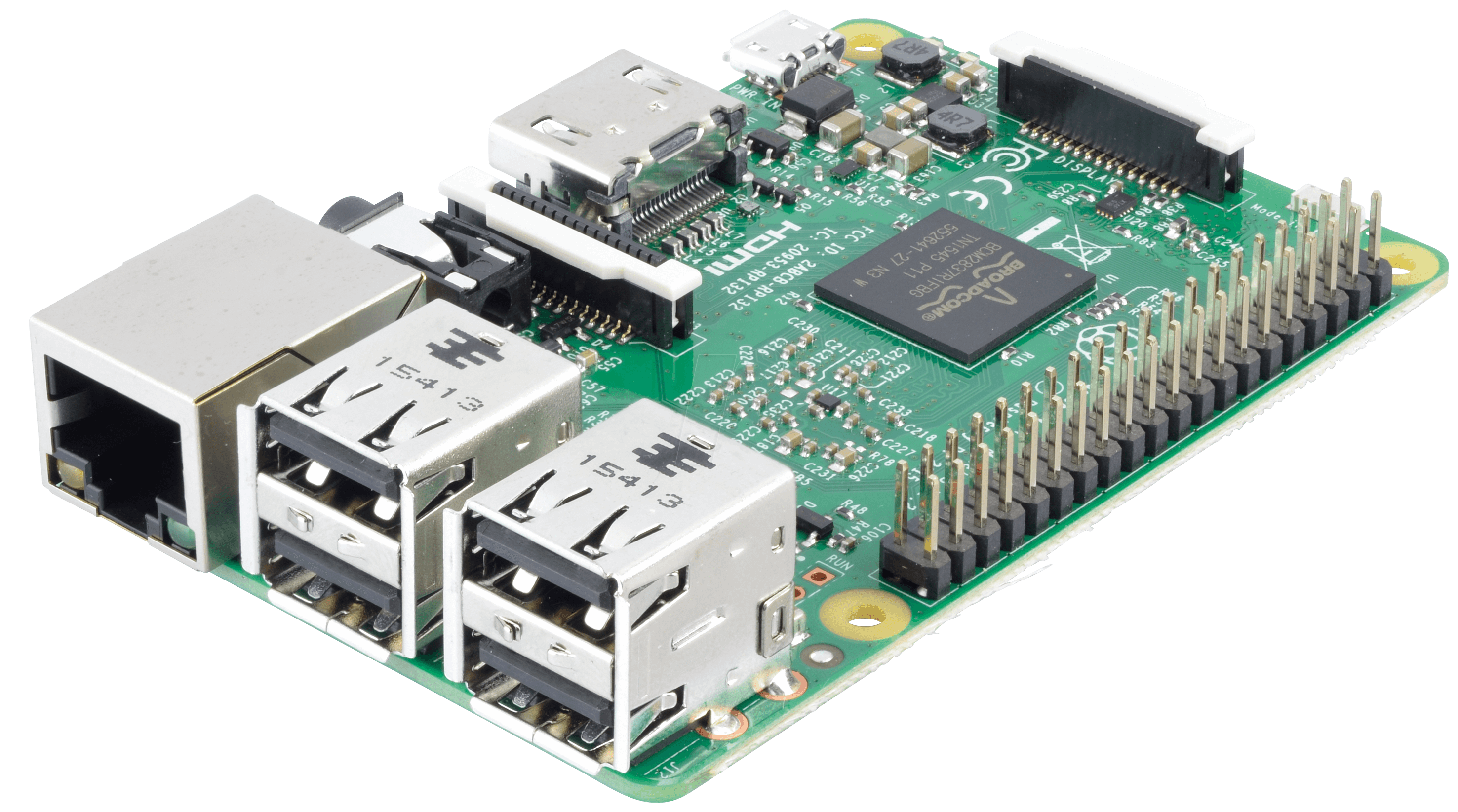


Figura - Raspberry Pi

## 1.4 Obiectivele lucrării

În cadrul acestei lucrări am abordat 3 scenarii :

### 1.4.1 Scenarii propuse

a)    Primul scenariu propune oprirea curentului la o anumită priză dacă la aceasta consumul este foarte scăzut într-o perioadă specifică de timp. Este în principal folosit pentru oprirea consumului de putere al unui încărcător dacă acesta nu alimentează niciun dispozitiv.

b)    Cel de-al doilea scenariu ales face referire la închiderea automată a iluminării unei camere în momentul în care aceasta este inutilă. Cele două elemente pe care acest scenariu le corelează sunt: un senzor de mișcare și starea de aprins/închis a unui bec.

c)    Cel de-al treilea face referire la împărțirea optimă a consumului de energie electrică dintr-o locuința între cea generată de panoul solar și alimentarea normală cu electricitate.

### 1.4.2 Monitorizarea dispozitivelor din sfera IoT [4]

În dicționarul explicativ al limbii române monitorizarea este definită ca: „a supraveghea prin [intermediul](http://dexonline.net/definitie-intermediu) monitorului”.Odată cu trecerea timpului și a evoluției tehnologiei multe din activitățile de rutină ce erau îndeplinite de către oameni au fost înlocuite de dispozitive ce sunt programate doar într-un anumit scop. În trecutul nu atât de îndepărtat, pentru a observa mișcarea într-un anumit spațiu ar fi fost necesar ca o persoană sa fie acolo fizic pentru a îl observa. O altă modalitate, odată cu evoluarea tehnologiei, este de a monta o cameră pentru a supraveghea acel spațiu, o persoană uitandu-se printr-un monitor la clipul video transmis de acea camera. Ambele abordări sunt destul de ineficiente deoarece implică atenția continuă din partea unei fiinițe umane pe o perioaă lungă de timp. În prezent, pentru astfel de activități sunt folosite simple dispozitive ce, la cea mai mica mișcare, vor trimite un semnal de alarmă în cel mai scurt timp posibil. Astfel, in cazul unui pericol iminent, prin folosirea unui astfel de senzor timpul de raspuns va fi îmbunătățit.

Ca și in cazul unei persoane, un astfel de senzor, în functie de locul unde este plasat, poate oferi și date eronate. Aceste erori pot fi datorate unor defecțiuni tehnice, dar si sensibilitătii crescute a senzorului.

Pentru a putea lua decizii datele de la aceste dispozitive trebuie colectate și prelucrate. Într-un “mediu inteligent” o multitudine de astfel de dispozitive sunt monitorizate, iar datele obținute de la acestea sunt prelucrate pentru atingerea unui anumit scop. Pentru monitorizare, acești senzori trebuie să fie conectați la un dispozitiv ce poate colecta, interpreta și transmite mai departe datele.

Din multitudinea de dispozitive existente în acest scop, cele mai uzuale sunt:

* Arduino [5] ce folosește limbajul cu același nume pentru a traduce semnalele oferite de senzori și de a oferi unei persoane o modalitate de a le înțelege.
* Raspberry PI. Poate rula o multitudine de sisteme de operare, acest lucru oferindu-i un grad de superioritate față de dispozitivul anterior menționat prin faptul că se pot folosi mai multe limbaje pentru interpretarea semnalelor oferite de senzori.

### 1.4.3  Procesarea evenimentelor complexe

Odată cu înmulțirea dispozitivelor de tip IoT(Internet of Things), din ce în ce mai multe date au început a fi colectate. Problema apare atunci când un răspuns, ca reacție la datele colectate, trebuie oferit într-un timp foarte scurt. Această [multitudine](https://sin0nime.com/dex/index.php?cheie=multitudine%2C&) a evenimentelor și nevoia identificării și interpretării lor pentru luarea unor hotărâri, a dus la apariția unei noi paradigme de prelucrare și anume Procesarea Evenimentelor Complexe (eng. Complex Event Processing), abreviat CEP [6].

CEP urmărește, după cum ne spune și numele obținerea evenimentelor complexe. Acestea reprezintă evenimentele obținute pe baza datelor generate de un sistem. Un eveniment complex poate fi creat  din evenimente elementare, evenimente elementare și alte evenimente complexe sau numai din evenimente complexe. CEP realizează cu ușurință  prelucrarea evenimentelor de tipuri diferite, provenind din surse diferite.

Conceptul de bază în CEP este fluxul de evenimente (eng. event stream). Un flux de evenimente este o secvență de evenimente ce devin disponibile odată cu trecerea timpului. De exemplu, fluxul de evenimente al unei burse de valori cuprinde evenimentele ce anunță o modificare în prețul acțiunilor. Utilizatorul furnizează interogări către motorul CEP, iar motorul CEP le potrivește cu evenimentele ce sunt conținute în fluxul de evenimente.

CEP iese în evidență față de alte modalități de prelucrare a evenimentelor prin includerea interogărilor temporale (de exemplu: ferestrele de timp). De exemplu, o interogare tipică CEP va arăta în felul următor:

“Dacă valoarea stocului Microsoft [7] a crescut cu mai mult de 15% în decurs de o oră, trimite o notificare”

O astfel de interogare CEP are câteva caracteristici definitorii:

1. În general interogările funcționează continuu, continuând sa emită evenimente, când evenimentele se potrivesc condiției date în interogare.
2. Interogările CEP stochează doar o cantitate mica de evenimente.
3. Răspunsul motorului CEP la o condiție îndeplinită este de ordinul milisecundelor.

### 1.4.4 Controlul dispozitivelor din sfera IoT

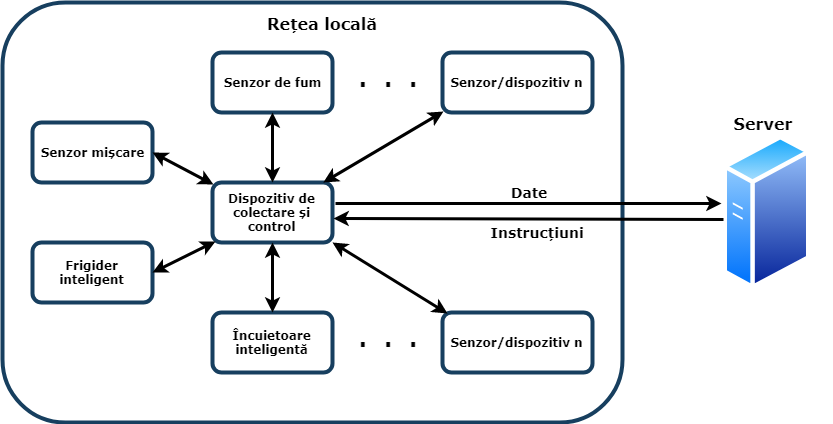
În majoritatea cazurilor, după ce informația ce conține datele fiecărui senzor în parte a fost colectată aceasta este trimisă către un server (un dispozitiv cu o capacitate mare de stocare și procesare), aflat de obicei în afara rețelei locale, pentru a fi interpretată și prelucrată. După terminarea procesului de prelucrare a datelor, în cazul unui sistem complet automat, în cazul

Figura 2 - Schemă de principiu a monitorizării și controlului dispozitivelor IoT

în care anumite standarde sau praguri, ce au fost în prealabil setate, au fost depășite un semnal de comandă este trimis înapoi pentru corectarea unei anumite abateri. În cazul în care sistemul nu este unul automat, încălcarea anumitor praguri anterior setate va rezulta într-o alertă ce va fi afișată pe un ecran, o persoană trebuind să ia decizia cu privire la acțiunea ce în continuare va trebui să fie realizată. În cazul luării unei decizii, ca și în cazul sistemului automat, un semnal de comandă este trimis înapoi către dispozitivul ce anterior a trimis datele pentru corectarea unei anumite abateri sau chiar schimbarea stării unor anumite dispozitive. Acea acțiune este direcționată către un anumit actuator ce trebuie să o execute. Acțiunile pot fi diverse, de la unele simple (de exemplu: aprinderea sau stingerea unui bec, oprirea sau pornirea curentului electric ce alimentează o priză) la unele ce au un grad mai mare de complexitate, de exemplu: folosirea unei camere video pentru analiza alimentelor din frigider și oferirea stării alimentelor și de asemenea a unei liste de cumpărături pentru a ajunge la cantitatea optimă de alimente ce a fost în prealabil setată.

# 2 Studiu de caz

## 2.1 Descriere studiu de caz

Pentru a realiza procesarea datelor provenite de la dispozitivele instalate în locuințe a fost utilizat motorul CEP Esper [8], o bibliotecă scrisă în limbajul Java [9]. Acest motor analizează evenimente simple și detectează situații complexe pe baza acestora (pe baza cărora se pot lua anumite acțiuni în cadrul sistemului) .

În cadrul acestei lucrări au fost luate în considerare 3 scenarii din domeniul casei inteligente. Pe baza datelor (evenimentelor simple) achiziționate de la dispozitivele IoT din casă și corelarea acestora se detectează 3 situații în care consumul de energie electrică poate fi optimizat :

1. Primul scenariu face referire la economisirea puterii consumate de la o priză atunci când la aceasta este conectat un încărcător care nu este folosit pentru o perioadă îndelungată.

Un încărcător lăsat în priză după ce dispozitivul care a fost conectat la el a fost alimentat și deconectat va consuma energie electrică (eng. vampire power). Consumul de putere poate varia de la încărcător la încărcător, dar îl vom considera pe cel cu un consum maxim, de 0.37 W/h ca și punct de referință. Presupunând că acesta va sta conectat  la priză 24 de ore pe zi, 7 zile pe săptămâna pentru un întreg an rezultă un consum de 2.6 kW care este aproximativ egal cu 1,3 lei. Acest cost pare destul de mic pentru un întreg an, dar ținând cont de faptul că într-o familie fiecare are cel puțin câte un dispozitiv electronic ce are nevoie de încărcare (telefon mobil, tabletă, laptop) acest cost poate crește considerabil. Pentru că acest cost să poată fi eliminat a fost considerat un astfel de scenariu.

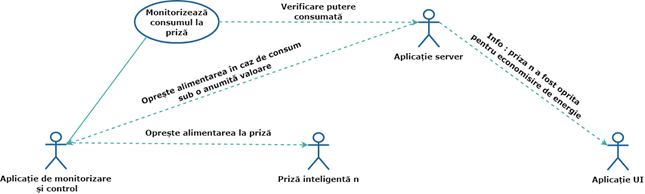
Mod de implementare : Motorul de CEP ascultă evenimentele generate privind consumul de putere al prizei, iar după trecerea unei anumite perioade de timp prestabilită (eng. time\_batch)

Figura - Scenariul 1, Exemplu de utilizare

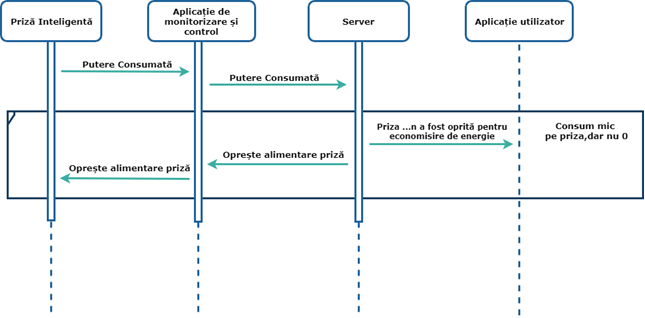
se calculează media aritmetică a energiei consumate pe priza curentă. Dacă rezultatul obținut (energia consumată) este mai mare decât 0, dar mai mic decât o valoare prestabilită (ce semnifică consumul maxim al unui încărcător ce nu este conectat la niciun dispozitiv) rezultă faptul că în acea priză este conectat un încărcător ce consumă ineficient energie. Când motorul de CEP detectează condiția de mai sus generează un eveniment care trimite către sistemul de execuție ce controlează priza comanda opririi acesteia

Figura 4 - Scenariul unu, Diagrama de secvență

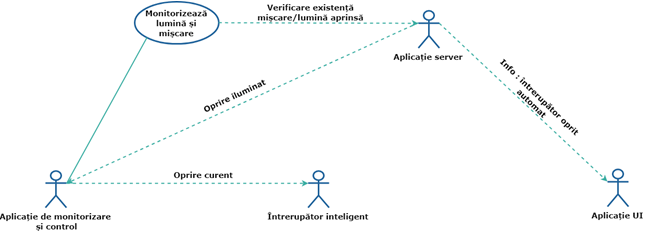
1.  Al doilea scenariu face referire la corelarea informației generate de doi senzori (utilizați pentru detecția mișcării și a stării întrerupătorului) în vederea economisirii puterii consumate de sistemul de iluminare prin detecția situației în care în cameră nu există mișcare.

Figura - Scenariul doi, Exemplu de utilizare

Existența luminii aprinse în locuințe, chiar și atunci când aceasta nu este necesară, este o problema destul de des întâlnită ce conduce la un consum ridicat de energie ce poate fi evitat prin comutarea fizică de către un om a unui întrerupător. Acest consum inutil de energie poate avea multe cauze. O bună parte din acestea o reprezintă era în care trăim, mai exact faptul că toată lumea este în grabă, nimeni un mai are timp de nimic, fiecare dorește să facă cât mai multe lucruri. Ca rezultat al acestei goane după a rezolva cât mai multe lucruri , persoanele pot uita anumite lucruri care aparent nu ar avea un impact atât de mare, nu sunt atât de importante. Printre aceste lucruri se numără și uitarea luminii aprinse în una sau mai multe camere ale locuinței. Presupunând că în medie un om uită lumina aprinsă într-o cameră măcar o dată pe săptămână, iar acesta se presupune că ar dura în medie 8 ore vom avea următorul calcul : un bec de 100W consumă 0.1 kW / h, rezultând un consum de 0.8 kW în cele 8 ore în care becul este aprins. Înmulțind acest număr cu numărul de săptămâni dintr-un an (52) ne va rezulta un consum de 41 kW, iar costul acestor kW poate ajunge la 33 lei. Având în vedere calculul anterior ne putem da seama ce ar însemna dacă am lasă pentru o perioada mai lungă de timp sau mai multe becuri aprinse, costul pe an ar fi destul de ridicat.

