**Universitatea „Alexandru Ioan Cuza” Iași**

`

**Facultatea de Informatică**

****

LUCRARE DE LICENȚĂ

**OPID - Online Platform for IoT Devices**

propusă de

**Lazăr Lila-Ciprian**

**Sesiunea:** Iulie, 2017

Coordonator științific

**Conferențiar, Dr. Sabin Corneliu Buraga**

**Universitatea „Alexandru Ioan Cuza” Iași**

**Facultatea de Informatică**

**OPID – Online Platform for IoT Devices**

Managementul dispozitivelor interconectate via Internet

**Lazăr Lila-Ciprian**

**Sesiunea:** Iulie, 2017

Coordonator științific

**Conferențiar, Dr. Sabin Corneliu Buraga**

DECLARAŢIE PRIVIND ORIGINALITATE ŞI RESPECTAREA

DREPTURILOR DE AUTOR

Prin prezenta declar că Lucrarea de licenţă cu titlul “*Online Platform for IoT Devices*” este scrisă de mine şi nu a mai fost prezentată niciodată la o altă facultate sau instituţie de învăţământ superior din ţară sau străinătate. De asemenea, declar că toate sursele utilizate, inclusiv cele preluate de pe Internet, sunt indicate în lucrare, cu respectarea regulilor de evitare a plagiatului:

* toate fragmentele de text reproduse exact, chiar şi în traducere proprie din altă limbă, sunt scrise între ghilimele şi deţin referinţa precisă a sursei;
* reformularea în cuvinte proprii a textelor scrise de către alţi autori deţine referinţa precisă;
* codul sursă, imagini etc. preluate din proiecte *open*-*source* sau alte surse sunt utilizate cu respectarea drepturilor de autor şi deţin referinţe precise;
* rezumarea ideilor altor autori precizează referinţa precisă la textul original.

Iaşi, *29.06.2017*

Lila-Ciprian Lazăr

(semnătura în original)

DECLARAŢIE DE CONSIMŢĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licenţă cu titlul „*Online Platform for IoT Devices*”, codul sursă al programelor şi celelalte conţinuturi (grafice, multimedia, date de test etc.) care însoţesc această lucrare să fie utilizate în cadrul Facultăţii de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea Alexandru Ioan Cuza Iaşi să utilizeze, modifice, reproducă şi să distribuie în scopuri necomerciale programele calculator, format executabil şi sursă, realizate de mine în cadrul prezentei lucrări de licenţă.

Iaşi, *29.06.2017*

Lila-Ciprian Lazăr

(semnătura în original)

**Rezumat**

*„Internet of Things”*  este un concept ce devine mai popular pe zi ce trece având potențialul de a influența modul în care trăim. Pe scurt, termenul IoT cuprinde toate obiectele conectate la Internet, însă este folosit din ce în ce mai des pentru a defini dispozitive care interacționează între ele. Combinarea acestor dispozitive cu sisteme automate face posibilă strângerea de date, analizarea lor și crearea unor anumite acțiuni cu scopul de a automatiza sarcini sau pentru a aduna cunoștințe.

*„Online Platform for IoT Devices”* vine în ajutorul utilizatorilor facilitând interacțiunea dintre aceștia și senzori/actuatori, oferind în același timp o soluție pentru problema interacțiunii între dispozitive. Comunicarea între dispozitivele fizice și platformă se realizează prin intermediul protocolului de comunicare MQTT. Utilizatorii vor fi notificați în legătură cu modificările survenite în sistem chiar și în momentul în care nu interacționează cu platforma. Automatizarea acțiunilor unui utilizator se realizează prin intermediul unui sistem bazat pe reguli.

Aplicația a fost realizată prin intermediul unui API RESTful dezvoltat folosind Express, un *framework* bazat peNode.js pentru dezvoltarea de aplicații pe partea de server. Pentru notificarea utilizatorilor s-a folosit serviciul de *Push Notification* pus la dispoziție de către cei de la Firebase. Pentru dezvoltarea aplicației *front-end*  au fost utilizate *framework*-urile AngularJs și Materialize. Datorită faptului că datele furnizate de dispozitive nu au o structură fixă, s-a decis că o soluție potrivită ar fi utilizarea unei baze de date NoSQL (e.g MongoDB) ca modalitate de stocare. Transferul de mesaje între dispozitive și platformă se realizează prin intermediul unui broker MQTT oferit de AWS IoT.