Mod de implementare : Motorul de CEP procesează evenimentele generate privind consumul de putere al fiecărui bec în parte. Aceste date le corelează cu cele venite de la senzorul de mișcare instalat în aceeași camera cu becul în cauza (legătura lor fiind făcută de către utilizator în momentul configurării acestui scenariu prin adăugarea în baza de date id-ul becului și id-ul senzorului corespunzător acestuia) iar dacă, după trecerea unei anumite perioade de timp prestabilite (aceasta fiind setată tot de utilizator la configurare) nu a fost detectată mișcare în camera se generează un eveniment de închidere a întrerupătorului ce controlează becul în cauză.

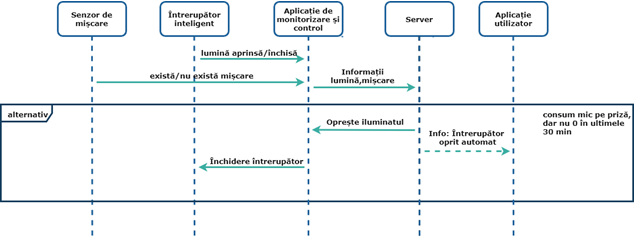


Figura 6 - Scenariul doi, Diagrama de secvență

1. Al treilea scenariu ne permite să detectăm o creștere bruscă a consumului de la o priză și să recalculăm modul de alimentare a circuitelor (de exemplu : panou solar fotovoltaic , alimentarea normală) prioritizând utilizarea energiei verzi produsă de panourile solare.

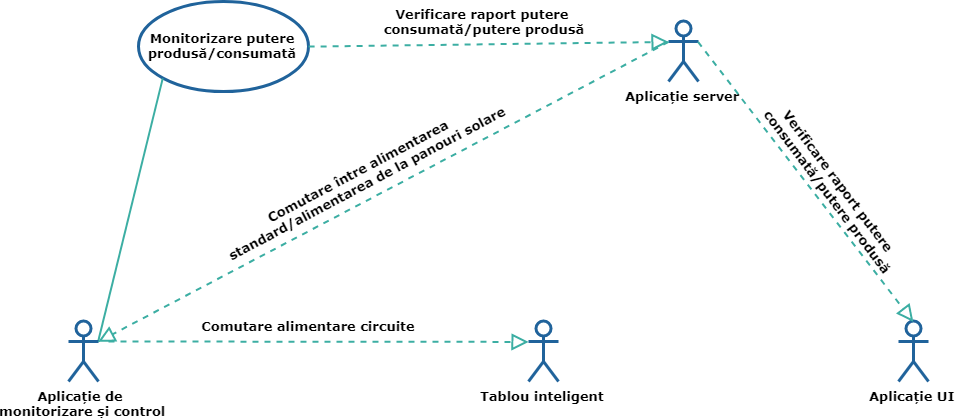
Locuințele au instalația electrică compusă din mai multe circuite (de exemplu: circuite ce alimentează prizele -> acestea consumă mai multă energie electrică, circuite de iluminat -> alimentează becurile și restul corpurilor de iluminat ale casei). Cum acestea sunt independente unul față de celălalt, alimentarea se poate face separat ceea ce ne permite să avem o flexibilitate mult mai mare în utilizarea cât mai eficientă a puterii generate de către un panou solar fotovoltaic. Pe piață există contoare electrice cu comutator de circuite ceea ce înseamnă că un circuit își poate schimba sursa de alimentare printr-un semnal primit de la un dispozitiv exterior.

Figura - Scenariul trei, Exemplu de utilizare

Mod de implementare :  Motorul de CEP ascultă evenimentele generate privind consumul de putere al fiecărui consumator în parte. Următorul pas este compararea puterii consumate a 2 măsurări consecutive ale aceluiași dispozitiv (consumator). În cazul în care puterea consumată pe un circuit este mai mare decât cea anterioară (aprinderea unui bec, conectarea unui nou consumator la o priză) se generează un eveniment care preia din baza de date toate circuitele alimentate de panoul solar și recalculează puterea consumată de acestea. În cazul în care aceasta nu depășește puterea generată de panou nu se va produce nicio schimbare. Pe de altă parte, în caz că noua putere consumată pe circuit este mai marte decât cea generată se preiau toate circuitele din baza de date , se calculează puterea consumată pentru fiecare, iar apoi se găsește cea mai bună posibilitate de aranjare a acestora astfel încât puterea generată de panoul solar să fie folosită cât mai optim, alegând combinația de circuite ce se apropie cât mai mult de această putere generată. Circuitele rămase, cele ce au fost inițial alimentate de energia panoului solar își vor schimbă alimentarea la cea normală. După ce a fost calculată noua configurare de alimentare a circuitelor din casă va fi trimisă o comandă cu noua configurație de alimentare a circuitelor pentru a realiza comutarea alimentării celor necesare.

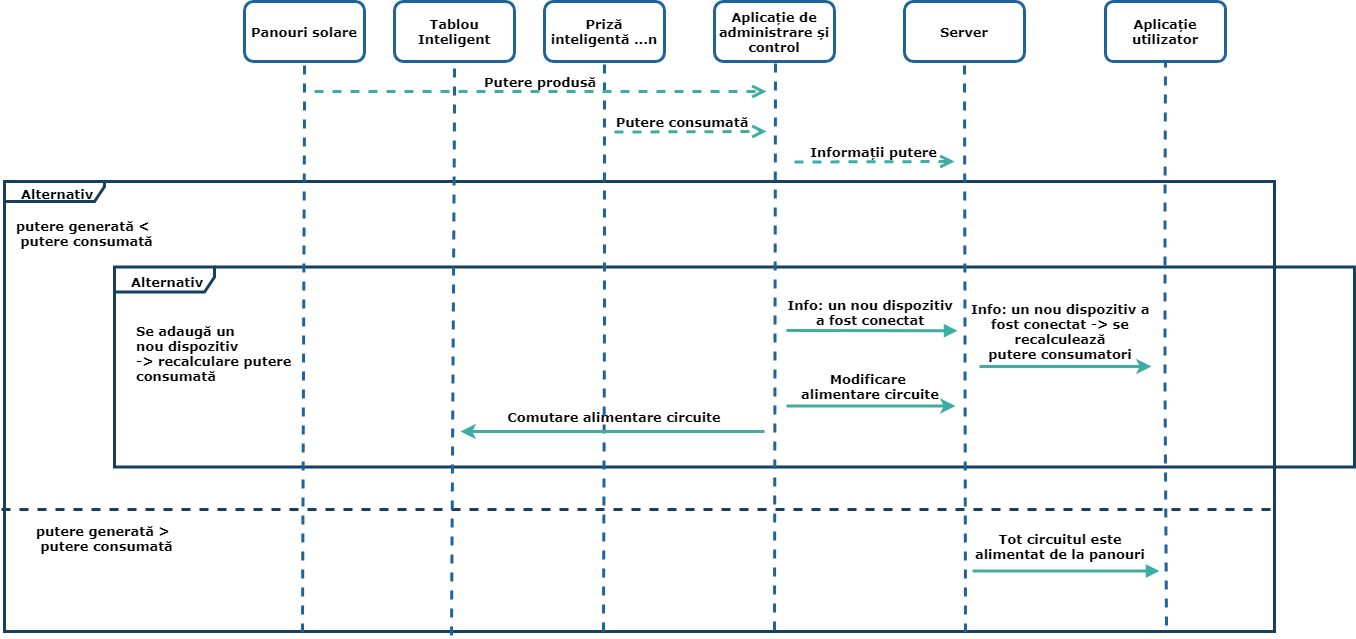


Figura 8 - Scenariul trei, Diagrama de secvență

## 2.2 Arhitectura.

### 2.2.1 Deployment diagram

Proiectul are în componență patru aplicații ce comunică între ele prin diverse servicii de transmitere a datelor. Două dintre acestea rulează pe un dispozitiv electronic numit Raspberry PI, iar celelalte două rulează în cadrul unui laptop.

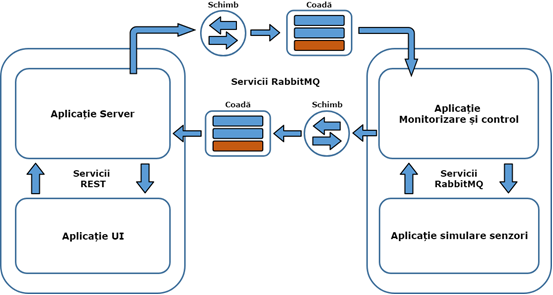
Structura modulară a proiectului oferă mai multă stabilitate și în același timp crește ușurința testării. După cum se va putea observa și în paginile viitoare, această tehnică a modularizării este folosită și în fiecare aplicație ce aparține acestui proiect.

Figura 9 - Diagrama logică a componentelor proiectului

Structura modulară a proiectului aduce, odată cu multitudinea de avantaje și un dezavantaj major: în cazul în care conexiunea la internet este întreruptă funcționarea aplicațiilor este oprită deoarece datele și instrucțiunile nu se mai pot transmite între aplicații.

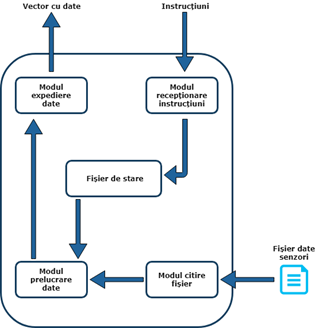
***Aplicație simulare senzori***

Figura 10 – Diagrama logică a aplicației simulare senzori

Această aplicație, după cum sugerează și numele, este cea care „pune în mișcare” proiectul. Are ca scop simularea senzorilor preluând date dintr-un document de formă tabelară și trimițându-le către aplicația de monitorizare și control. Programul citește linie cu linie la interval de o secundă fișierul de tip tabel și încapsulează datele în format obiect pentru a putea fi trimise mai departe. Trimiterea datelor se realizează prin intermediul unui broker de mesaje numit RabbitMQ [10]. Pentru a trimite date prin acest serviciu, ele sunt criptate cu o cheie și puse într-o coadă (eng. queue) de așteptare astfel că informația, chiar dacă sunt întârzieri din cauza vitezei internetului sau a altor factori, va ajunge la un moment dat la aplicația de monitorizare și control. O altă funcție a acestei aplicații o reprezintă recepționarea, prin același serviciu de mesagerie menționat anterior, și executarea instrucțiunilor primite, instrucțiuni ce conțin starea pe care un anumit dispozitiv trebuie să o aibă. Executarea instrucțiunilor implică rescrierea unui fișier de stări ce are în componență starea de oprit/pornit a fiecărui dispozitiv electronic simulat, fie că este senzor, întrerupător sau priză. În componența fișierului de stări se regăsește fiecare circuit existent în casă și modul în care se face alimentarea acestora, energia electrică putând proveni fie de la panoul, fie de la alimentarea normală.

#### Aplicație achiziție și control

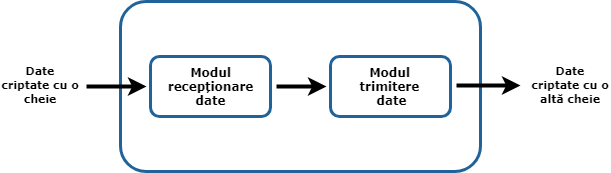
Cea de-a două aplicație face legătura între partea hardware (simulată de aplicația anterioară) și partea de server, ea aflându-se pe un dispozitiv electronic numit Raspberry PI. Funcționalitatea acestei aplicații este asemănătoare cu cea a unui router wireless, preia datele încapsulate sub formă de obiect dintr-un canal folosind o anumită cheie (cu care datele au fost criptate în aplicația anterioară), iar după finalizarea acestei acțiuni, datele sunt plasate într-un alt canal folosind o altă cheie de criptare. Este compusă din 2 module.

Figura 11 - Diagrama logică a aplicației de achiziție și control

Primul funcționează ca și receptor. Ascultând mereu un anumit canal în care sunt adăugate date, la interval de o secundă, de către aplicația ce simulează senzorii. În momentul în care datele au fost recepționate, acestea sunt trimise către al doilea modul al aplicației, modulul de expediere a datelor. Acest modul preia datele, le criptează cu o anumită cheie urmând ca mai apoi să le trimită unui schimb (eng. exchange), urmând ca acesta să le plaseze într-o anumită coadă pentru a fi recepționate de aplicația server. În momentul în care datele au ajuns la aplicația server, aceasta trimite înapoi un mesaj de confirmare. Acest mesaj de confirmare este adăugat într-un fișier ce conține fiecare activitate efectuată de această aplicație.

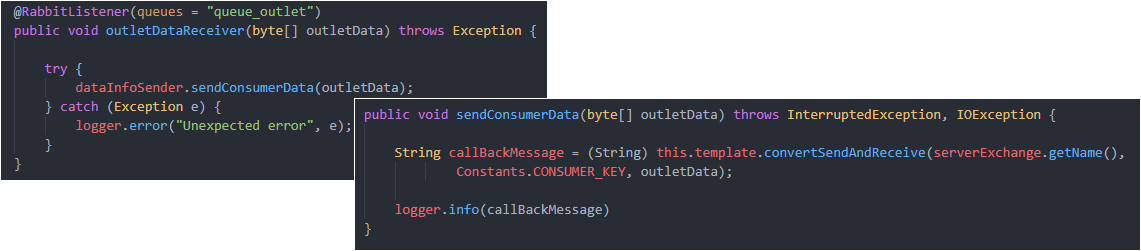


Figura 12 – Recepționarea și trimiterea datelor în cadrul aplicației de achiziție și control

***Aplicație server***

Este „sediul central” al proiectului îndeplinind concomitent o multitudine de funcții. Funcția principală a aplicației, după cum spune și numele, este cea de server, permițându-i să fie accesată de pe orice dispozitiv ce are o conexiune la internet. Ca și aplicațiile anterioare, și aceasta este împărțită în mai multe module pentru a face testarea mai ușoară dar și pentru ca oricine să poată implementa noi funcționalități unor anumite părți ale aplicației fără a introduce erori în alte zone:

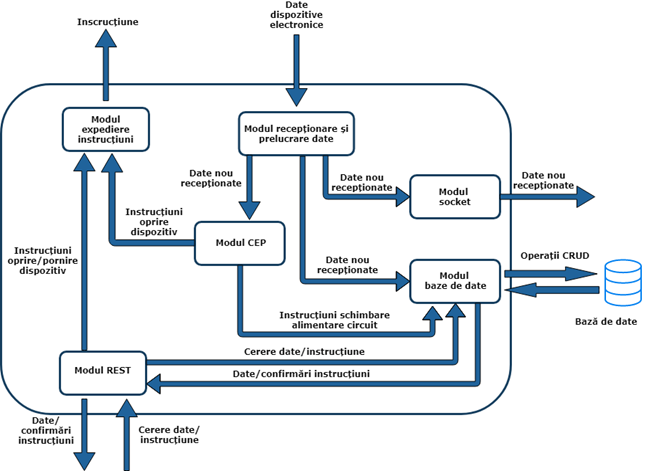


Figura 13 - Structură aplicație server

1. Modulul de recepționare și prelucrare a datelor primite de la aplicația de achiziție și control.

Acest modul funcționează ca și un receptor. Ascultă continuu canalul în care sunt adăugate date, la interval de o secundă, de către aplicația de administrare și control. În momentul în care un mesaj a fost recepționat, acesta este decriptat cu ajutorul cheii (eng. routing key) și transformat din formatul “byte”, ce este necesar pentru transferarea datelor în format “obiect” pentru a putea fi mai departe prelucrate. După terminarea prelucrării datelor, acestea sunt transferate în același timp către alte două module, primul fiind modulul CEP, iar cel de-al doilea, modulul de baze de date.

1. Modulul CEP

Acest modul conține funcționalitatea de bază a proiectului. Datele primite de la modulul anterior, cel de recepționare, sunt alocate, în funcție de conținut, unuia sau mai multor scenarii. În cazul în care datele primite au ca și caracteristică te tip „priză”, acestea vor fi direcționate către cele 2 scenarii ce verifică constant consumul pe fiecare priză. Ca și o descriere succintă, primul scenariu verifică dacă puterea consumată pe unul sau mai multe circuite este prea mare pentru energia generată de panoul solar, iar cel de-al doilea, dacă puterea consumată de o priză la un moment dat este prea mică, având o valoare prestabilită de prag. În cazul în care tipul datelor este referit ca „întrerupător” sau „senzor” datele sunt atribuite celui de-al treilea scenariu astfel încât, dacă valoarea ce semnifică detecția mișcării are valoarea zero pentru o perioadă mai mare decât stabilită, iar valoarea consumului întrerupătorului este diferită de zero, un eveniment va fi declanșat. La declanșarea unui eveniment, în funcție de scenariul ce l-a emis, unul sau mai multe module ale serverului vor fi notificate în vederea transmiterii unor instrucțiuni.

Pentru cel de-al treilea scenariu, atât modulul de trimitere instrucțiuni (ce va trimite instrucțiunea de schimbarea a configurației alimentării circuitelor către tabloul electric inteligent), cel de baze de date (ce va modifica configurația deja existentă a alimentării din baza de date), dar și cel socket (ce va trimite o notificare către aplicația UI în vederea înștiințării utilizatorului) vor fi notificate.

Pentru primele doua scenarii doar modulul socket, dar și cel de trimitere a instrucțiunilor vor fi notificate. Ca și în cazul celui de-al treilea scenariu, modulul socket va trimite către interfața Web o notificare în care se va specifica faptul că o modificare a stării unui dispozitiv avut loc, dar și o comandă de a modifica starea unui comutator din cadrul interfeței, utilizatorul putând observa acest lucru fără a fi nevoit să reîncarce pagina. Modulul de trimitere a instrucțiunilor va trimite instrucțiunea oprire a prizei sau a întrerupătorului către aplicația de monitorizare și control.

1. Modulul de baze de date.

După cum sugerează și numele, cel ce are rolul de a interacționa cu baza de date, mai exact, de a efectua operații CRUD (eng. Create, Read, Update, Delete), crearea de tabele, rânduri, coloane, etc și citirea, actualizarea sau ștergerea acestora. Prin acest modul, nu doar conținutul serverului, dar și celelalte aplicații ce fac parte din proiect devin conectate la baza de date. Legătura cu baza de date este realizată prin intermediul unei tehnologii numite JPA care are rolul de a diminua pe cât posibil nevoia programatorului de a scrie cod de tip SQL.

Baza de date folosită este MySQL [11]. Această bază de date vine cu un server propriu, ceea ce permite accesarea bazei de date chiar dacă aplicația ce are nevoie de acele date nu se află pe aceeași mașină. Accesarea bazei de date se face prin intermediul unei legături (eng. link) ce este de forma: jdbc:mysql://localhost:3306/date. Prima parte a acestui link este reprezentată de API-ul (eng. application programming interface) specific Java ce definește modul de conectare la baza de date, iar al doilea cuvânt semnifică baza de date folosită. A treia parte acestui link ne arată locația dispozitivului în cadrul căreia baza de date rulează (în cazul de față, rețeaua locală), iar ce-l de-al patrulea cuvânt indică numele bazei de date. În cazul unei baze de date publice, pentru conectare și efectuarea de operațiuni, autentificarea nu este necesară. Baza de date folosită, având conținut privat, necesită furnizarea unui nume de utilizator și a unei parole pentru accesarea și modificarea datelor. Toate aceste date de autentificare, dar și restul configurațiilor necesare pentru funcționarea optimă a modulului se regăsesc într-un fișier de configurație ce este localizat în folderul de resurse al programului. Ca și îndatoriri, acest modul preia fiecare obiect trimis de către modulul de recepționare și le inserează în baza de date știind, prin intermediul JPA [12], să asocieze fiecare atribut al obiectului cu tabelul și coloana corespunzătoare din baza de date. Pentru ca JPA sa poată realiza legătura dintre Java și baza de date și implicit pentru a limita pe cât posibil scrierea de cod SQL la crearea entităților, deasupra corpului clasei trebuie adăugată o adnotare @Entity.

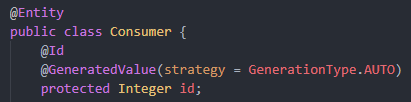


Figura 14 – Crearea unei entități JPA

Prin adăugarea acestei adnotări, JPA va considera clasa ca fiind o entitate și va crea în baza de date un tabel cu același nume, iar variabilele definite în corpul clasei vor fi coloanele tabelului. Pentru operațiile de tip CRUD este nevoie de o clasă separată numită “repository” ce conține principalele metode necesare creării, inserării, actualizării și ștergerii datelor în baza de date.

Pe lângă funcționalitatea enunțată mai sus, acest modul poate face și alte interogări sau inserții venite de la module interne (de exemplu : schimbarea alimentării unui circuit ca și rezultat al prelucrării unor date de către modulul CEP) sau de la interfața web în care utilizatorul poate schimba starea unor dispozitive (de exemplu : aprinderea sau stingerea luminilor) sau chiar să ceară anumite date privind consumul.

Acest modul este unul dintre cele mai solicitate deoarece majoritatea acțiunilor enunțate mai sus se realizează la un interval foarte mic una de cealaltă. Datele venite de la senzori trebuie constant introduse, iar aplicația web trebuie continuu actualizată (la un interval de o secundă). Dezavantajul acestei continue accesări ale bazei de date este folosirea unei mari părți ale resurselor mașinii pe care serverul rulează, acest lucru reducând semnificativ performanța programului.

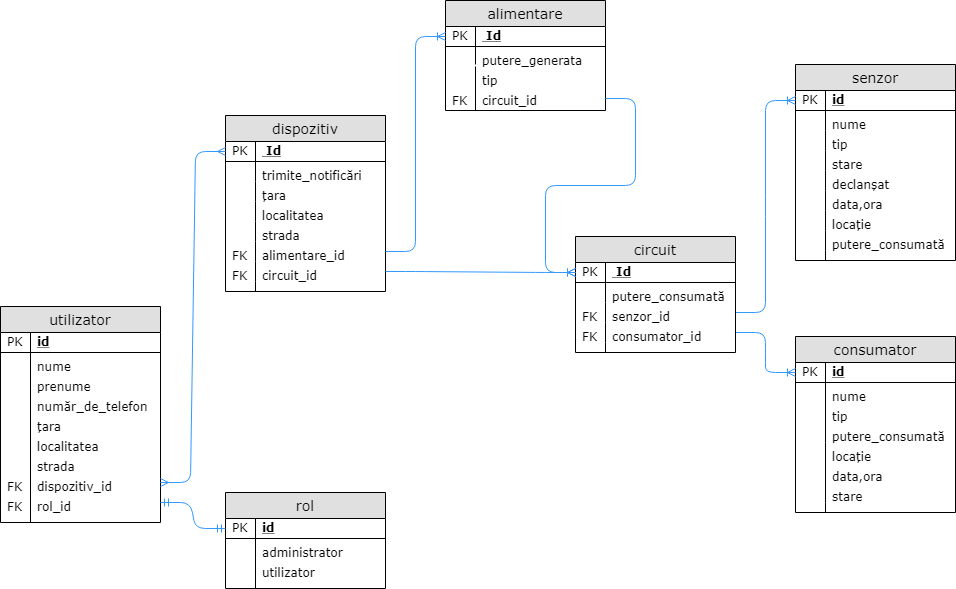
Mai jos este prezentată diagrama bazei de date a proiectului:

Figura 15 - Diagramă model bază de date

După cum se poate observa tabelele sunt legate între ele, ceea ce ajută la o navigare mai ușoară a bazei de date.

1. Modulul REST(eng. Representational state transfer) [13].

Realizează legătura dintre aplicația Server și cea de UI (web). Acest modul facilitează comunicarea între cele două aplicații, ce rulează independent una de cealaltă, prin intermediul unor cereri (eng. requests) realizate de aplicația UI ce sunt destinate serverului. În funcție de necesitatea cererii primite, acest modul o direcționează către un altul pentru a fi soluționată. Soluționarea poate fi sub formă de oprire sau pornire dispozitiv, această comandă fiind recepționată de către un “controller” (după cum se poate observa și în figura 16), iar mai apoi fiind trimisă către modulul de trimitere de instrucțiune, în urmă căreia rezultă o confirmare sau eroare în executarea acțiunii.

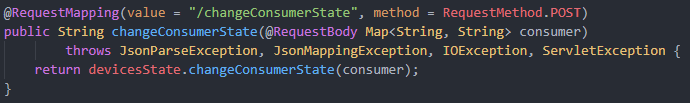


Figura 16 – Exemplu controller de tip REST

Un alt tip de cerere este aceea în care, ca și răspuns, aplicația UI așteaptă un set de date. Pentru a obține datele se trimite o cerere către modulul de baze de date, iar acesta ca răspuns va oferi informația dorită. Timpul de soluționare a acestei cereri depinde în principal de resursele mașinii pe care rulează serverul de baze de date, acest lucru putând fi considerat un avantaj, dar și un dezavantaj. După primirea setului de date, ce este de obicei împachetat în format JSON (eng. JavaScript Object Notation) [14], se trimite răspunsul către aplicația UI.

1. Modulul Socket [15].

Acesta are rolul de a trimite notificări de tipul “push” către interfața web fără ca aceasta să trimită în prealabil o cerere în vederea obținerii acestor notificări. Pentru ca această acțiune să fie posibilă, între aplicația UI și server se creează în prealabil o conexiune la încărcarea paginii. În momentul în care acea conexiune a fost realizată, în canalul creat serverul poate introduce date fără că acestea să fie explicit cerute de către aplicația UI. La sfârșitul executării unor anumite evenimente, spre exemplu : schimbarea sursei de alimentare a unui circuit, oprirea unor consumatori sau senzori, se trimite automat către interfața web o notificare ce conține un text anunțând faptul că acel eveniment a avut loc. O altă întrebuințare a modulului este trimiterea datelor provenite de la modulul de recepționare (de la aplicația de monitorizare și control) către aplicația UI. În momentul în care datele despre senzori sunt recepționate, se verifică existența acelei entități în baza de date. Dacă răspunsul este pozitiv, datele sunt trimise către modulul socket. În caz contrar se creează o nouă entitate, după care datele sunt din nou trimise către modulul socket. La primirea datelor acesta le expediază către aplicația Web pe canalul deja creat între aceste 2 aplicații.

O altă întrebuințare a acestui modul este trimiterea periodică (la interval de o secundă) a datelor ce definesc starea dispozitivelor electronice (prize, senzori, întrerupătoare) către aplicația UI pentru ca utilizatorul să poată sesiza în timp real procesele desfășurate în mod automat fără a fi nevoit să reîncarce pagina.

#### Aplicație UI

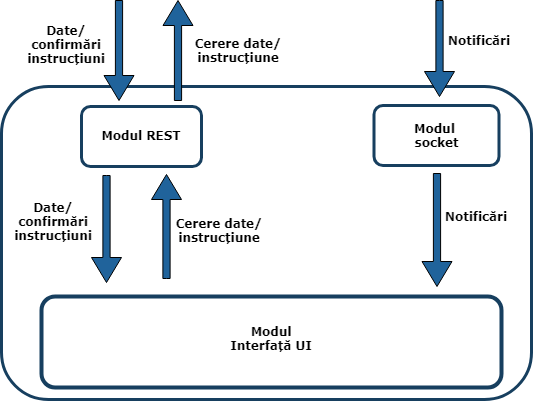


Figura 17 - Structură aplicație UI

Dintre toate aplicațiile existente în acest proiect, aceasta este cea cu care utilizatorul

interacționează în mod direct. În cadrul interfeței pot fi vizualizate atât notificări ale finalizării executării evenimentelor ce se desfășoară în mod automat în cadrul aplicației server dar și grafice ale datelor ce caracterizează dispozitivele electronice simulate în timp real. Tot în cadrul interfeței pot fi vizualizate și stările în care dispozitivele electronice se află. Utilizatorul, prin schimbarea stării unor comutatoare, poate comanda direct starea de pornit/oprit a dispozitivelor electronice. Aceasta constă într-o aplicație web creată folosind un framework al javascript-ului [16] numit Angular [17]. Acest framework continue toate uneltele (eng. tools) necesare creării unei aplicații web de tip enterprise, de la animarea interfeței, până la modalități de comunicare cu alte aplicații. Decizia de a scrie aplicația UI ca și o aplicație web a fost luată având în vedere mai multe motive. În ziua de astăzi, majoritatea companiilor, serviciilor și chiar a persoanelor fizice au măcar un site web pe care utilizatorii (indiferent că sunt simpli utilizatori sau alte companii) îl pot accesa pentru a citi anumite informații sau pentru a beneficia de un anumit serviciu. Acest lucru este posibil deoarece toată lumea este conectată la internet. Beneficiul adus de acest trend, această multitudine de site-uri web este dat de cursa producătorilor de unelte (eng. tools) pentru determinarea și convingerea programatorilor de a-l folosi pe cel creat de ei. Beneficiile aduse de această competiție sunt diverse: îmbunătățirea continua a vitezei, sporirea securității datelor, scrierea din ce în ce mai succintă a codului, dar în același timp acțiunile realizate sunt într-un număr din ce în ce mai mare. Existând un număr considerabil de site-uri web, numărul de probleme ce pot fi rezolvate doar printr-o simplă căutare pe un motor de căutare este destul de mare. În crearea de astfel de aplicații un programator (ca și în alte tipuri de aplicații) se va lovi de un număr considerabil de erori, dar cum comunitatea este mare și mereu în creștere, majoritatea problemelor deja au un răspuns. Un alt motiv pentru care am ales crearea unei aplicații web ca și mod de a interacționa cu utilizatorul este faptul că, fiind website, poate funcționa în cadrul a mai multe platforme, fie ele PC, tabletă sau telefon inteligent. Avantajul oferit constă în crearea unei singure aplicații ce se adaptează platformei din cardul căreia este accesată, forma și componentele existente vor avea o așezare diferită în funcție de mărimea ecranului.

Pentru a avea acces la facilitățile aplicației web utilizatorul trebuie să își creeze un cont prin

introducerea unor date personale. După confirmarea creării contului, utilizatorul poate accesa

aplicația folosind e-mailul și parola introduse în momentul în care contul a fost creat. Logarea

unui utilizator se face prin intermediul unei tehnologii numite JWT (eng. JSON Web Tokens) [18].

Tot prin intermediul acestei tehnologii se verifică cererile primite de server ce solicită acces la

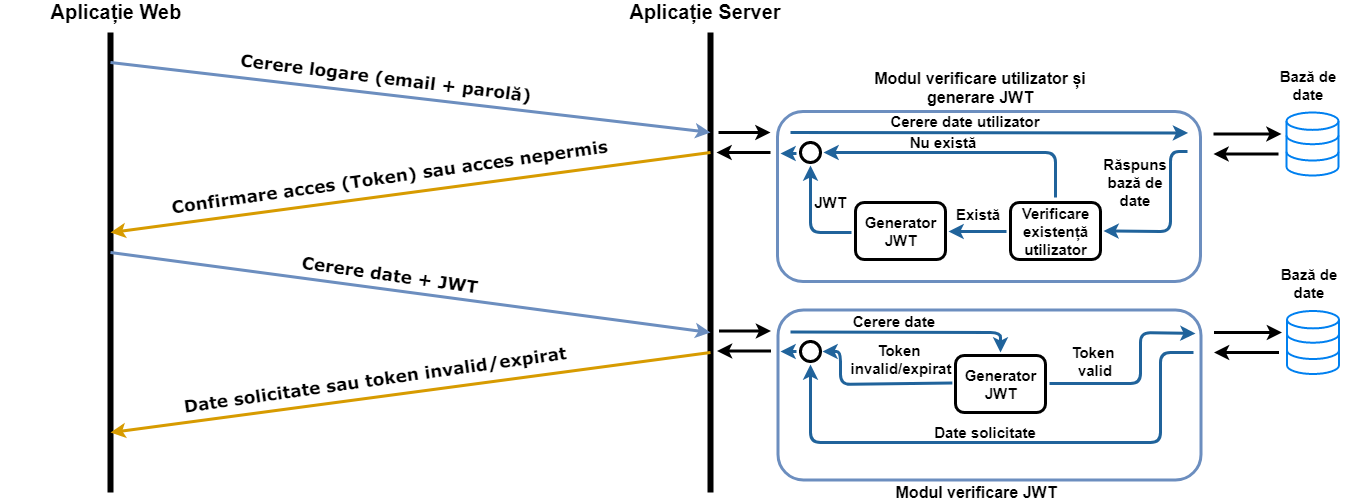
resurse private.

Figura 18 - Structura autentificării de tip JWT (JSON Web Token)

Pornind de la premisa că nu există erori, procesul de logare folosind JWT decurge astfel: utilizatorul completează câmpurile necesare autentificării (email și parolă) apoi le trimite către server folosind butonul “autentificare”. Odată ce informațiile de autentificare au ajuns la server, se interoghează baza de date în căutarea unor date identice. În cazul în care o potrivire a fost găsită (utilizatorul există în baza de date) se generează un token unic folosind o cheie de criptare și se trimite către aplicația web împreună cu confirmarea autentificării.

În cazul în care datele de autentificare trimise către server nu sunt corecte, la verificarea existenței acestora în baza de date, răspunsul rezultat va fi negativ. În acest caz se va trimite ca răspuns, către server, un mesaj de eroare cu codul codul 401 – Neautorizat.

După logare, în cazul în care o cerere este trimisă către server solicitând acces la o resursă privată procesul de validare a acestei cereri, decurge astfel: în “header-ul” cererii trimise se introduce un nou câmp numit token (acesta fiind token-ul generat anterior în procesul de logare). În momentul în care cererea a ajuns la aplicația server, se verifică validitatea token-ului. În cazul în care validitatea token-ului este confirmată, cererea decurge normal, trimițând ca răspuns datele necesare aplicației web. În cazul în care token-ul este greșit sau expirat se va trimite către aplicația web un mesaj de eroare.

În cazul deteriorării token-ului în timpul transmisiei, următoarea situație va avea loc: în momentul în care cererea a ajuns la server, se reia procesul de creare a token-ului folosind cheia de criptare, iar apoi se compară cu cel primit rezultând într-un răspuns negativ. În urma rezultatului primit, cererea de date va fi automat respinsă, trimițând ca răspuns către aplicația server un mesaj de eroare având codul 401 – Neautorizat.

Din punct de vedere arhitectural aplicația este compusă din trei module:

1. Modulul REST.

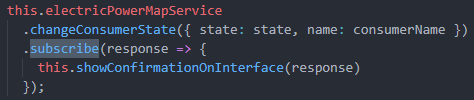
Este principala legătură a aplicației UI cu cea de server. Această legătură se realizează prin cereri de tip REST (REpresentational State Transfer) ce permit celor două aplicații să lucreze independent una de cealaltă. Pentru că aplicația să nu intre în stare de blocare în așteptarea răspunsului fiecărei cereri trimise, comunicarea dintre cele două aplicații se realizează în mod asincron. Acest lucru însemnând faptul că se trimite o cerere către server, astfel creându-se un canal la care aplicația se abonează (după cum se poate observa în figura 19 acest lucru se realizează folosind cuvântul cheie “subscribe”), iar în timpul în care cererea se soluționează, aplicația își continuă funcționarea obișnuită. În momentul în care cererea este rezolvată, serverul trimite un răspuns înapoi pe canalul creat. Doar la primirea răspunsului aplicația alocă resurse pentru prelucrarea datelor.