În capitolele următoare vor fi menționate fundamentele care stau la baza dezvoltării acestei platforme și tehnologiile folosite. De asemenea, se va prezenta o detaliere a soluției identificate făcând referire la: principalele componente ale aplicației, interacțiunea dintre ele și implementarea efectivă. Un manual de utilizare a fost inclus pentru a exemplifica modalitatea de funcționare. Lucrarea de față se încheie cu o serie de concluzii referitoare la modul în care soluția identificată rezolvă problema dată și cu direcțiile de dezvoltare ulterioare.

Cuprins

[1.Introducere 7](#_Toc486628115)

[1.1 Obiective 7](#_Toc486628116)

[1.2 Formularea problemei și identificarea soluției 7](#_Toc486628117)

[1.3 Prezentarea soluției 7](#_Toc486628118)

[2. Fundamente 8](#_Toc486628119)

[2.1 IOT 8](#_Toc486628120)

[2.2 Modele de comunicare 12](#_Toc486628121)

[2.2.1 Comunicare Device-to-Device 12](#_Toc486628122)

[2.2.2 Comunicare Device-to-Cloud 13](#_Toc486628123)

[2.2.3 Modelul Device-to-Gateway 14](#_Toc486628124)

[2.2.4 Back-End Data-Sharing Model 15](#_Toc486628125)

[2.3 Cloud Computing 16](#_Toc486628126)

[2.4 Cloud of Things 19](#_Toc486628127)

[2.5 MQTT(Message Queueing Telemetry Transport) 20](#_Toc486628128)

[2.5.1 Client 22](#_Toc486628129)

[2.5.2 Broker 22](#_Toc486628130)

[2.5.3 Topics 23](#_Toc486628131)

[3. Tehnologii folosite 25](#_Toc486628132)

[3.1 Node.js 25](#_Toc486628133)

[3.2 AngularJS 26](#_Toc486628134)

[3.3 MongoDB 26](#_Toc486628135)

[3.4 Firebase Cloud Messaging 27](#_Toc486628136)

[3.5 AWS IoT 28](#_Toc486628138)

[4. Scopurile și cerințele aplicației 29](#_Toc486628139)

[4.1 Justificarea necesității aplicației 29](#_Toc486628140)

[4.2 Modul în care aplicația rezolvă problema vizată 29](#_Toc486628141)

[5. Analiza și proiectarea aplicației 31](#_Toc486628142)

[5.1 Analiza problemei 31](#_Toc486628143)

[5.1.1 Problema existentă 31](#_Toc486628144)

[5.1.2 Identificarea soluției 31](#_Toc486628145)

[5.2 Proiectarea aplicației 31](#_Toc486628146)

[5.2.1 Cerințele aplicației 31](#_Toc486628147)

[5.2.2 Arhitectura generala 32](#_Toc486628148)

[5.2.3 Things 33](#_Toc486628149)

[5.2.4 Rules 35](#_Toc486628150)

[5.2.5 FCM Notifications 36](#_Toc486628151)

[5.2.6 Job Scheduler 37](#_Toc486628152)

[5.2.7 Baza de date 37](#_Toc486628153)

[5.2.8 Sensors Simulator 38](#_Toc486628154)

[5.2.9 Devices Gateway 39](#_Toc486628155)

[6. Implementare 40](#_Toc486628156)

[6.1 Detalii tehnologice 40](#_Toc486628157)

[6.2 Interfața cu utilizatorul 49](#_Toc486628158)

[7. Manual de utilizare 53](#_Toc486628159)

[8. Studiu de caz 62](#_Toc486628160)

[9. Concluzii 66](#_Toc486628161)

[10. Bibliografie 68](#_Toc486628162)

[Anexa 1 71](#_Toc486628163)

# 1.Introducere

## Obiective

În această lucrare ne propunem să identificăm problemele referitoare la interacțiunea dintre utilizator și device-uri, interoperabilitatea între dispozitive și să analizăm o soluție care ar putea rezolva această problemă.

## Formularea problemei și identificarea soluției

Datorită faptului că *„Internet of Things”* este un concept în plină dezvoltare, nu există un protocol standard care să fie utilizat pentru comunicarea M2M (Machine-to-Machine). Acest fapt se datorează și numărului foarte ridicat de dispozitive pe care IoT-ul dorește să le interconecteze. IoT-ul în momentul de față, nu are foarte multe elemente comune cu Internetul și ar trebui să poarte numele de „Intranet of Things” datorită grupurilor izolate de dispozitive ce nu au fost proiectate să comunice între ele. O altă caracteristică definitorie a IoT-ului o reprezintă posibilitatea de interacțiune cu dispozitivele oricând și oriunde ne-am afla. În acest sens, nu au fost dezvoltate încă foarte multe platforme web care să faciliteze acest tip de interacțiune. Utilizatorul trebuie să fie ținut mereu la curent cu ultimele informații furnizare de device-uri.

O soluție pentru această problemă ar fi dezvoltarea unei platforme web care să rezolve o parte din problemele curente ale IoT-ului. În loc să dezvoltăm noi protocoale poate ar trebui să ne uităm în trecut și să vedem datorită cărui fapt, Internetul a ajuns ce este în ziua de azi. Pentru ca utilizatorii să poată interacționa cu dispozitivele o soluție ar fi utilizarea protocoalelor HTTP și MQTT, cel din urmă fiind folosit pentru comunicare dintre device-uri și platformă.

## Prezentarea soluției

*„Online Platform for IoT Devices”* are scopul de a rezolva problemele de interoperabilitate dintre device-uri și de a facilita accesul utilizatorului la dispozitive, de oriunde s-ar afla. Aplicația este formată din trei mari module. Primul modul se ocupă cu interacțiunea dintre utilizator și device-uri. Utilizatorii vor putea adăuga/elimina dispozitive din sistem și vor putea vedea toate informațiile primite de la senzori/actuatori în timp real. Al doilea modul are ca principal scop notificarea clientului în momentul în care s-a primit un mesaj din partea unui dispozitiv. Utilizatorul va primi aceste notificări și în momentul în care nu interacționează cu aplicația. Ultimul modul se concentrează pe oferirea posibilității de automatizare a anumitor acțiuni. Utilizatorul va avea posibilitatea să își seteze anumite acțiuni care să fie realizate în momentul în care sunt îndeplinite anumite condiții.

# 2. Fundamente

## 2.1 IOT

De la începuturile Internet-ului numărul dispozitivelor interconectate, inclusive cele cu putere limitată de calcul, a crescut exponențial. În 1990, John Romkey a creat primul dispozitiv conectat la Internet, respectiv, un prăjitor de pâine ce putea fi pornit și oprit prin intermediul Internetului [1]. În 1999 termenul de ‘Internet of Things’ a fost inventat de către Kevin Asthon. De asemenea, tot în acest an, Asthon a inventat un sistem de identificare la nivel global bazat pe RFID (Radio-Frequency Identification). Un salt major în comercializarea produselor IoT a avut loc în anul 2000, atunci când producătorul de electrocasnice LG a anunțat intenția de a dezvolta un fridgider inteligent care să determine singur dacă anumite produse stocate în mod regulat s-au terminat . În 2003, RFID (Radio-Frequency Identification) a fost implementat în toate supermarket-urile Walmart din lume. În 2005, mai multe publicații de renume precum The Guardian, Scientific American și Boston Globe au scris o multitudine de articole legate de IoT și de rolul important pe care acesta îl va avea în viitor iar la lansarea din 2011 a IPv6 s-a putut observa o creștere masivă a interesului public pentru acest domeniu.

IoT poate fi explicat într-un mod simplu ca o interconectare/interconexiune între oameni, computere și dispozitive. Aproape toate echipamentele pe care le folosim în viață de zi cu zi pot fi controlate și monitorizate utilizând IoT. Aceasta poate fi realizată utilizând trei tipuri de echipamente: senzori, actuatoare și noduri centrale. Senzorii sunt folosiți pentru a converti unități fizice în semnale digitale, nodurile centrale preiau datele măsurătorilor de la senzori și transmit aceste date către utilizator, iar actuatoarele transformă comenzile digitale din partea utilizatorului în acțiuni fizice. În acest mod putem monitoriza schimbările care apar din orice parte a lumii prin intermediul Internetului. Arhitectura respectivului sistem se bazează pe procesarea în timp real a datelor de la senzori. În cazul locuințelor inteligente, fiecare întrerupător poate fi conectat la un smartphone astfel încât să poată fi acționat de la distantă. În acest caz, nu este nevoie de un procesor sau un mediu de stocare ci doar de un actuator care să interpreteze comenzile închis/deschis într-o acțiune concretă în locuința. Evident, această arhitectură poate varia în funcție de nevoile utilizatorului.