Figura 19 - Exemplu cerere ce nu blochează funcționare programului în așteptarea răspunsului

În exemplul de mai sus este prezentată funcția ce comunică serverului o schimbare a stării unui dispozitiv electronic realizată de către utilizator în cadrul interfeței. În acest caz metoda “showConfirmationOnInterface” nu va fi executată decât în momentul în care un răspuns va ajunge înapoi de la server, iar în timpul în care cererea este rezolvată programul își continuă funcționarea normală. În acest caz timpul de rezolvare a cererii este de ordinul milisecundelor deoarece acțiunea necesară a fi completată este una ce nu necesită transportul sau prelucrarea unui volum mare de date. Cererile ce așteaptă ca rezultat un volum mare de date vor avea mici întârzieri (insesizabile în cadrul interfeței) datorate vitezei de internet sau chiar a capacității reduse de procesare a serverului.

Angular este renumit pentru faptul că detectează, aproape instantaneu, orice modificare

a interfeței. Acest lucru permite o mai bună coordonare a evenimentelor ce se declanșează în urmă unor modificări aduse interfeței. Orice modificare, indiferent cât de nesemnificativă, poate declanșa un eveniment atâta timp cât programatorul oferă instrucțiuni explicite framework-ului în vederea ascultării modificărilor. Una din funcțiile principale ale aplicației web este comandarea dispozitivelor electronice, utilizatorul acționând comutatoarele destinate acestor acțiuni din interfața grafică. În momentul în care este detectată schimbarea stării unui comutator se trimite o cerere te tip POST către server în care se specifică faptul că acel dispozitiv trebuie să își schimbe starea.

1. Modulul Socket.

Are rolul de a recepționa notificări de tipul “push” trimise de către aplicația server fără a trimite în

prealabil o cerere în vederea obținerii acestora. Așa cum am menționat și la modulul socket al

aplicației server, între aplicația UI și server se creează în prealabil o conexiune la încărcarea paginii.

În momentul în care acea conexiune a fost realizată, în canalul creat serverul poate introduce date

fără că acestea să fie explicit cerute de către aplicația UI. Ascultând continuu canalul, la primirea

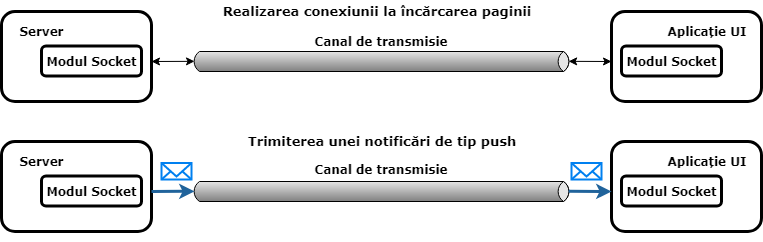
unui mesaj modulul socket îl transmite mai departe către interfață web pentru a fi afișat.

Figura 20 - Exemplu creare canal și trimitere notificare între modulele socket ale aplicației UI și server

Prin acest modul se primesc notificări ale unor schimbări ale configurației alimentării circuitelor

ce rezultă în urma prelucrării datelor de către modulul CEP din cadrul serverului și actualizări

ale puterii consumate dar și starea actuală (pornit/oprit) a fiecărui dispozitiv în parte.

1. Modului Interfață UI.

Funcționalitatea principală a modulului interfață este conectarea utilizatorului cu dispozitivele electronice, oferindu-i posibilitatea de a vizualiza în timp real informații despre modul de conectare, starea și locul unde este amplasat dispozitivul în interiorul casei. Inițial, pentru a putea avea acces la aceste informații, utilizatorul, la o prima accesare a aplicației web, este direcționat către o pagină în care i se cere să completeze un formular cu datele personale, ce includ: nume, prenume, data nașterii, dar și id-ul dispozitivului electronic folosit pentru monitorizarea senzorilor, prizelor și întrerupătoarelor.

După completarea formularului, prin apăsarea butonului “înregistrare”, datele sunt transferate către server printr-o cerere de tip POST. În momentul ajungerii datelor la server, acesta creează contul și îi asociază dispozitivul ce este deja conectat și trimite date către server. După ce contul a fost creat, utilizatorul poate intra în cont introducând numele de utilizator și parola anterior create. După primirea confirmării faptului că acel utilizator există în baza de date din partea serverului, utilizatorul este redirecționat către pagină principala a aplicației. Interfața web este constituită din 3 pagini create folosind limbajele HTML [19], CSS [20] și Typescript [21].

Dat fiind faptul că în crearea aplicației a fost folosind framework-ul Angular, structura acesteia este bazată pe componente. Avantajul acestui lucru este faptul că în momentul în care o componentă este modificată, rezultatul modificării nu va afecta și restul aplicației. Un alt avantaj este dat de faptul că la schimbarea paginii, aceasta nu trebuie mereu reîncărcată, anumite componente ce sunt neesențiale vor dispărea iar altele, ce sunt necesare paginii dorite, vor apărea.

În cazul acestei aplicații există două componente ce sunt prezente în fiecare pagină :

1)    Oferă o modalitate de navigare a aplicației. Prin apăsarea butoanelor, anumite componente vor dispărea pentru a face loc celor specifice paginii selectate.

2)  Conține butonul de ieșire din aplicație (deconectare), determinând ștergerea semnăturii utilizatorului (token-ul generat în momentul autentificării) din stocarea locală și lista tuturor tipurilor de monezi existente și valoarea lor exactă, utilizatorul putând selecta din această lista pe cea în care dorește să îi fie afișate valorile monetare din cadrul aplicației. Tipul monedei poate fi schimbat oricând pe parcursul navigării interfeței. Pentru obținerea cursului valutar este folosit un API ce la intrarea în aplicație obține valoarea fiecărei monezi existente în funcție de valoarea leului.

Prima pagină are în alcătuire cinci componente, fiecare lucrând independent față de celelalte:

a)    Utilizatorul își poate vedea o parte din datele introduse în momentul în care s-a înregistrat, aceste date fiind preluate din baza de date în momentul în care autentificarea utilizatorului a rezultat într-un răspuns afirmativ. În cazul în care utilizatorul a decis să își adauge și o fotografie de profil (încărcarea pozei putând fi realizată prin accesarea paginii setări), aceasta va fi prezentă împreună cu datele menționate anterior, iar în caz contrar o fotografie implicită va ocupă acel loc.

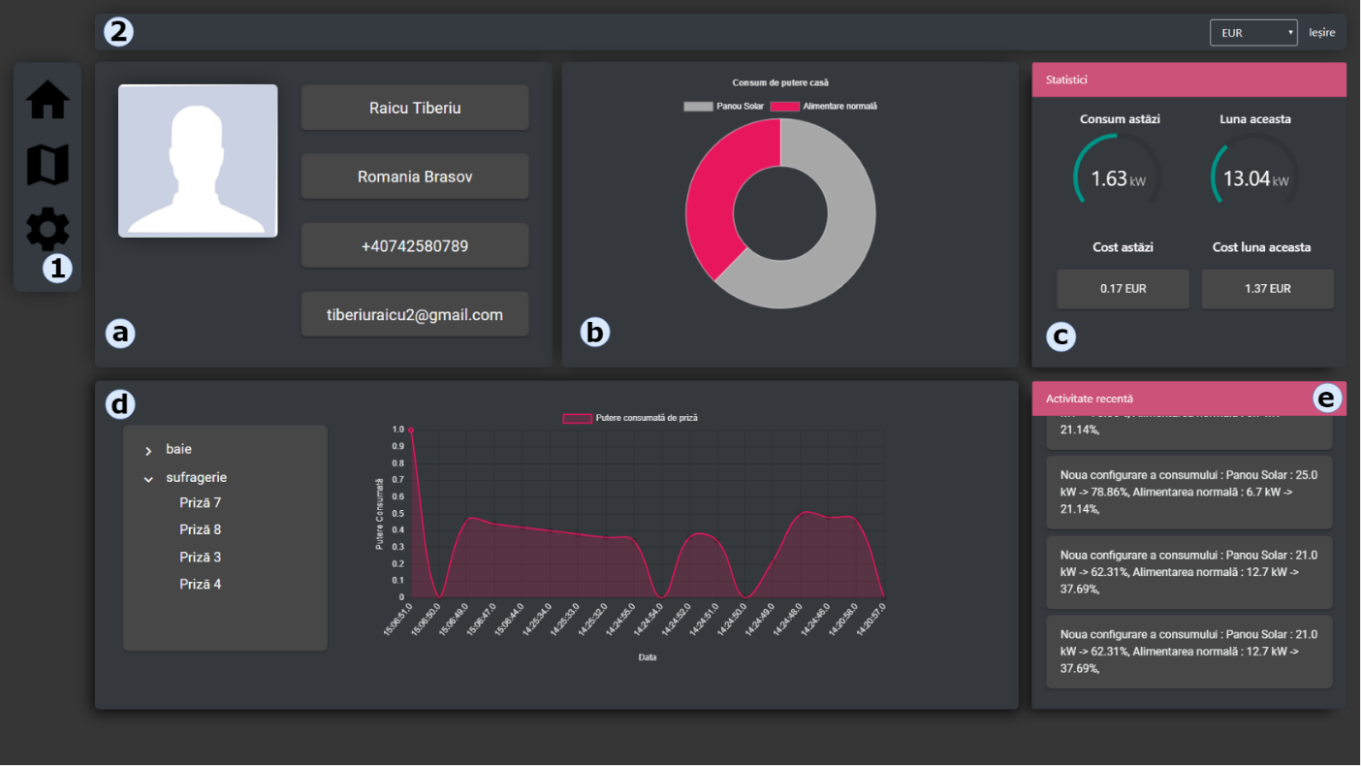
b)    Conține un grafic în care este prezentat procentul în care panoul solar acoperă consumul de energie al dispozitivelor electronice ce sunt alimentate de la circuitele casei (fiind reprezentat de culoarea gri). După cum se poate observa, un panou solar nu este suficient pentru a alimenta o întreagă casă de aceea este necesară o sursă secundara de energie. Culoarea roșu reprezintă restul energiei necesare alimentării circuitelor rămase în urma împărțirii optime a curentului produs de panoul solar. Curentul necesar completării deficitului de energie provenind de la alimentarea normală.

Figura 21 - Prima pagină a aplicației web

c)    Are în componență două grafice ce prezintă utilizatorului consumul de energie din ziua curentă, dar și consumul total din acea lună, cu costurile aferente într-o monedă ce poate fi selectată din meniul aflat în partea de sus a paginii. La încărcarea paginii se trimite o cerere de tip GET către server în care se cere efectuarea analizei consumului pentru ziua și luna curentă. Serverul preia din baza de date toate informațiile legate de consum ce coincid cu ziua și luna cerută, iar după procesarea datelor un răspuns în format JSON este trimis înapoi către aplicația Web. Odată ajuns, din corpul răspunsului se extrag datele necesare, urmând a fi trimise către modulul UI pentru a fi afișate.

d)    Prima componentă conținută în jumătatea inferioară a paginii are în componență un meniu în care se găsesc locațiile casei ce conțin consumatori inteligenți dar și consumatorii existenți în fiecare dintre acestea. Prin selectarea unui consumator se deschide un nou canal între modulul socket al aplicației Web și modulul socket al serverului. Serverul începe a trimite prin acel canal datele primite de la aplicația de monitorizare și control ce fac referire la dispozitivul electronic selectat. În momentul primirii unei date (la interval de o secundă), aceasta va fi adăugată într-un grafic, utilizatorul putând viziona în timp real tendința consumului de energie al acelui dispozitiv.

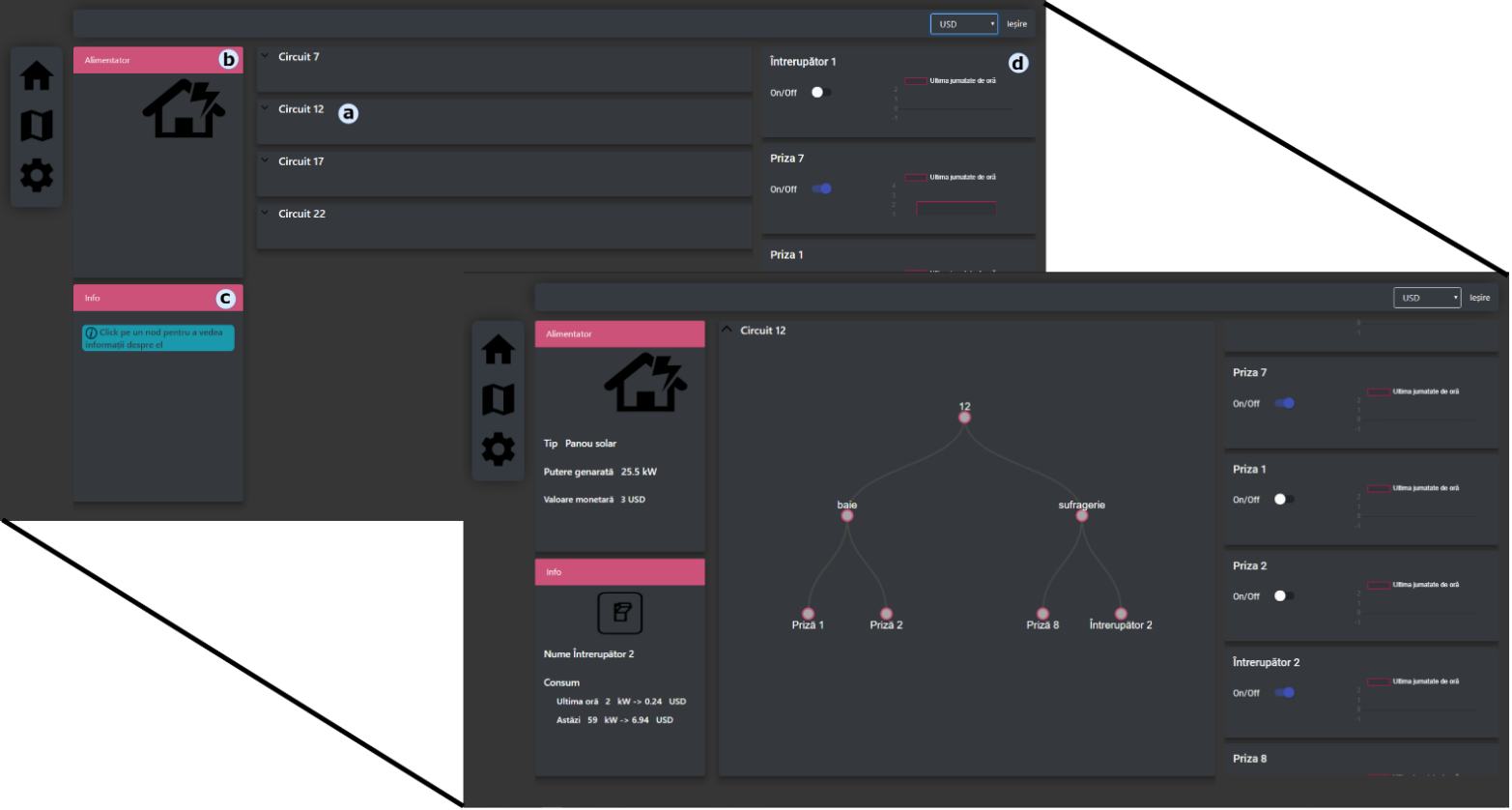
e) Ține utilizatorul la curent cu evenimentele produse în mod automat. Conține o listă de carduri în care se adaugă în mod continuu informații referitoare la activitatea celor trei scenarii implementate în proiect. În momentul generării unui eveniment de către un scenariu CEP, se trimite pe canalul creat în prealabil între cele doua aplicații o notificare ce informează utilizatorul cu privire la acțiunea automată ce tocmai s-a produs. La recepționarea notificării, mesajul acesteia este încorporat într-un card ce este introdus în lista conținută de această componentă.

Figura 22 - A doua pagină a aplicației web

Ca și în cazul paginii anterioare se poate observa existența celor două meniuri fixe. Din punct de vedere al componentelor specifice acestei pagini se poate remarca faptul că sunt în număr de patru:

a)    Are în componentă o serie de carduri ce reprezintă circuitele ce alcătuiesc instalația electrică a casei. La selectarea unuia dintre carduri acesta se va redimensiona ocupând toată suprafață componentei. În timpul realizării animației o cerere de tip POST este trimisă către server cerând datele dispozitivelor electronice ce sunt alimentate de circuitul respectiv. În momentul terminării animației (în cazul unei viteze optime a internetului), datele solicitate în format JSON vor fi ajuns la aplicația Web. După redimensionare, în corpul cardului va apărea o harta pe trei nivele (eng. tree) în care se vor regăsi toate dispozitivele electronice ce fac parte din circuitul selectat. Primul nivel este reprezentat de numărul de identificare a circuitului prin care poate fi găsit în baza de date, iar al doilea conține camerele a căror dispozitive electronice sunt alimentate de acest circuit. Al treilea nivel are în componență dispozitivele electronice alimentate de circuit, fiecare nod dispozitiv fiind legat de nodul cameră din care face parte.

b)    La selectarea unui circuit, acest modul va oferi utilizatorului informații despre modul în care circuitului îi este furnizat curent (putând proveni de la unul sau mai multe panouri solare sau de la alimentarea normală). Tot în această componentă putem regăsi puterea generată de sursa de alimentare (valoarea exactă în cazul panoului solar, alimentarea normală fiind o sursă infinită) și cât valorează aceasta (acest câmp fiind omis în cazul alimentării normale). Toate aceste date sunt încapsulate în răspunsul la cererea de tip POST trimisă anterior către aplicația server la selectarea circuitului.

c)    La selectarea unui dispozitiv din harta unui circuit, câmpul „Info” va afișa o imagine specifică însoțită de puterea consumată în ziua curentă, dar și de cea din ultima ora. În funcție de moneda selectată din meniul fix aflat în partea superioară a paginii, consumul va fi tradus și în suma de bani ce va trebui plătită.

d)    Are în componență o serie de carduri ce reprezintă fiecare consumator existent în casă. Din aceste subcomponente este posibilă controlarea stării dispozitivului (pornit/oprit). Starea de oprit/pornit este dată de un comutator ce este conținut în cardul ce reprezintă dispozitivul. Pentru valorile ce nu îndeplinesc standardele scenariilor implementate în aplicația server, în jurul comutatorului acest lucru va fi semnalat prin apariția unui puls de culoare roșie. În cazul în care dispozitivul este oprit, acest lucru se va putea observă și în cadrul interfeței prin schimbarea în mod automat a stării comutatorului. Pe lângă o modalitate de controlare, cardul conține și un grafic ce reprezintă consumul dispozitivului în ultima jumătate de ora.