*Internet of Things* este un concept și o paradigmă care presupune o proprie omniprezentă în mediu și implică o varietate de dispozitive interconectate fie prin intermediul unui mediu fizic de transmisie (wired) fie wireless. Modalitatea de comunicare dintre dispozitive necesită o schemă unică de adresare pentru a se evita coliziunile. Prin această interacțiune dispozitivele pot coopera pentru a atinge un scop comun, acela de a asigura servicii pentru utilizator prin intermediul aplicațiilor. Considerând toate acestea, potențialul de cercetare și dezvoltare în momentul de față este enorm, fiecare idee aducându-ne tot mai aproape de o lume în care realul și digitalul sunt unul și același lucru. Obiectivul *IoT*-ului este acela de a permite accesul la dispozitivele interconectate oricând, oriunde și oricui, revoluționând astfel Internet-ul așa cum îl știm noi astăzi. Putem să preconizam că inteligența sistemului va evolua exponențial datorită faptului că odată cu trecerea timpului tot mai multe elemente vor fi adăugate în rețea, fiecare dintre ele având capacitatea de a se auto recunoaste și a comunica informații despre sine, o parte dintre ele având rolul de a agrega date [2].

Această transformare este concomitentă cu apariția *cloud-computing*-ului și tranziția către *IPv6*, ce are o capacitate de adresare aproape nelimitată [3]. În viitor, capacitatea de stocare și comunicare va fi universală și extrem de distribuită, oamenii, dispozitivele inteligente și mediul înconjurător vor crea un fond comun descentralizat, interconectat printr-o rețea dinamică de sub-rețele. Modalitatea de comunicare se va baza pe protocoale interoperabile, conducând la o cooperare în medii și platforme eterogene. În acest context, *IoT-ul* este un termen generic, toate dispozitivele jucând un rol activ datorită interconectării prin intermediul Internet-ului, cu sau fără fir, iar sistemul per ansamblu căpătând un caracter inteligent prin faptul că asigură accesul la informații și servicii.

*Internet of Things* este o rețea de obiecte fizice care conțin tehnologie embedded cu scopul de a comunica, simți sau interacționa cu starea lor internă sau mediul extern. Inevitabilitatea convenționalității conceptului IoT este ajutată de confluența mai multor factori: creșterea eficienței protocoalelor wireless, dezvoltarea senzorilor și înmulțirea companiilor și start-up-urilor care au acest domeniu de activitate.

Numărul de dispozitive conectate la Internet a depășit populația planetei în 2011, iar în 2020, se preconizează că numărul acestora va ajunge undeva la 25-50 de miliarde. Toate dispozitivele rețelei vor avea o adresă IP și vor putea fi monitorizate. Noțiunea de convergență a rețelelor folosind IP-ul este fundamentală și se bazează pe utilizarea unei rețele de IP-uri comună care să suporte o multitudine de aplicații și servicii. [4]

De cele mai multe ori, se consideră că arhitectură IoT este formată din trei nivele: *Perception layer, Network Layer, Application Layer*. Însă anumiți cercetători [5][6] mai adaugă doua nivele la arhitectura precedentă: *Middleware layer* și *Business layer.* Cele cinci straturi sunt prezentate în figura de mai jos:

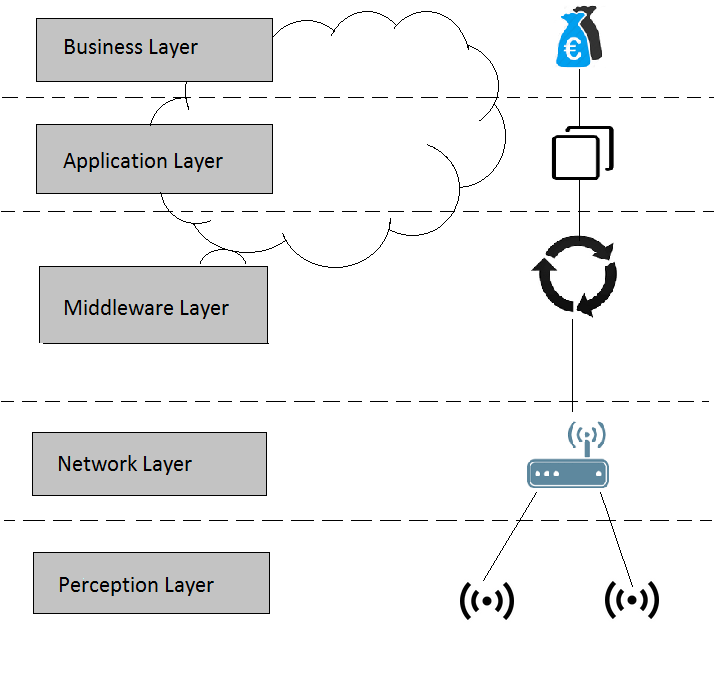


Figura 1. Arhitectura IoT [7]

**Perception Layer**. Este nivelul care se afla la baza arhitecturii IoT-ului. Așa cum sugerează și numele, scopul acestuia este acela de a prelua datele din mediu. Senzorii, cititoarele de codurile de bare, cititoarele de tag-urile RFID, camerele și sistemele de recepție GPS, se află la acest nivel. Principalul scop al acestui nivel este identificare obiectelor și colectarea de date.