Ce de-a treia pagină a aplicației web conține, pe lângă meniurile fixe anterior menționate și o

serie de câmpuri ce oferă utilizatorului flexibilitatea schimbării datelor anterior introduse în

procesul de înregistrare, dar și adăugarea unor date noi precum adăugarea unei fotografii de

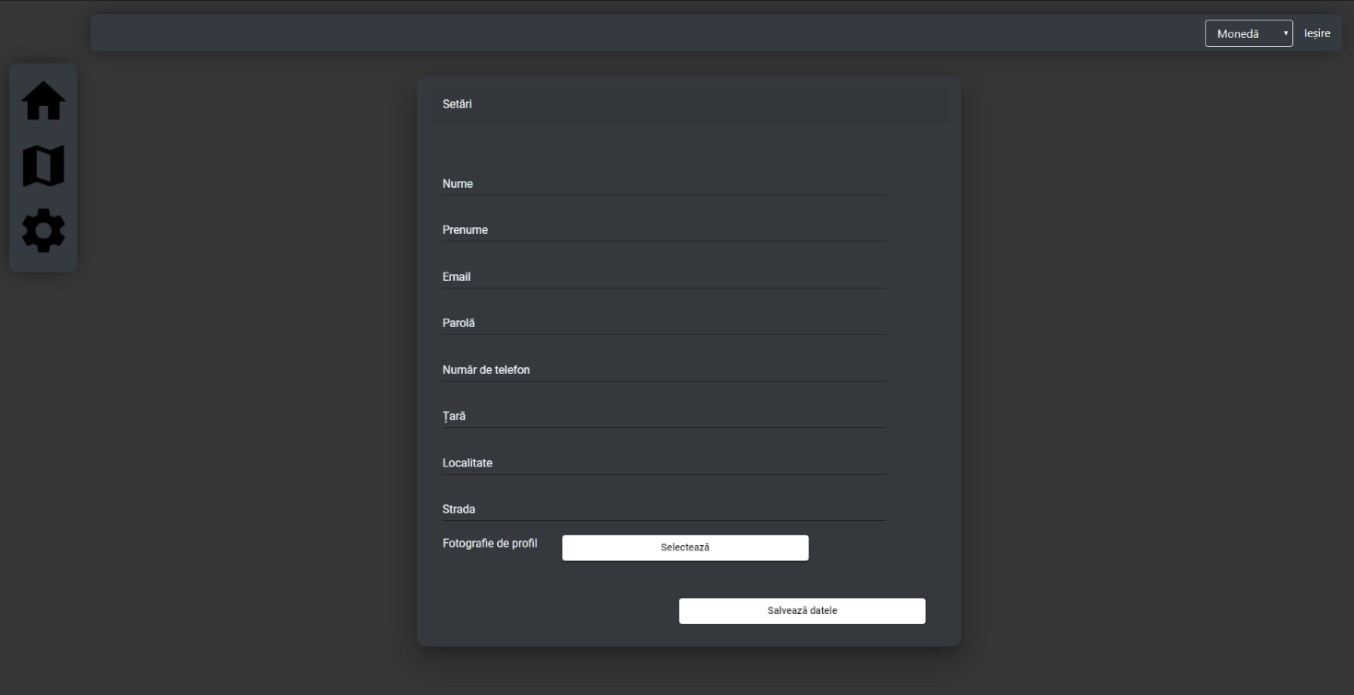
profil ce va fi prezentă pe prima pagină a aplicației.

Figura 23 - Pagina de setări a aplicației web

### 2.2.2 Modul de interacțiune a componentelor software bazat pe scenarii

* Scenariul unu

Primul scenariu, așa cum a fost explicat și anterior, face referire la economisirea puterii consumate de la o priză atunci când la aceasta este conectat un încărcător care nu este folosit pentru o perioadă îndelungată. În momentul în care un eveniment a fost declanșat în urma îndeplinirii condițiilor scenariului, modulul CEP va notifica alte două module transmițându-le informațiile necesare îndeplinirii sarcinii pentru care au fost notificate. Cele două module sunt : modulul Socket și modulul de expediere instrucțiuni.

Modulul Socket preia informația primită, cea de oprire a alimentării unei anumite prize și o plasează pe canalul creat în urma accesării de către utilizator a aplicației web folosind numele de utilizator și parolă. Aplicația web, ascultând continuu canalul creat între aceasta și server, la primirea unei informații, o redirecționează către interfață UI unde va fi afișată în secțiunea de notificări. În același timp, această informație este folosită și la actualizarea stării prizei ce este vizibilă în cadrul interfeței.

Modulul de expediere preia instrucțiunea primită, trimițând-o prin intermediul serviciului de mesaje numit RabbitMQ către aplicația de monitorizare și control ce rulează în cadrul unui Raspberry Pi. Aceastăa redirecționează instrucțiunea către aplicația ce simulează senzorii. Odată ajunsă acolo, această instrucțiune determină schimbarea unui fișier de configurare ce conține starea actuală a fiecărei prize.

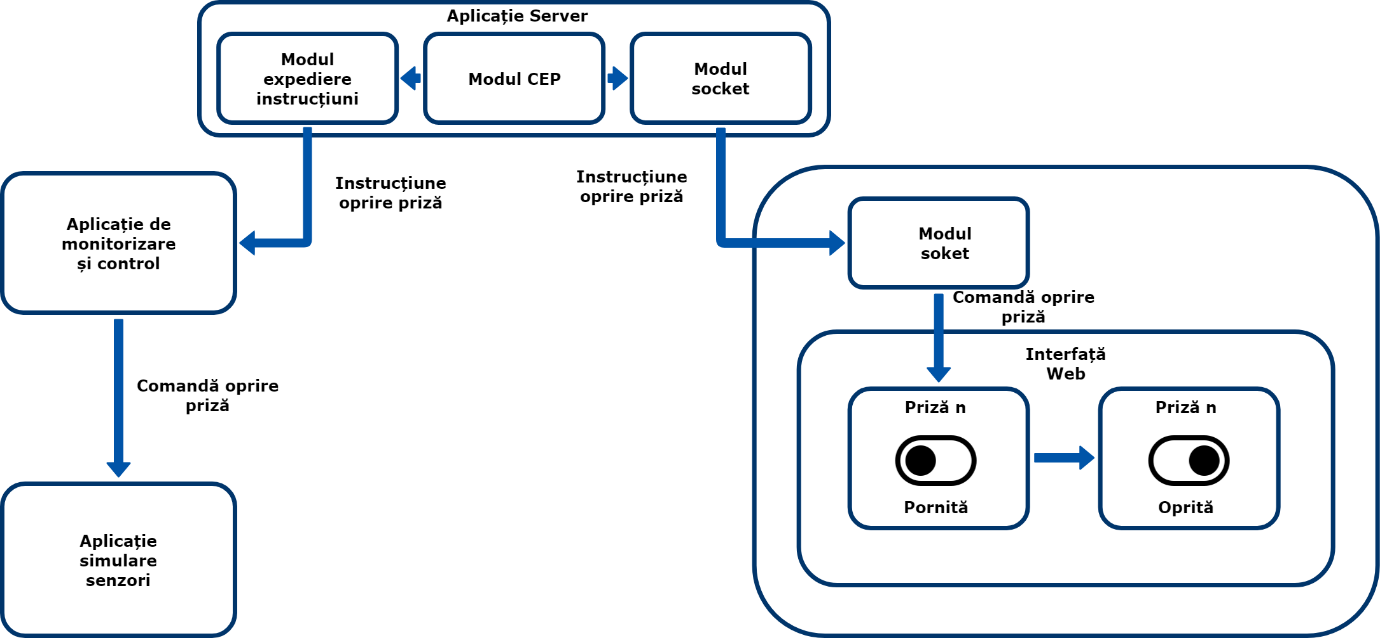


Figura 24 – Interacțiunea componentelor software ce sunt puse în mișcare de activarea primului scenariu

• Scenariul doi

Al doilea scenariu face referire la corelarea informației generate de doi senzori (utilizați pentru detecția mișcării și a stării întrerupătorului) în vederea economisirii puterii consumate de sistemul de iluminare prin detecția situației în care în cameră nu există mișcare. Din punct de vedere al modului de lucru acest scenariu este asemănător celui anterior. La declanșarea unui eveniment se trimite către modulele de transmisie informațiile necesare, iar acestea le vor trimite mai departe către cele două aplicații. Dacă în scenariul anterior dispozitivele afectate erau prizele, în acest caz, instrucțiunile trimise au ca și scop starea întrerupătoarelor. In aplicația Web, comutatorul (ce semnifică starea de pornit/oprit a întrerupătoarelor) se va schimba din pornit în oprit.

Modulul de expediere preia instrucțiunea primită, o trimite prin intermediul serviciului de mesaje numit RabbitMQ către aplicația de monitorizare și control, de aici fiind redirecționată către aplicația ce simulează senzorii. Ajunsă acolo, instrucțiunea determină schimbarea unui fișier de configurare ce conține starea actuală a fiecărui întrerupător.

• Scenariul trei

Al treilea scenariu permite detectarea unei creșteri bruște a consumului de la o priză și permite recalcularea modului de alimentare a circuitelor (de exemplu : panoul solar fotovoltaic, alimentarea normală) prioritizând utilizarea energiei verzi produsă de panourile solare. La declanșarea evenimentului, modulul CEP este notificat în vederea verificării existenței unei cantități suficiente de energie pentru a satisface creșterea bruscă a consumului. O cerere este trimisă de către acest modul către cel de baze de date în care este solicitată actualizarea consumului la acea priză. După efectuarea cererii anterioare, o altă cerere este trimisă în care se cere consumul total al circuitului din care acea priză face parte. În cazul în care circuitul este în prezent alimentat de un panou solar, se calculează noua putere necesară alimentării tuturor circuitelor alimentate de acel panou, iar în cazul în care puterea generată are o valoare mai mică față de cea consumată se trimite o cerere către baza de date solicitând toate circuitele existente în locuința. Apoi se găsește cea mai bună posibilitate de aranjare a acestora astfel încât puterea generată de panoul solar să fie folosită cât mai optim, alegând combinația de circuite ce se apropie cât mai mult de această putere generată. Circuitele rămase, cele ce au fost inițial alimentate de energia panoului solar își vor schimba alimentarea la cea normală. După terminarea creării noii configurații, aceasta se trimite către modulul de baze de date care va trimite o cerere către baza de date ce solicită modificarea vechii configurații a alimentării circuitelor.

Instrucțiunea de schimbare a configurării alimentării circuitelor nu este trimisă doar modulului de baze de date, ci și modulului socket și modulului de transmitere a instrucțiunilor. Modulul socket preia informația și o trimite către aplicația web prin canalul deja existent. În momentul în care datele ce conțin noua configurație a alimentării circuitelor ajung la aplicația web, acestea sunt afișate într-un panou sub formă de notificare.

Modulul de expediere preia instrucțiunea primită, o trimite prin intermediul RabbitMQ către aplicația de monitorizare și control, de aici fiind redirecționată către aplicația de simulare a senzorilor. Ajunsă acolo, instrucțiunea determină schimbarea unui fișier de configurare ce conține modul de alimentare a fiecărui circuit.

# Tehnologii utilizate

## 3.1 Java

### 3.1.1 Scurtă caracterizare

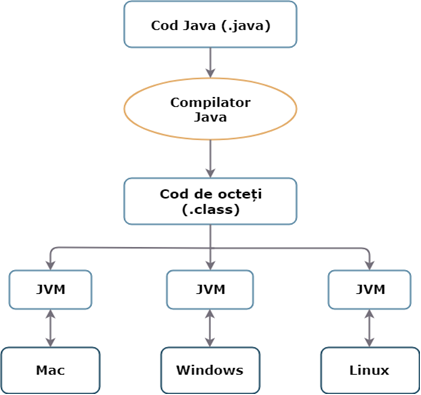
Inițial denumirea acestui limbaj a fost “Oak” în anul 1991, care a fost creat pentru a scrie aplicații ce urmau a fi folosite în domeniul electronic . Mai târziu, în anul 1995  numele său a fost schimbat în Java. Java a fost creat de către James Gosling, la Sun Microsystem. Oak a fost reproiectat în 1995 sub numele de Java pentru a permite crearea de aplicații ce pot rula în cadrul internetului. Folosind Java, programele create pot fi încorporate în pagini HTML. Acest limbaj nu este limitat doar la aplicații web, prin acest limbaj putând fi create o multitudine de alte tipuri de aplicații. Unul din punctele forte ale acestui limbaj îl reprezintă faptul că poate rula pe orice platforma, sloganul acesteia fiind “Write once, run anywhere (WORA)”. Această facilitate este posibilă prin folosirea mașinii virtuale java (eng. Java Virtual Machine - JVM). Când un program este scris în Java, compilatorul transformă codul într-un fișier ”.class” sau într-un format numit cod de octeți și nu în codul nativ al mașinii. Codul de octeți genera teste, nu cod neexecutabil având nevoie de un traducător pentru a putea fi executat pe o anumită mașină. Acest traducător este JVM și prin urmare codul de octeți este executat de JVM

Figura 25 - Modul de funcționare JVM (Java Virtual Machine)

### 3.1.2 Caracteristici Java :

1. Simplu – limbajul Java este mai simplu față de limbajele anterioare (C sau C++). Java a eliminat conceptul de pointer ce era anterior prezent în C și C++ din motive de securitate, conceptul de pointer în limbajul C fiind adesea folosit în mod greșit, acesta fiind unul din principalele motive pentru care programatorii consideră limbajul C ca fiind un limbaj de programare complicat. Creatorii Java nu au dorit să introducă această complexitate în limbajul lor de programare și au exclus în mod intenționat conceptul de pointer. Această decizie a fost luată deoarece existența pointerilor într-un program poate crea multe bug-uri în cod. Printre proprietățile acestui limbaj se numără și alocarea automată de memorie și colectorul de gunoi (eng. Garbage Collector) , ceea ce în C/C++ trebuie realizat de către programator.
2. Orientat pe obiecte – majoritatea limbajelor de programare sunt de tip procedural. Java este un limbaj de programare orientat pe obiecte deoarece folosește conceptul de obiect. În Java totul depinde de obiecte (crearea obiectelor și găsirea de modalități pentru a le face să lucreze împreună). Din cauza faptului că Java este un limbaj orientat pe obiecte permite un nivel foarte mare de modularitate, reflosire a codului și flexibilitate.
3. Distribuit – Java implementează interfețe de comunicație, ca și protocoale de internet, ftp și http pentru a avea acces la fișiere într-o rețea. În acest fel, folosind biblioteci, se poate face cu ușurință transferul fișierelor într-o rețea.
4. Securitate – este unul din cele mai securizate limbaje de programare existente la momentul actual deoarece : conceptul de pointeri este eliminat, iar din acest motiv datele nu pot fi corupte, iar locațiile din memorie nu pot fi rescrise; suportă tratarea excepțiilor, forțând programatorul să scrie cod pentru excepții, care este posibil să apară în timpul execuției programului. În acest fel programul să poată fi terminat fără erori care să oprească executarea șirului de instrucțiuni.
5. Face diferența între literele mari și mici (case sensitive)
6. Multithread – permite executarea mai multor instrucțiuni în același timp.