**Network Layer.** Este similar nivelelor Rețea și Transport a modelului ISO/OSI. Colectează datele de la nivelul inferior și le trimite mai departe către nivelele superioare. De asemenea, acesta poate să includă numai un gateway, având o interfață conectată la un senzor de rețea și alta la Internet.

**Middleware Layer.** Are ca principal scop managementul serviciilor și stocarea datelor. De asemenea, procesează datele primite iar pe baza rezultatelor obținute ia decizii în mod automat. Toate datele obținute le trimite mai departe la nivelul superior.

**Application Layer.** Primește informațiile de la nivelul *Middleware* și furnizează management-ul global al aplicației, prezentând informația, bazându-se pe informațiile procesate la nivelul inferior. În funcție de tipul dispozitivului, rolul lui în cadrul nivelului *„Perception”*, modul în care au fost procesate datele la nivelul inferior și nevoile utilizatorului, nivelul “Application” prezintă datele într-o multitudine de formate: oraș inteligent, casă inteligentă, sistem de transport inteligent, etc.

**Business Layer.** Se concentrează pe monetizarea serviciului oferit. Datele primite de la nivelul *„Aplication”,* sunt reprocesate, transformându-se în servicii cu o valoare reală. Pe baza acestor servicii putând fi create la rândul lor alte servicii. [7]

Caracteristicele fundamentale ale IoT-ului sunt următoarele:

* Interconectivitatea: În ceea ce privește IoT-ul, orice poate fi conectat la infrastructura globală de comunicarea.
* Servicii legate de dispozitive: IoT-ul este capabil să ofere servicii legate de dispozitive, ținând cont de constrângerile caracteristice fiecărui element din rețea, cum ar fi protecția datelor private și consistența semantică dintre dispozitivele fizice și dispozitivele virtuale asociate lor. Astfel, atât tehnologiile din lumea dispozitivelor fizice cât și tehnologiile din lumea dispozitivelor informatice se vor schimba.
* Eterogenitatea: Acesta este un factor cheie deoarece dispozitivele IoT sunt bazate pe diferite rețele și platforme hardware
* Schimbările dinamice: Stările unui device se schimbă dinamic (e.g. conectat, deconectat) la fel ca și contextul dispozitivelor incluzând locația și viteză. În plus, numărul de dispozitive se poate schimba de-asemenea dinamic.
* La scara largă: Numărul de dispozitive care trebuiesc gestionate și care comunică între ele va fi cel puțin la fel de mare ca și numărul de dispozitive conectate la Internet, în prezent. Numărul de cereri de comunicație inițiate de dispozitive va crește semnificativ comparativ cu numărul de cereri de comunicație inițiate de oameni, înclinând balanța clar în favoarea dispozitivelor. O parte critică constă în gestionarea datelor generate și interpretarea lor pentru a fi folosite în cadrul unei aplicații.

*Internet of Things* nu este o singură tehnologie ci un concept în cadrul căruia, majoritatea noilor elementelor din rețea sunt interconectate, funcționale (e.g. corpuri de iluminat stradal conectate la rețea, senzori embedded, dispozitive capabile să ruleze algoritmi de recunoaștere de imagine, dispozitive capabile de a augmenta realitatea utilizatorilor, dispozitive dotate cu tehnologia NFC) și integrate în servicii care asigură suportul deciziilor situaționale, management-ul bunurilor, ș.a.m.d. . Ținând cont de toate acestea, oportunitățile de afaceri sunt multiple.

Pentru a acomoda diversitatea IoT-ului, există un amestec eterogen de tehnologii de comunicare, ce trebuie să fie adaptate pentru a răspunde la necesitățile aplicațiilor IoT cum ar fi: fiabilitatea, securitatea, eficiența. Exemple de standarde în această categorie includ tehnologii cu și fără fir cum ar fi: Ethernet, Wi-FI, Bluetooh, ZigBee, Z-Wave.

### 2.2 Modele de comunicare

În Martie 2015, Internet Architecture Board (IAB) a publicat un document ce prezintă o arhitectură pentru crearea de rețele de obiecte inteligente (RFC 7452), descriind astfel un framework de patru modele de comunicare utilizate de dispozitivele IoT. Paragrafele ulterioare prezintă acest framework și explică fiecare caracteristică importantă a fiecărui model.

### 2.2.1 Comunicare Device-to-Device

Acest model de comunicare ‘Device-to-Device’ reprezintă două sau mai multe device-uri ce sunt conectate și comunică între ele în mod direct și nu prin intermediul unui intermediar (spre exemplu un server). Aceste dispozitive comunică prin intermediul mai multor tipuri de rețea, inclusiv Internet sau rețele IP. Deseori, aceste dispozitive utilizează următoarele protocoale de comunicare: Bluetooth, Z-Wave, ZigBee.

digrama1

Figura 2. Model de comunicare device-to-device

Această rețea, ‘device-to-device’, permite dispozitivelor să adere la un anumit protocol de comunicare și permite schimbul de mesaje spre a-și îndeplini funcționalitatea. Acest model de comunicare este folosit deseori în aplicații precum case autonome, care folosesc pachete de informații de dimensiuni mici pentru comunicarea între dispozitive. Spre exemplu, în cazul unei locuințe, obiecte precum: becuri, întrerupătoare, termostate, încuietori, etc. transferă cantități mici de informații între ele (e.g. o încuietoare poate trimite un mesaj de stingere a luminilor).

O problemă apare în faptul că protocoalele de comunicare ‘device-to-device’ nu sunt compatibile între ele, acest lucru determinând constrângerea utilizatorul de a achiziționa dispozitive ce folosesc același set de protocoale.

### 2.2.2 Comunicare Device-to-Cloud

În modelul de comunicare ‘device-to-cloud’ dispozitivele IoT sunt conectate la un serviciu din cloud, cum ar fi un furnizor de servicii de aplicații, cu scopul de a transmite date și de a controla traficul de mesaje. Această abordare profită de mecanismele tradiționale de comunicare existente, cum ar fi Ethernet sau Wi-Fi, pentru a stabili o conexiune între dispozitiv și o rețea IP, care se conectează în cele din urmă la serviciile din cloud.

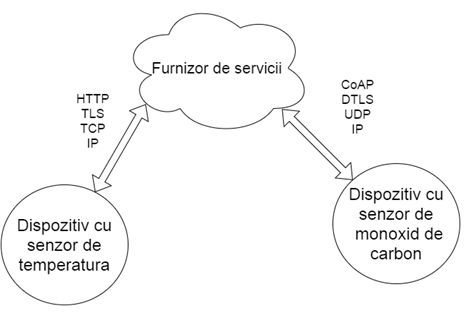


Figura 3. Model de comunicare device-to-cloud

Acest model de comunicare este folosit de o suită de device-uri populare din sfera IoT precum: termostatul inteligent al celor de la Nest Labs [8] și Samsung Smart TV [9]. În cazul celor de la Nest, dispozitivul trimite date către o bază de date din cloud, unde ulterior aceste date pot fi folosite pentru a analiza energia consumată din acea locuința. Mai departe, conexiunea cu cloud-ul permite utilizatorului să obțină acces, de la distantă, asupra termostatului prin intermediul unui smartphone sau a unei interfețe web. Similar, în cazul SmartTV-urilor de la Samsung, aceștia utilizează o conexiune la Internet pentru a transmite informații despre utilizator către Samsung și pentru a permite interacțiunea cu dispozitivul prin intermediul sistemului de recunoaștere a vocii. În ambele cazuri, modelul ‘device-to-cloud’ extinde capabilitățile device-ului.

Cu toate acestea, pot apărea dificultăți în momentul în care integrăm device-uri de la producători diferiți. De cele mai multe ori, device-ul și serviciul din cloud provin de la același distribuitor. Dacă sunt folosite protocoale proprietare între device și serviciu, persoana ce va folosi dispozitivul poate fi legată de un anumit serviciu din cloud, limitându-i-se astfel posibilitatea de a folosi alte servicii de la alți furnizori. De asemenea, utilizatorii pot avea încredere că, în general, dispozitivele concepute pentru această platformă pot fi integrate.

### 2.2.3 Modelul Device-to-Gateway

În cadrul acestui model, care este cunoscut și sub numele *Device-to-Application-Layer Gateway* (ALG)**,** dispozitivele IoT se conectează prin intermediul unui ALG la serviciul din cloud. În termeni mai simpli, „*local gateway”* acționează ca un intermediar între dispozitive și serviciile din cloud. De asemenea, se oferă un grad de securitate și alte funcționalități cum ar fi interpretarea corectă a datelor sau protocolului.