### 3.1.3 Principiile OOP (eng. Object-oriented programming) în Java

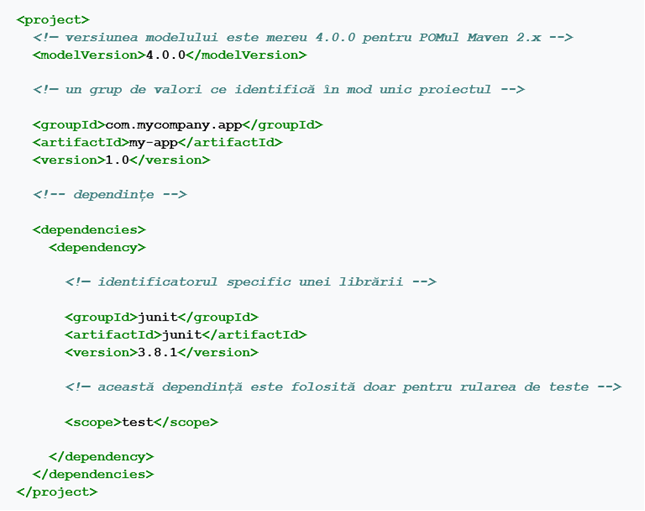
1. Încapsularea - ne permite să protejăm datele dintr-o clasa nepermițând accesarea lor directă. În Java acest principiu poate fi implementat prin adăugarea modificatorului de acces „private” variabilelor componente și crearea unor metode publice, numite getter și setter, prin care se obțin și se setează aceste valori.
2. Abstractizarea – are scopul de a ascunde complexitatea și de a arăta utilizatorilor doar informația relevantă. Spre exemplu: dacă se utilizează un dispozitiv, nu este nevoie să știm cum funcționează mecanismul interior. Utilizatorul trebuie să știe doar cum să manevreze acel dispozitiv prin interfața oferită. În acest caz cunoștințele utilizatorului în ceea ce privește dispozitivul sunt abstracte. În programare, abstractizarea este definită ca lucruri simple, obiecte, clase și variabile, ce reprezintă un cod mult mai complex.
3. Moștenirea – este o caracteristică specială a programării orientate pe obiect în Java. Permite programatorilor să creeze noi clase care să conțină anumite atribute ale unor clase deja existente. Acestă caracteristică ne permite să ne bazăm pe munca anterioară fără a reinventa roată.
4. Polimorfismul – poate fi tradus ca “mai multe forme”  și apare atunci când există mai multe clase ce sunt legate între ele prin moștenire.

Așa cum am specificat la punctul anterior, moștenirea ne permite să folosim metode și atribute dintr-o altă clasa. Polimorfismul folosește aceste metode pentru a îndeplini diferite sarcini.

### 3.1.4 Apache Maven [22]

Maven este un sistem de “[build](https://ro.wikipedia.org/w/index.php?title=Build&action=edit&redlink=1" \o "Build — pagină inexistentă)” și administrare a proiectelor, scris în [Java](https://ro.wikipedia.org/wiki/Java).

Maven abordează două aspecte legate de crearea unui program : în primul rând descrie modul în care acesta este construit și în al doilea rând, descrie dependințele acestuia. Proiectul Maven este descris de un fișier XML (eng. Extensible Markup Language) [23] denumit POM (eng. Project Object Model). Acest fișier conține informații despre modulele proiectului, precum și dependințele acestuia (alte proiecte). Ordinea operațiunilor este definită prin declararea unor pluginuri, din cadrul cărora unele goaluri sunt setate și configurate în diferite faze predefinite din ciclul de viață al unui build. Bibliotecile Java și pluginurile, sunt descărcate automat de către Maven din unul sau mai multe repository-uri și le stochează în cache-ul local.

Exemplu :

Caracteristicile principale ale Maven :

* Conține o gamă largă de librării.
* Permite gestionarea mai multor proiecte simultan.
* Bibliotecile și plugin-urile Java necesare sunt descărcate automat din depozitele Maven.
* Actualizarea automată a dependințelor.
* Acces instant la noi feauture-uri cu foarte puțină sau chiar zero configurare.

## 3.2 Spring Framework [24]

### 3.2.1 Caracterizare

Este cea mai populară platformă pentru crearea de aplicații Java. Caracteristicile principale ale acestui Framework pot fi folosite pentru a scrie aplicații Java, dar aceasta conține și extensii ce permit și dezvoltarea de aplicații web.

### 3.2.2 Avantajele Spring Framework :

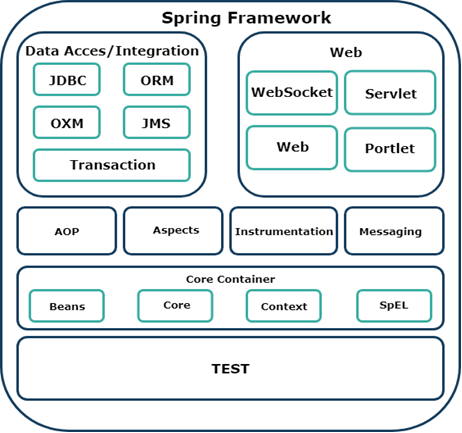
1. Organizare modulară. Chiar dacă numărul claselor și pachetelor este substanțial, programatorul folosește doar ceea ce îi este necesar, ignorând restul modulelor.
2. Spring nu reinventează roata, în schimb folosește multe din tehnologiile deja existente cum ar fi :  câteva frameworkuri ORM (eng. Object Relațional Mapping), frameworkuri de logare a datelor, JEE (eng. Java Platform, Enterprise Edition), etc.

Figura 26 - Arhitectura Spring Framework

1. O aplicație scrisă cu ajutorul Spring este ușor de testat deoarece codul dependent de mediul înconjurător este mutat în acest cadru. Prin folosirea POJO-urilor (eng. Plain Old Java Object), injectarea dependințelor devine mai ușor de utilizat.
2. Frameworkul de web este foarte bine conceput.
3. Spring oferă un API potrivit pentru traducerea excepțiilor specific tehnologiei (aruncate de Hibernate, JDO sau JDBC, de exemplu) în excepții consecvente și necontrolate.
4. Nivel de abstractizare JDBC ajută utilizatorii să gestioneze erorile într-o manieră ușoară și eficientă. Codul JDBC poate fi redus substanțial atunci când acest nivel de abstractizare este implementat într-o aplicație Web

### 3.2.3 Injectarea Dependințelor

Injectarea Dependințelor este cea mai cunoscută tehnologie din Spring Framework a Inversiunii de Control. Inversiunea de Control (IoC) este un concept general și poate fi exprimat în mai multe moduri diferite. Injectarea Dependințelor este doar un exemplu concret de Inversiune de Control.

La scrierea unei aplicații complexe trebuie avut în vedere crearea claselor astfel încât acestea să fie cât mai independente una față de cealaltă. Acest lucru mărește posibilitatea reutilizării claselor.

Dependența se traduce ca fiind o asociere între două clase. Spre exemplu o clasă B este dependentă de clasa A. Acum uitandu-ne la cel de-al doilea cuvant, injectarea, poate fi tradus prin: clasa A va fi injectată în clasa B.

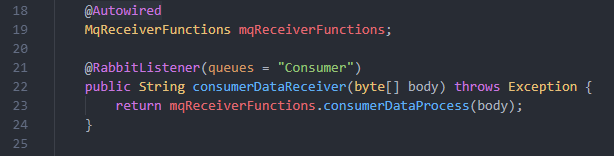


Figura 27 - Exemplu injectare de dependințe

Injectarea Dependințelor poate fi realizată în momentul transmiterii parametrilor către constructor sau prin post-construcție folosind metode de setare.

### 3.2.4 Spring Boot

Spring Boot este un mod de simplificare a creării unei aplicații independente, având nevoie de configurații minime sau chiar zero. Știe să își aleagă singur dependințele și să își configureze automat caracteristicile pe care utilizatorul dorește să le folosească, aplicația putând fi pornită doar printr-un singur click.

Avantaje ale utilizării Spring Boot :

* Spring Boot crește productivitatea și scade timpul dezvoltării aplicației.
* Conține server HTTP încorporat, cum ar fi Jetty, Tomcat etc. Pentru dezvoltarea și testarea aplicațiilor web.
* Spring Boot oferă o multitudine de plugin-uri pentru dezvoltarea și testarea aplicațiilor Spring într-un mod foarte ușor utilizând instrumente de construire cum ar fi Gradle și Maven.
* Acesta oferă o multitudine de plugin-uri pentru lucrul cu bazele de date din memorie sau cele încorporate.

## 3.3 RabbitMQ

RabbitMQ este un program de așteptare a mesajelor (eng. message-queueing) numit broker de mesaje. Mai simplu spus, este un program unde pot fi definite cozi, iar aplicațiile se pot conecta la acestea pentru a trimite mesaje.

Un mesaj poate include orice tip de informație. Poate, de exemplu să conțină informații despre un proces/activitate care ar trebui să înceapă într-o altă aplicație (care ar putea exista pe un alt server), sau ar putea fi un simplu mesaj text. Managerul de coadă (eng. queue-manager) stochează mesajele până când aplicația căreia îi este destinat mesajul se conectează la coadă și îl preia.

Broker-ul de mesaje poate acționa ca intermediar pentru diverse servicii. Acesta poate fi folosit pentru a reduce sarcinile și timpul de livrare de către serverele de aplicații web, deoarece sarcinile, care în mod normal ar necesita destul de mult timp pentru a fi executate, pot fi delegate unei terțe părți al cărei singur scop este de a le îndeplini.

Arhitectura de bază a unei cozi de mesaje (eng. Message queue) este relativ simplă. Există aplicații client numite producători care creează mesaje și le trimit către broker (coadă de mesaje). Alte aplicații, numite consumatori, se conectează la coadă și se abonează la mesajele din aceasta.

O aplicație poate fi producător, sau consumator, sau amândouă, un consumator și un producător al mesajelor. Mesajele inserate în coadă sunt stocate până când consumatorul le preia.

*Când și de ce ar trebui folosit RabbitMQ?*

Această metodă de trimitere a mesajelor permite aplicațiilor de tip server să răspundă într-un timp mai scurt la cereri (eng. request) în loc să fie forțate să efectueze operații, ce necesită multe resurse, exact în acel moment. O altă situație în care această modalitate de transfer poate fi folosită este în momentul în care un mesaj trebuie trimis mai multor destinatari .

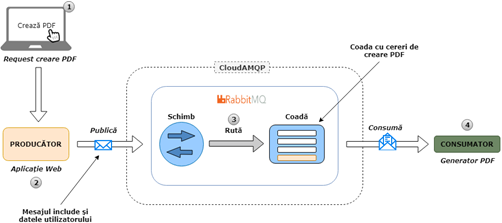
Consumatorul poate prelua mesajul din coadă și să înceapă prelucrarea unui PDF, iar în același timp producătorul să publice alte noi mesaje în coadă. Consumatorul poate fi pe un cu totul alt server față de producător sau chiar pe același server. Cererea poate fi creată într-un limbaj de programare și prelucrată de altul – cele două aplicații vor comunica doar prin mesajele ce sunt trimise între ele.

Figura 28 - Exemplu aplicație RabbitMQ

1. Utilizatorul trimite o cerere pentru crearea unui document PDF către aplicația web.
2. Aplicația web (producătorul) trimite un mesaj către RabbitMQ conținând datele cererii cum ar fi numele și e-mailul.
3. Un schimb (eng. exchange) acceptă mesajul de la aplicația producător, trimițând-o către coada specifică generatorului PDF.

4.    Generatorul PDF primește cererea și începe crearea documentului.

***Exchanges***

Mesajele nu sunt publicate direct într-o coadă, în schimb, producătorul trimite mesajul către un schimb (eng. exchange). Un exchange are datoria de a ruta mesajele către diferite cozi. Un exchange acceptă mesaje de la aplicația producător și le rutează către cozi cu ajutorul legăturilor și a cheilor de rutare.

Fluxul de mesaje în RabbitMQ :

1. Producătorul trimite mesajul către un exchange.

2. Mesajul este primit de către exchange, acum fiind responsabil de rutarea

corespunzătoare a acestuia.

3.    Sunt create legături între exchange și cozi. În acest caz sunt  create 2 legături între exchange și cele două cozi.

4.    Mesajul rămâne în coadă până când acesta este preluat de consumator.

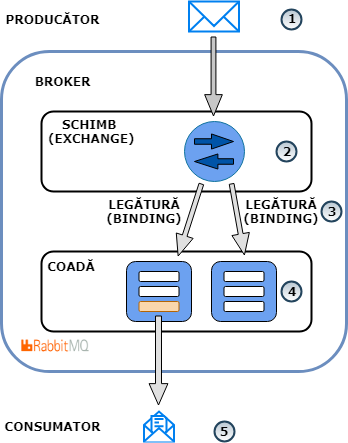
5.    Consumatorul prelucrează mesajul.

Figura 29 - Funcționare exchange în RabbitMQ

## 3.4 CEP (eng. Complex Event Processing)

CEP (eng. Complex Event Processing) sau ESP (eng. Event Stream Processing) sunt tehnologii  folosite în sisteme bazate pe evenimente. Aceste tipuri de evenimente consumă și reacționează la un flux de evenimente în timp real. De obicei acestea sunt tranzacțiile financiare, identificarea fraudelor și sisteme de monitorizare a proceselor – unde trebuie identificat, înțeles și reacționat într-un timp cât mai scurt la anumite lucruri ce apar într-un flux de evenimente.

Un sistem de tip CEP este asemănător cu o baza de date, dar inversat. Într-un sistem de baze de date se stochează date și se execută interogări asupra lor, iar în CEP se stochează interogările, iar datele sunt “trecute” prin acestea.

După cum este menționat în [25] principala sarcină a unui sistem CEP este aceea de a detecta tiparele evenimentelor complexe care prezintă interes în cadrul unui sistem. Detectarea este făcută pe baza unei multitudini de evenimente simple. Un sistem CEP trebuie să filtreze, să separe, să agrege, să transforme și să îmbogățească evenimentele simple semnalate în cadrul sistemului. În urma acestor operații se generează evenimente complexe noi.

În figura 30 sunt prezentate blocurile componente ale unui sistem CEP. Pentru a putea face posibilă procesarea evenimentelor în mod automat este necesar un mecanism bazat pe meta-date. Modelul evenimentelor definește tipurile evenimentelor întâlnite în cadrul sistemului. Regulile de procesare

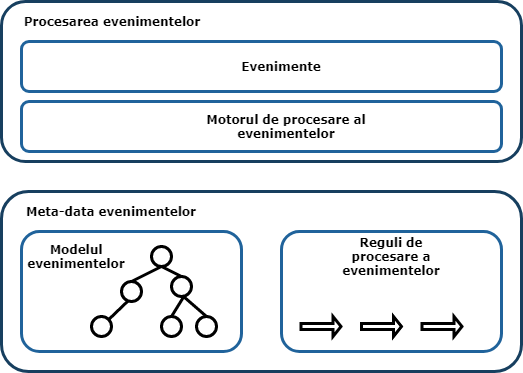
ale evenimentelor definesc corelațiile dintre evenimente sub forma modelelor evenimentelor, și determină acțiunile care trebuie luate.

Figura 30 - Funcționare exchange în RabbitMQ

### Modelul evenimentelor

Un eveniment simplu reprezintă o schimbare produsă în cadrul unui sistem la un anumit moment de timp. Acest moment de timp devine un atribut al evenimentului. Mai multe evenimente simple pot compune un eveniment complex/compus prin utilizarea anumitor operatori. Acești operatori definesc relația evenimentelor simple în cadrul unui eveniment complex. În cazul în care și evenimentele complexe compun la rândul lor altele, se produce un nou nivel al evenimentelor complexe. Acestea nu au ca atribut un singur moment de timp, ci un interval de timp [t1, t2]. Acest interval reprezintă perioada de timp în care s-a produs primul și ultimul eveniment care a dus la compunerea evenimentului complex .

## 3.5 Javascript

Înainte de inventarea acestui limbaj, site-urile web aveau un comportament de tip static, acest lucru însemnând că erau limitate în funcțiile și facilitățile pe care le puteau oferi. Utilizatorii nu puteau interacționa cu pagina web ci puteau doar vizualiza conținutul acesteia care, în mare parte, era compus doar din text. Apariția JavaScript a permis utilizatorilor să interacționeze și să manipuleze anumite părți ale unui site web. În ziua de astăzi JavaScript este folosit în aproape toate site-urile web existente.

Istorie

JavaScript a fost creat de către Brendan Eich în anul 1995 în timpul în care era angajat la compania numită Netscape Communications. Limbajul a fost inspirat de Java, Scheme și Self.

Pentru un timp browser-ul creat de compania Netscape se bucura de dominanță pe piața de browsere. Spre finalul anului 1995, Microsoft a lansat Internet Explorer ceea ce s-a dovedit o adevărată amenințare, acesta începând să fie folosit de din ce în ce mai mulți utilizatori. Netscape a răspuns acestei amenințări printr-un proces de standardizare pentru a opri Microsoft din a prelua controlul limbajului JavaScript și prin crearea unui parteneriat cu compania Sun. Sun a început dezvoltarea limbajului Java în anul 1990 într-o încercare de a crea un limbaj pentru “aplicații inteligente”, care s-a prăbușit în 1994, acesta “punându-și ochii” pe Web ca și platforma de promovare. Acest parteneriat a însemnat faptul că Sun primea dreptul de a folosi browser-ul web al companiei Netscape, iar acesta câștigase un aliat puternic împotriva Microsoft. Ca și reacție, Microsoft a creat propria versiune de JavaScript numită JS Script ce a fost lansată în anul 1996. În 1998 America Online (AOL) a achiziționat Netscape și o dată cu aceasta, limbajul JavaScript. De-a lungul anilor acest limbaj a trecut prin multe schimbări pentru a se adapta cerințelor programelor web, ultima versiune lansată fiind ES2018 ce a fost lansată în iunie 2018.

JavaScript poate fi fololosit pentru :

a)    Afișarea de mesaje utilizatorului, ca parte a unei pagini web sau ca și ferestre de avertizare.

b)    Animarea sau crearea imaginilor în momentul în care o acțiune a fost efectuată (ex : mișcarea mouse-ului pe deasupra pozei).

c)    Detectarea evenimentelor (ex : deplasarea mouse-ului peste o anumită zonă a ecranului)

d)    Trimiterea unei cereri către server, iar informația venită ca răspuns să fie afișată fără ca pagina să fie reîncărcată.

e)    Afișarea sau ascunderea anumitor elemente din pagină în funcție de activitatea utilizatorului.

f)     Folosirea librăriilor sau framework-urilor pentru rezolvarea mai rapidă a unor sarcini.

g)    Detectarea tipului browser-ului și afișarea de conținut diferit pentru în fiecare în parte.

h)    Detectarea plugin-urilor instalate și notificarea utilizatorul în caz că un altul este necesar.

i)      Validarea datelor introduse înainte ca acestea să fie trimise către server (ex : un formular în care se cere adresa de e-mail, se poate verifica dacă textul introdus conține simbolul @).

Caracteristici :

1. JavaScript este interpretat, nu compilat. Unele limbaje de programare trebuie să fie compilate, sau translatate în limbaj mașină înainte de a putea fi executate. JavaScript, pe de altă parte, este un limbaj interpretat : browerul execută, rând pe rând, fiecare linie de cod.
2. JavaScript face diferența între litera mare și cea mică. Orice referință la o funcție sau obiect în JavaScript trebuie să conțină exact textul cu care a fost definit (ex : o funcție numită afisareText() trebuie să fie menționată în program că afisareText() folosind litera “T” și nu “t”).
3. Un comentariu pe o singură linie începe cu bară dublă (//). Pentru a include un comentariu în JavaScript, bara dublă poate fi plasată oriunde în cadrul unui rând iar interpretorul JavaScript va ignora orice caracter aflat după acesta până la sfârșitul liniei. De obicei acest simbol este plasat la începutul rândului.
4. Comentariile ce cuprind mai multe linii încep cu /\* și se termină cu \*/. Interpretorul ignoră textul aflat între aceste două simboluri.
5. JavaScript folosește aceleași reguli de scriere că și în cazul limbajului C unde în numele variabilelor pot fi folosite litere, numere și caracterul underscore (“\_”).

## 3.6 HTML5

HTML (HyperText Markup Language) este limbajul standard pentru crearea paginilor și aplicațiilor web, descriind structura semantică a acestora. Acesta a fost creat de către Sir Tim Berners-Lee spre sfârșitul anului 1991, dar nu a fost lansat oficial decât în anul 1995 sub numele de HTML 2.0. Browserele web primesc documente în format HTML de la un server și le transformă în pagini web multimedia. Elementele HTML sunt blocurile de bază ale unei pagini web. HTML oferă un mijloc de a crea documente structurate prin desemnarea unei semantici structurale pentru text, cum ar fi titluri, paragrafe, liste, linkuri, citate și alte elemente. Elementele HTML sunt delimitate prin etichete, scrise cu paranteze unghiulare. Etichete precum <img /> și <input /> introduc direct conținutul în pagină. Alte etichete precum <p> delimitează și furnizează informații despre textul documentului, acestea putând include și alte etichete ca subelemente.

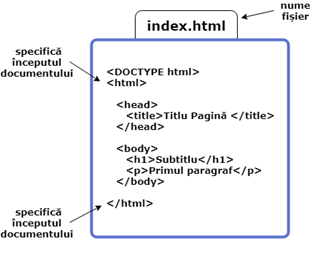
Browserele nu afișează etichetele HTML, ci le folosesc pentru a interpreta conținutul paginii. Codul HTML poate include programe scrise în limbaje de scriptare, cum ar fi JavaScript ce afectează conținutul și comportamentul paginii.

Figura 31 - Exemplu fișier HTML

## 3.7 Node.js [26]

Node.js este un mediu „open-source” și „cross-platform” pentru aplicații scrise în limbajul JavaScript. Este un instrument popular pentru aproape orice tip de proiect.

Node.js folosește motorul JavaScript V8 care este și nucleul Google Chrome [27]. O aplicație Node.js se execută într-un singur proces, fără a crea un fir nou pentru fiecare solicitare. Node.js oferă un set de primitive I / O asincrone în librăria standard care împiedică blocarea codului JavaScript și, în general, librăriile din Node.js sunt scrise folosind paradigme non-blocante, făcând ca blocarea să fie mai degrabă o excepție decât o normă

Când Node.js trebuie să efectueze o operație I / O, cum ar fi citirea de la rețea, accesând o

bază de date sau un sistem de fișiere, în loc de a bloca firul și de a pierde ciclurile procesorului

în așteptare, Node.js va relua operațiunile atunci când răspunsul va reveni. Acest lucru permite Node.js să se ocupe de mii de conexiuni în același timp fără a fi nevoie sa folosească mai multe fire de execuție (eng. thread), care ar putea fi o sursă semnificativă de bug-uri.

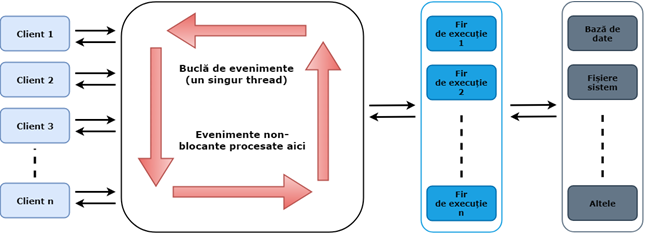
Avantajul Node.js este faptul că milioane de programatori front-end, ce pot scrie cod JavaScript pentru browser, sunt acum în măsură să scrie cod de server, fără a fi nevoie să învețe un limbaj de programare complet diferit.

Figura 32 - Exemplu funcționare Node.js

## 3.8 CSS

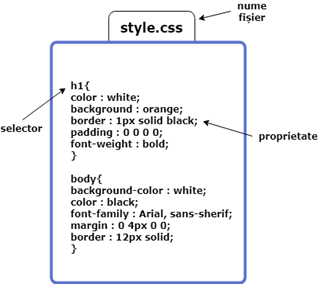
CSS (eng. Cascading Style Sheets) este un limbaj de stilizare folosit pentru a descrie modul în care arată un document scris într-un limbaj de marcare (eng. Markup language). Acest limbaj este unul din limbajele de baza ale World Wide Web, alături de HTML și Javascript.

Figura 33 - Exemplu fișier CSS

Este conceput pentru a permite separarea prezentării și a conținutului, inclusiv aspectul, culorile și fonturile. Această separare poate îmbunătăți accesibilitatea conținutului, poate oferi mai multă flexibilitate și control în specificațiile caracteristicilor de prezentare, permite mai multor pagini web să împărtășească formatarea prin specificarea atributelor CSS relevante într-un fișier .css separat și reducerea complexității și repetării în conținutul structural. Pe lângă HTML, există și alte limbaje de marcare ce permit utilizarea CSS pentru înfrumusețare (XHTML, XML simplu, SVG, XUL).

### 3.8.1 Sass (Syntactically awesome style sheets) [28]

Este un limbaj de scriptare ce este interpretat sau compilat în CSS. Folosește acolade pentru a delimita blocurile de cod și caracterul punct și virgulă (;) pentru delimitarea proprietăților. Sass extinde limbajul CSS prin adăugarea unor mecanisme specifice limbajelor orientate pe obiect.

## 3.9 Bootstrap [29]

Este un framework al CSS ce permite stilizarea aplicațiilor web de interfață grafică. Acest framework este gratuit, „open-source” și orientat către dezvoltarea de aplicații web „responsive” și direcționate în principal către platformele mobile. Acesta conține șabloane de design CSS și (opțional) bazate pe JavaScript pentru tipografie, formulare, butoane, navigație și alte componente ale interfeței.

După lansarea open-source în 2011, Bootstrap a devenit popular foarte repede. Designerii și programatorii web folosesc Bootstrap, deoarece este flexibil și ușor de utilizat. Printre principalele sale avantaje se numără și faptul că oferă un design consistent prin utilizarea componentelor reutilizabile, este foarte ușor de utilizat și rapid de învățat. Oferă extensibilitate bogată prin folosirea JavaScript, venind cu suport integrat pentru pluginurile jQuery [30]. Bootstrap poate fi folosit cu orice IDE sau editor, și orice tehnologie și limbă de la server, de la ASP.NET [31] la PHP [32] la Ruby on Rails [33]. Cu Bootstrap, dezvoltatorii web se pot concentra pe muncă de dezvoltare, fără să își facă griji cu privire la design.

### 3.9.1 Avantaje

* Economisește timp și este ușor de folosit. Folosind Bootstrap, se poate reduce timpul folosit pentru a crea felul în care o componentă arată. Pot fi utilizate direct șabloanele și clasele de design predefinite în Bootstrap.
* Customizabil. Un avantaj al aplicației Bootstrap este că oferă multe moduri de a fi personalizat, astfel încât fiecare își poate crea o temă proprie. Bootstrap poate fi ajustat în funcție de dorințele utilizatorului sau ale proiectului. Acest lucru este realizat folosind pagina de personalizare Bootstrap.
* Consistență. Problema pe care Bootstrap a rezolvat-o de la început a fost neconcordanțele dintre designeri și programatorii care lucrează la proiectele lor. Acesta este unul dintre principalele motive pentru care Bootstrap a ajuns ceea ce este în ziua de astăzi : un set central de cod de dezvoltare care rezolvă problemele dintre programatori și utilizatorul final. Felul în care Bootstrap stilizează componentele interfețelor este același pentru fiecare platformă, arătând la fel în toate browserele (Internet Explorer, Chrome, Firefox).
* Adaptarea la formă ecranului. Nevoia de a avea un site web ce își modifică felul în care este așezat în pagină conținutul este foarte importantă. Crearea site-urilor ce pot fi vizualizate și de pe o platformă mobilă este acum un proces mult mai simplu folosind Bootstrap datorită aspectului grilei (eng. grid) care se adaptează dinamic la rezoluția corectă a ecranului. Dacă se trece de la un laptop la un iPad, Bootstrap se adaptează la schimbarea platformelor.
* Este bine documentat. Un alt avantaj al aplicației Bootstrap este faptul că oferă o documentație impresionantă, exemple și demo-uri care vor face mai ușoară utilizarea pentru cineva ce abia începe a folosi acest framework. În acest fel, poate fi economisit mult timp, toate informațiile necesare fiind în documentație.
* Este open-source. Unul dintre principalele motive pentru a utiliza Bootstrap este faptul că este un proiect open-source, găzduit de Github. Acesta nu trebuie cumpărat, lăsând și posibilitatea de a schimba codul după preferințe.

### 3.9.2 Dezavantaje

* Toate site-urile arată la fel. Bootstrap este popular și ușor de folosit, de aceea toată lumea îl folosește. Chiar dacă este personalizabil, oferindu-ți șansa de a-ți crea propria temă, nu toată lumea face asta. Ca și rezultat un număr impresionant de site-uri web vor avea mare parte din felul în care componentele arată foarte asemănător.
* Este foarte “voluminos”. Bootstrap ajută în construirea unui site atractiv și responsive, dar din cauza problemele legate de încărcarea lentă și de descărcarea rapidă a bateriilor, unii utilizatori ar putea renunța la vizualizarea site-ului în favoarea altuia ce îndeplinește funcțiile necesare într-un timp mai redus. Bootstrap vine cu o mulțime de linii de CSS și JS, ceea ce este un lucru bun, dar și un lucru rău din cauza conexiunii slabe la internet.
* Nu suportă Sass. Bootstrap este construit cu LESS (eng. Leaner Style Sheets) [34] și nu oferă suport nativ pentru SASS.

## 3.10 ChartJS [35]

Este o librărie open source scrisă în limbajul JavaScript ce ne permite să creăm o multitudine de tipuri de grafice direct în pagina web folosind un set predefinit de date. Avantajul și în același timp dezavantajul acestei librării este faptul că absolut fiecare element poate fi personalizat, dar în același timp configurarea tuturor opțiunilor poate fi o adevărată provocare pentru unele persoane.

## 3.11 Angular

Este un framework al limbajului JavaScript ce este folosit pentru a crea aplicații web dinamice de tip SPA (eng. Single Page Application). A fost creat ca și un proiect secundar de către un angajat Google numit Miško Hevery pentru a îl ajuta în scrierea mai rapidă a unor aplicații web interne ale companiei. Mai târziu, acest proiect secundar, a devenit cunoscut sub denumirea de AngularJS (motivul acestei denumiri fiind caracterele <, > din HTML). În 2010 Angular JS a fost publicat ca și proiect „open source”, iar cu timpul unele din cele mai mari branduri au început să îl folosească pentru crearea de aplicații web.

La câțiva ani după lansare, modul de dezvoltare al aplicațiilor web a început să se schimbe, iar Angular începuse să rămână în urmă, nemaiputând fi îmbunătățit în forma existența. Acest lucru a determinat echipa ce lucra la îmbunătățirea continuă a acestui proiect să îl rescrie complet, iar în dată de 14 septembrie 2016 Angular 2 a fost lansat, versiunea anterioară nemaifiind suportată. Pentru a se evita confuzia, echipa a decis să elimine versiunea din numele Framework-ului, denumirea finală rămânând Angular.

### Avantaje

Arhitectura bazată pe componente. Componentele pot fi considerate ca fiind mici bucăți ale unei interfețe ce sunt independente una de cealaltă.

În figura 34 avem o simplă aplicație cu o listă de elemente ce corespund unui text dat. Chenarul în care sunt afișate elementele, cel în care se scrie textul (după care se vor caută elementele) și cel ce le conține pe amândouă sunt considerate componente separate în Angular.

În timp ce AngularJS fusese creat pentru construirea de aplicații cu arhitectură de tip MVC (eng. Model-View-Controller), începând de la versiunea a doua, Angular se bazează mai mult pe folosirea componentelor. Arhitectura este similară cu cea MVC, diferența fiind abilitatea de a reutiliza componentele oriunde în aplicație. Această abordare ne permite construirea de interfețe grafice ce conțin multe părți în mișcare. Beneficiile unei astfel de arhitecturi sunt:

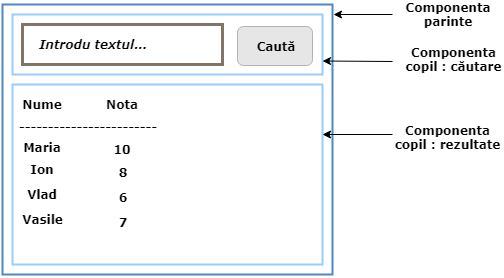
1. Reutilizare. Componentele pot fi folosite de către programatori în mai multe părți ale aplicației. Acest lucru este folositor mai ales în aplicațiile de tip “enterprise” ce conțin un număr considerabil de componente, unele din ele având subcomponente similare (ex: liste de sortare, calendare, etc).
2. Lizibilitate. Proiectul fiind structurat pe componente face orientarea unui programator nou, ce tocmai a intrat într-un proiect, mult mai ușoară.

Figura 34 - Exemplu interfață bazată pe componente

1. Codul este ușor de testat. Natura independentă a componentelor simplifică testarea proiectului, putând fi verificate până și cele mai mici părți ale proiectului.
2. Ușor de menținut. Componentele pot fi ușor înlocuite cu implementări mai bune.

### 3.11.2 TypeScript

Angular este scris folosind limbajul TypeScript, acesta compilandu-se în JavaScript. În timp ce micile proiecte scrise în JavaScript nu necesită o verificare amănunțită, cele de tip „enterprise” obligă programatorii să își scrie codul mai clar și să îi verifice mai des calitatea. Acest limbaj și-a făcut prima apariție publică în octombrie 2012 (la acel timp având versiunea 0.8), după doi ani de dezvoltare de către firma Microsoft, iar în anul 2014 la conferința „Microsoft’s Build” a fost lansat oficial. În prezent, TypeScript este considerat limbajul de programare de baza pentru Angular, documentația acestui framework fiind creată pentru TypeScript.

### 3.11.3 RxJS [36]

RxJS este o librărie folosită în principal în Angular pentru cererile asincrone de date. În mai multe site-uri de specialitate, această librărie este comparată cu linia de asamblare a mașinilor a lui Henry Ford. Permite realizarea independentă și în paralel a evenimentelor și continuarea execuției fără a fi nevoie a aștepta anumite evenimente să se întâmple (în acest timp pagina web rămânând blocată). În principiu funcționează asemenea unei linii de asamblare, unde execuția unei comenzi este împărțită în mai multă părți individuale și interschimbabile. În timp ce mulți ingineri se plâng de faptul că timpul necesar învățării RxJS este destul de mare, odată stăpânită aceasta va deveni un instrument indispensabil în aplicațiile bazate pe arhitectura cerere-răspuns.  Această librărie folosește observatori (eng. Observables). Observatorii pot fi asemănați cu un model (eng. blueprint) ce descriu modul în care fluxurile de date sunt combinate și cum aplicația reacționează la variabilele din aceste fluxuri. RxJS poate fi folosit și cu alte framework-uri de „front-end” (React) sau „back-end” (NodeJS) dar în Angular RxJS a devenit o necesitate.

### 3.11.4 Asistență Google

Mulți ingineri sunt de părere că asistența oferită acestui Framework de către Google ca fiind un mare avantaj. Chiar dacă acesta părere este justificată, doar suportul Google nu este suficient. Vestea bună este faptul că, Google, a anunțat suport pe termen lung (eng. Long-Term Support) pentru această tehnologie, Igor Minan și Steven Fuin, inginerii din spatele Angular confirmând acest lucru.

## 3.12 REST (REpresentational State Transfer)

Este un stil arhitectural software care definește un set de constrângeri care trebuie utilizate pentru crearea serviciilor Web. Serviciile Web ce respectă REST asigură interoperabilitatea între sistemele conectate la Internet. Sistemele compatibile cu REST, deseori numite sisteme RESTful, se caracterizează prin modul în care acestea separă clientul de server.

În arhitectura de tip REST, implementarea aplicației client cât și cea de server pot fi făcute independent una față de cealaltă. Acest lucru permite modificarea, în orice moment, a aplicației client fără a afecta funcționalitatea serverului, iar codul de pe server poate fi modificat fără a afecta funcționalitatea clientului. Atâta timp cât fiecare parte cunoaște formatul mesajului necesar celuilalt, acestea pot fi păstrate modulare și separate. Separând preocupările legate de interfața cu utilizatorul din preocupările legate de stocarea datelor, îmbunătățim flexibilitatea interfeței între platforme și totodată îmbunătățim și scalabilitatea prin simplificarea componentelor. În plus, separarea permite fiecărei componente capacitatea de a evolua independent.