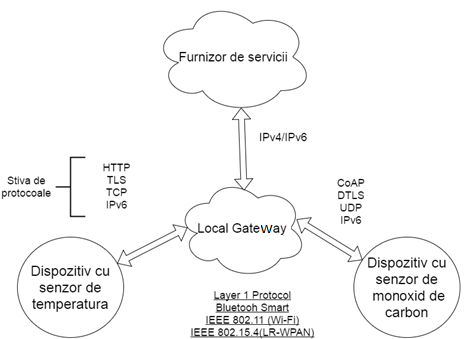


Figura 4. Model de comunicare device-to-gateway

Diferite forme ale acestui model sunt prezente în dispozitivele consumatorilor. Modelul acesta este foarte popular printre dispozitivele pentru fitness. Acestea nu au posibilitatea nativă de a se conecta direct la serviciul din cloud. Prin urmare, de cele mai multe ori se bazează pe o aplicație de pe un smartphone ce va servi drept intermediar pentru a conecta dispozitivul la serviciul din cloud.

O altă formă a acestui model constă în apariția unor dispozitive *„hub”* folosite în cadrul aplicațiilor de automatizare a locuinței. Aceste dispozitive joacă rolul de ‘local gateway’ între dispozitivele IoT și un serviciu din cloud, permițând astfel interoperabilitatea între device-uri (e.g. SmartThings hub este un dispozitiv gateway autonom ce are aparate de transmisiune, Z-Wave și Zigbee, instalate pentru a comunica cu ambele familii de dispozitive). Apoi, dispozitivul se conectează la serviciul SmartThings din cloud, permițând utilizatorului să obțină acces la dispozitive folosind o aplicație pe mobil și o conexiune la Internet.

Cu alte cuvinte, acest model este utilizat frecvent pentru a integra noi dispozitive inteligente într-un sistem care nu este nativ interoperabil cu acestea. Un dezavantaj al acestei abordări constă în faptul că dezvoltarea necesară a gateway-ul sporește complexitatea și costurile sistemului.

Evoluția sistemelor ce utilizează acest model de comunicare și rolul important în abordarea problemelor văzând interoperabilitatea între dispozitive IoT este încă în derulare.

### 2.2.4 Back-End Data-Sharing Model

Acest model prezintă o arhitectură de comunicare ce permite utilizatorilor să exporte și să analizeze datele de la un serviciu din cloud împreună cu date provenite din altă sursă. Acestă abordare vine ca o extensie a modelului *„device-to-cloud”*. O arhitectură de acest tip permite datelor colectate de la un singur dispozitiv IoT să fie agregate și analizate. Deseori, utilizatorii doresc să aibă posibilitatea de a exporta și analiza date în combinație cu datele de la alte surse.

Prin urmare, apare dorință de a acorda accesul la datele furnizate de către senzori, unor terțe persoane. Acest model este cunoscut sub numele de *mashup*, aplicat obiectelor inteligente. De cele mai multe ori constituie un API RESTful [10] proiectat în combinație cu o tehnologie de autentificare și autorizare (e.g OAuth 2.0 [11]). De cele mai mule ori nu există un protocol standard.

Spre exemplu, o persoană ce este responsabilă de un complex de birouri este interesată de analizarea modului în care este consumată energia electrică. În acest scop, ea utilizează datele furnizate de către toți senzorii și sistemul din cloud. În cazul modelului ‘device-to-cloud’ datele pe care fiecare senzor din sistem le furnizează nu pot fi utilizate împreună. Acest model, bazat pe *„back-end data sharing”*, poate să permită companiei să acceseze și să analizeze într-o manieră ușoara date din cloud, respectiv acelea ce au fost produse de către senzorii din clădire. De asemenea, acest tip de arhitectură facilitează nevoia de portabilitate a datelor, permițând utilizatorilor să își mute datele atunci când își schimbă furnizorul de servicii.

Acest model sugerează existența unor servicii cloud de care este nevoie pentru a obține interoperabilitatea în cloud a datelor provenite de la dispozitivele inteligente. [12]