Sistemele care urmăresc paradigma REST sunt „stateless”, ceea ce înseamnă că serverul nu trebuie să știe nimic despre starea în care se află clientul și invers. În acest fel, serverul și clientul pot înțelege orice mesaj primit, chiar fără a vedea mesajele anterioare. Din acest motiv aplicațiile REST sunt mai fiabile, au o performanță crescută și sunt scalabile, acest lucru permițând actualizarea și reutilizarea componentelor fără a afecta sistemul ca întreg, chiar și în timpul funcționării acestuia.

### 3.12.1 Comunicarea între client și server.

În arhitectura de tip REST, clientul trimite cereri pentru a obține sau modifica o anumită resursă, iar serverul răspunde la acestea prin resurse sau printr-un mesaj de confirmare sau eroare în executarea acțiunii.

În general, o cerere către server trebuie să conțină:

* un verb de tip HTTP, ce definește tipul operației ce trebuie executate
* un “header”, ce permite clientului să trimită informații adiționale despre acea cerere
* calea către resursa dorită
* un mesaj opțional ce conține date necesare serverului pentru a efectua o anumită operațiune

Într-un sistem de tip REST există 4 verbe HTTP ce sunt folosite în cereri pentru interacțiunea cu resursele :

* GET – permite obținerea unei anumite resurse de la un anumit server folosind un anumit URI (eng. Uniform Resource Identifier). Aceste tipuri de cereri sunt folosite doar pentru a obține o anumită resursă și nu pentru a modifica datele deja existente.
* POST – permite trimiterea informației către server, de exemplu, datele de contact ale unui client, fișiere, etc.
* PUT – înlocuiește toate reprezentările curente ale unei resurse cu conținutul nou încărcat.
* DELETE – șterge toate reprezentările curente ale unei resurse.

Când o cerere este trimisă către server, acesta răspunde ori printr-un mesaj de confirmare a efectuării operației dorite, o eroare sau o anumită resursă. Pentru fiecare dintre acestea, serverul trimite în headerul răspunsului și un cod numit status. Din multitudinea de coduri existente, cele mai des întâlnite sunt :

* 200 (OK) – Răspunsul standard pentru cererile HTTP realizate cu succes.
* 201 (Creat) – Răspunsul standard pentru cererile HTTP în urma cărora a fost creat cu succes un element/resursă.
* 204 (Fără conținut) – Răspunsul standard pentru cererile HTTP ce nu conțin nimic în corpul răspunsului.
* 400 (Cerere greșită) – Cererea nu poate fi prelucrată din cauza unor erori de sintaxă, mărime prea mare sau o altă eroare din partea clientului.
* 403 (Interzis) – Clientul nu are permisiunea de a accesa resursa.
* 404 (Negăsit) – Resursă căutată nu a putut fi găsită. Este posibil ca aceasta să fi fost ștearsă sau să nu existe încă.
* 500 (Eroare internă server) – Răspunsul generic pentru o eroare neașteptată, dacă nu există alte informații disponibile.

În REST orice informație ce poate fi manipulată poartă numele de resursă. Orice informație poate fi denumită resursă: un document sau o imagine, un serviciu temporar, o colecție de alte resurse, un obiect non-virtual, etc. REST folosește un identificator de resurse (eng. resource identifier) pentru a identifica o anumită resursă ce este implicată în interacțiunea dintre componente.

Starea unei resurse la un moment da teste cunoscută sub numele de reprezentare a resursei (eng. resource representation). Această reprezentare conține date și metadate ce descriu datele.

## 3.13 MySQL

Este un sistem de management al bazelor de date relaționale bazat pe limbajul SQL (eng. Structured Query Language) și susținut de compania Oracle. MySQL rulează pe aproape toate platformele, inclusiv Linux, UNIX și Windows. Deși poate fi folosit într-o gamă largă de aplicații MySQL este cel mai adesea asociat cu aplicațiile web.

MySQL este o componentă importantă LAMP. LAMP este o platformă de dezvoltare web care utilizează Linux ca sistem de operare, Apache ca server Web, MySQL ca sistem de management al bazelor de date relaționale și PHP ca limbaj de scripting orientat pe obiecte.

Inițial a fost concepută de compania suedeză MySQL AB. În 2008 a fost achiziționată de Sun Microsystems, iar apoi de Oracle când a cumpărat Sun în 2010. Dezvoltatorii pot folosi MySQL sub licență GNU General Public Licence (GPL), dar companiile trebuie să obțină o licență comercială.

În ziua de astăzi MySQL este în spatele multor site-uri de top din lume și nenumărate aplicații web, printre care se numără și Facebook, Twitter și YouTube.

MySQL este bazat pe modelul client-server. Partea cea mai importantă este reprezentată de serverul MySQL care gestionează toate instrucțiunile (sau comenzile) bazei de date. MySQL server este disponibil ca și program separat pentru utilizarea într-o aplicație de tip client-server dar și ca o librărie ce poate fi încorporată în aplicații. MySQL operează împreună cu mai multe programe utilitare care sprijină administrarea bazelor de date MySQL. Comenzile sunt trimise către MySQL Server prin intermediul clientului MySQL, care este instalat pe un computer.

MySQL a fost inițial dezvoltată pentru a gestiona rapid bazele de date mari. Deși MySQL este de obicei instalat pe o singură mașină, este capabil să trimită date către mai multe locații, deoarece utilizatorii o pot accesa prin interfețe client MySQL diferite. Aceste interfețe trimit instrucțiuni SQL către server și apoi afișează rezultatele.

MySQL permite stocarea și accesarea datelor în mai multe moduri, inclusiv InnoDB, CSV și NDB. Utilizatorii MySQL nu sunt obligați să învețe comenzi noi; aceștia pot accesa datele lor utilizând comenzi SQL standard.

Este scris în C și C++ și poate fi folosit pe multe platforme, incluzând Mac, Linux, Windows și Unix.RDBMS (eng. relațional database management system) suportă baze de date mari cu milioane de înregistrări, acceptând multe tipuri de date, incluzând numere întregi cu lungimi de 1,2,3,4 și 8 octeti; FLOAT; DOUBLE; CHAR; VARCHAR; BINARY; VARBINARY; TEXT; BLOB; DATE; TIME; DATETIME; TIMESTAMP; YEAR; SET; ENUM.

Pentru securitate, MySQL utilizează accesul bazat pe roluri (administrator, utilizator, etc) și un sistem de criptare a parolei. Clienții MySQL se pot conecta la serverul MySQL utilizând mai multe protocoale, inclusiv prize TCP / IP de pe orice platformă. MySQL suportă, de asemenea, o serie de programe client și utilitare, programe în linie de comandă și instrumente de administrare, cum ar fi MySQL Workbench.

## 3.14 Spring Data JPA (eng. Java Persistence API)

Maparea obiectelor Java în tabelele de baze de date și invers se numește mapare obiect-relațională (ORM). Prin intermediul JPA, dezvoltatorul poate mapa, stoca, actualiza și prelua date din bazele de date relaționale sub formă de obiecte Java și invers. JPA poate fi folosit în aplicațiile Java-EE și Java-SE.

JPA permite programatorului să lucreze direct cu obiecte, diminuând nevoia de a scrie instrucțiuni SQL. Implementarea JPA este numită în mod tipic furnizor de persistență (eng. persistence provider).

Maparea între obiectele Java și tabelele bazei de date este definită prin intermediul metadatelor de persistență. Metadatele JPA sunt definite prin adnotări ce sunt scrise în clasele Java (deasupra variabilelor, a funcțiilor sau chiar a claselor). Alternativ, metadatele pot fi definite prin XML sau printr-o combinație a celor două. O configurație XML suprascrie adnotările.

JWT este una dintre cele mai bune modalități de a securiza o transmisie între două părți (aplicații) prin folosirea token-urilor. Aceste părți pot fi servere, clienți sau orice altă combinație de aplicații.

## 3.15 JWT (JSON Web Token)

Securizarea JWT se face printr-un algoritm de criptare. Un avantaj principal al utilizării unui JWT este faptul că este foarte compact (presupunând că emitorul utilizează JWS Compact Serialization, care este și recomandat). Token-ul este în general destul de mic, din acest motiv putând fi trimis printr-o cerere POST, într-un antet HTTP sau chiar ca parte URL. Cu toate acestea, mărimea JWT va crește odată cu numărul opțiunilor adăugate. S-ar putea crea teoretic un JWT care depășește lungimea maximă a unei adrese URL (~ 2000 de caractere).

Mod de folosință JWT

JWT va trebui să fie trimis cu fiecare cerere către backend, acesta fiind un compromis care trebuie luat în considerare în momentul în care se ia decizia de a folosi aceasta tehnologie. Marele beneficiu al acestei abordări este acela că aceasta oferă o formă de autentificare stateless, deoarece serverul nu trebuie să-și amintească informațiile utilizatorului în timpul stocării sesiunii, reducând în mod semnificativ cantitatea de muncă necesară pentru a gestiona aceste informații în partea de backend. Un dezavantaj este că, deoarece JWT-urile sunt stateless, nu pot fi invalidate. JWT-urile vor fi invalidate automat după data de expirare a acestora, dar în funcție de durata de expirare (10 ore este comună), un utilizator poate păstra accesul la un serviciu chiar după ce a fost eliminat.

### JSON (JavaScript Object Notation)

JSON este o modalitate de a stoca informațiile într-un mod organizat și ușor de accesat. Pe scurt oferă o colecție de date ce este inteligibilă oamenilor. Prin faptul ca este mai simplu, JSON devine o alternativă mai bună și mai simplă a limbajului XML.

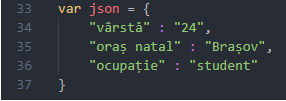
Modul de stocare a datelor în format JSON :

Figura 35 - Exemplu interfață bazată pe componente

Această modalitate de scriere creează un obiect ce poate fi accesat folosind variabila “json”.

Prin încapsularea valorii variabilei între acolade, indicăm faptul că valoarea este un obiect. În interiorul obiectului, putem declara orice număr de proprietăți folosind o pereche de forma “cheie”: “valoare” separate prin virgulă. Pentru accesarea informațiilor stocate în JSON, putem pur și simplu scrie numele obiectului urmat de punct și numele proprietății dorite.

# Concluzii

In concluzie, îmi susțin afirmația introdusă în introducere ce susține faptul că în ziua de astăzi, din dorința de a rezolva cât mai multe într-un timp cât mai scurt, fiecare persoană este într-o grabă continuă, iar ca și consecința lucrurile pe care trebuie să le reținem devin din ce în ce mai numeroase, iar șansele ca din aceste informații unele să fie uitate sunt destul de mari, mai ales cele de rutină cum ar fi, stingerea unui bec când s-a părăsit locuința sau scoaterea unui încărcător din priză în momentul în care dispozitivul a fost încărcat în întregime ce nu pot fi programate cu exactitate. Din aceste motive consider că o aplicație ce poate realiza aceste funcții într-un timp cât mai scurt, pentru economisirea energiei este imperios necesară.

În același timp așa cum am menționat și în introducere, necesitatea existenței unei aplicații ce gestionează modul în care curentul produs de un panou solar este împărțit între circuitele casei astfel încât energia dată de panou sa fie folosită cât mai optim este foarte ridicată, întrucât nu multe persoane au resursele financiare pentru a alimenta toată locuința doar din energie regenerabilă, astfel putând achiziționa doar un număr limitat de panouri.

Din punct de vedere arhitectural pentru realizarea proiectului au fost create 4 aplicații independente ce comunica între ele prin diverse servicii de mesagerie (REST, RabbitMQ, Sockets), iar pentru rezolvarea problemelor anterior menționate au fost implementate trei scenarii, implementarea lor fiind facilitată de serviciul CEP ce permite o prelucrare și corelare rapidă a mai multor evenimente în timp real.

## 4.1 Posibilități de dezvoltare ulterioară

Există multe direcții ulterioare în care acest proiect se poate îndrepta. Pe parcursul dezvoltării aplicațiilor am realizat faptul că fiecăreia i se pot adăuga îmbunătățiri pentru atingerea maximului potențial din punct de vedere al vitezei.

O modalitate de dezvoltare ar fi achiziționarea unui server, ce este specializat în procesarea datelor, în momentul actual majoritatea procesării efectuate de către aplicații este realizată de un laptop ce nu permite atingerea potențialului maxim având resurse limitate.

# 5 Bibliografie

|  |  |
| --- | --- |
| [1] | „Wikipedia,” [Interactiv]. Available: https://en.wikipedia.org/wiki/Smart\_home\_technology. [Accesat 10 Martie 2019]. |
| [2] | K. Frampton, „Wikipedia,” [Interactiv]. Available: https://ro.wikipedia.org/wiki/Le\_Corbusier. [Accesat 15 Iunie 2019]. |
| [3] | R. P. FOUNDATION, „Raspberry Pi,” [Interactiv]. Available: https://www.raspberrypi.org/. [Accesat 10 Martie 2019]. |
| [4] | „IoT Agenda,” [Interactiv]. Available: https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT. [Accesat 25 Mai 2019]. |
| [5] | „Arduino,” [Interactiv]. Available: https://www.arduino.cc/. [Accesat 22 Mai 2019]. |
| [6] | E. Y. D. a. S. R. Wu, „iwringer.wordpress.com,” [Interactiv]. Available: https://iwringer.wordpress.com/2012/05/18/how-to-scale-complex-event-processing-cep-systems/. [Accesat 5 Iunie 2019]. |
| [7] | R. E. Stross, „Microsoft,” [Interactiv]. Available: https://www.microsoft.com/. [Accesat 10 Iunie 2019]. |
| [8] | „EsperTech,” [Interactiv]. Available: http://www.espertech.com/esper/. [Accesat 12 Aprilie 2019]. |
| [9] | K. J. G. ș. D. H. Arnold, The Java programming language, Addison Wesley Professional, 2005. |
| [10] | A. a. J. J. W. Videla, RabbitMQ in action: distributed messaging for everyone, 2012: Manning. |
| [11] | „MySQL,” [Interactiv]. Available: https://www.mysql.com/. [Accesat 22 Mai 2019]. |
| [12] | „JAVAWORLD,” [Interactiv]. Available: https://www.javaworld.com/article/3379043/what-is-jpa-introduction-to-the-java-persistence-api.html. [Accesat 5 Iunie 2019]. |
| [13] | „REST API,” [Interactiv]. Available: https://restfulapi.net/. [Accesat 11 Aprilie 2019]. |
| [14] | „JSON,” [Interactiv]. Available: https://www.json.org/. [Accesat 8 Martie 2019]. |
| [15] | „Baeldung,” [Interactiv]. Available: https://www.baeldung.com/a-guide-to-java-sockets. [Accesat 10 Aprilie 2019]. |
| [16] | „JavaScript,” [Interactiv]. Available: https://www.javascript.com/. [Accesat 13 Aprilie 2019]. |
| [17] | „Angular,” [Interactiv]. Available: https://angular.io/. [Accesat 13 Aprilie 2019]. |
| [18] | „JWT,” [Interactiv]. Available: https://jwt.io/. [Accesat 10 Mai 2019]. |
| [19] | „HTML.COM,” [Interactiv]. Available: https://html.com/. [Accesat 17 Mai 2019]. |
| [20] | „w3schools.com,” [Interactiv]. Available: https://www.w3schools.com/css/css\_intro.asp. [Accesat 20 Mai 2019]. |
| [21] | „TypeScript,” [Interactiv]. Available: https://www.typescriptlang.org/. [Accesat 20 Mai 2019]. |
| [22] | „Apache Maven,” [Interactiv]. Available: https://maven.apache.org/. [Accesat 1 Iunie 2019]. |
| [23] | „W3,” [Interactiv]. Available: https://www.w3.org/XML/. [Accesat 1 Iunie 2019]. |
| [24] | „Spring,” [Interactiv]. Available: https://spring.io/. [Accesat 1 Iunie 2019]. |
| [25] | I. Cobeanu, Strategii de coordonare în timp real a comportamentului agenților inteligenți mobili, Brașov, 2012. |
| [26] | „Node.js,” [Interactiv]. Available: https://nodejs.org/en/. [Accesat 2 Iunie 2019]. |
| [27] | [Interactiv]. Available: https://www.google.com/intl/ro\_ro/chrome/. [Accesat 2 Iunie 2019]. |
| [28] | „SASS,” [Interactiv]. Available: https://sass-lang.com/. [Accesat 2 Iunie 2019]. |
| [29] | „Bootstrap,” [Interactiv]. Available: https://getbootstrap.com/. [Accesat 3 Iunie 2019]. |
| [30] | „jQuery,” [Interactiv]. Available: https://jquery.com/. [Accesat 3 Iunie 2019]. |
| [31] | „Microsoft .NET,” [Interactiv]. Available: https://dotnet.microsoft.com/apps/aspnet. [Accesat 4 Iunie 2019]. |
| [32] | „PHP,” [Interactiv]. Available: https://www.php.net/manual/ro/intro-whatis.php. [Accesat 4 Iunie 2019]. |
| [33] | „Ruby on Rails,” [Interactiv]. Available: https://rubyonrails.org/. [Accesat 4 Iunie 2019]. |
| [34] | „LESS,” [Interactiv]. Available: http://lesscss.org/. [Accesat 4 Iunie 2019]. |
| [35] | „Chart.js,” [Interactiv]. Available: https://www.chartjs.org/. [Accesat 5 iunie 2019]. |
| [36] | „RxJS,” [Interactiv]. Available: https://rxjs-dev.firebaseapp.com/. [Accesat 5 Iunie 2019]. |