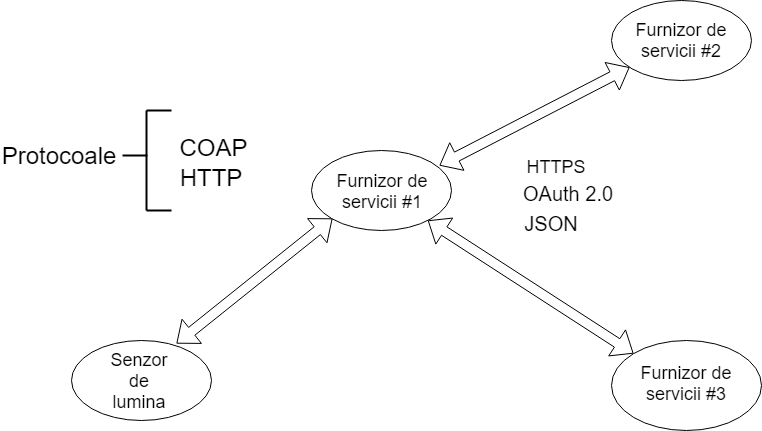


Figura 5. Model de comunicare back-end date-sharing

## 2.3 Cloud Computing

*Cloud Computing* este o paradigmă de calcul, unde un număr foarte mare de sisteme sunt conectate într-o rețea privată sau publică, ce oferă o infrastructură scalabilă pentru aplicații și un mediu de stocare [13]. Datorită apariției acestei tehnologii, costul puterii de calcul, costul de găzduire, de stocare și de livrare de conținut s-a redus semnificativ. Ideea de *cloud computing* se bazează pe un principiu fundamental, acela al reutilizării capabilităților resurselor IT.

Ian Foster definește *cloud*-ul ca fiind: „O paradigmă de calcul distribuit la scară largă a cărei evoluție este condusă de economiile de scară, în care o multitudine de dispozitive cu putere de procesare, dispozitive de stocare, platforme și servicii, toate împărtășind caracterul abstract, virtual, scalabil dinamic și monitorizat, sunt livrate la cerere către clienți externi prin intermediul Internet-ului.”[13]

*Cloud computing*-ul are o varietate de caracteristici, principalele fiind:

* Infrastructura partajată : Se folosește modelul hardware virtualizat, ce permite partajarea resurselor de calcul, stocare și de rețea.
* *On-demand self-service*: Clientul se poate folosi de capabilitățile furnizorului, cum ar fi puterea de calcul și de stocare, fără a necesita interacțiunea umană pentru fiecare furnizor de serviciu.
* Accesul de scara larga la rețea: Capabilitățile hardware și software ale cloud-ului sunt disponibile la cerere prin intermediul rețelei și pot fi accesate de pe diverse platforme (e.g. smartphone, tabletă, laptop)
* Gruparea multiplelor resurse : Resursele de calcul ale furnizorului sunt grupate împreună pentru a servi multipli clienți folosind un model *multi-tenant*, unde diferite resurse fizice și virtuale sunt asignate și reasignate dinamic în concordanță cu cerințele utilizatorilor. Conceptul de *multi-tenant* este una dintre cele mai importante caracteristici a unei aplicații bazate pe cloud.
* Elasticitate: Sistemul are abilitatea de a crește capacitatea atunci când cererea crește și de a o scădea atunci când ea scade. Acesta se bazează pe analiza în timp real a întregului sistem.
* *Managed metering*: Se urmărește gradul de încărcare a sistemului pentru gestionarea, optimizarea serviciului și pentru a oferi informații legate de cost. În acest fel, utilizatorii plătesc doar ce au consumat (pay-per-use) [14]

Serviciile oferite de furnizorii de cloud se pot grupa în trei categorii:

* Infrastructure as a Service (IaaS): Utilizatorul are control asupra sistemul de operare, sistemul de stocare și modalitatea de conectare la rețea însă nu controlează infrastructura cloud.
* Platform as a Service (PaaS): Destinată dezvoltatorilor software, care dezvoltă aplicații în acord cu specificațiile unei platforme, fără a implica factori legați de infrastructura hardware. Consumatorul nu are acces la managementul infrastructurii cloud (rețea, server, sistem de operare sau stocare), însă are control asupra aplicației dezvoltate și asupra eventualelor configurații privind găzduirea aplicației.
* Software as a Service (SaaS): Consumatorului îi este oferită o aplicație completă, ca un serviciu la cerere. Acesta nu deține soft-ul, nu cunoaște detaliile privind infrastructura, nu a făcut investiții în infrastructură, licențe, etc. Este nivelul cel mai vizibil din cloud pentru utilizatorii finali, oferind aplicațiile software expuse ca interfețe Web sau servicii Web (e.g. Gmail este o formă de SaaS oferit de cei de la Google). Serviciul trebuie să fie accesibil de pe orice dispozitiv, de oriunde, în mod continuu, adaptiv la variabilitatea disponibilității resurselor.

În funcție de modelul de *deployment*, *cloud-*urile pot fi grupate astfel:

* *Public Cloud*: Acestea sunt deținute și operate de către părți terțe. Utilizatorului i se oferă posibilitatea de a dezvolta și lansa un serviciu în cloud fără un efort financiar ridicat (în comparație cu cheltuielile de capital aferente aceluiași proces). Este cunoscut sub numele de *external-cloud* sau *multi-tenant cloud* fiind disponibil într-o manieră *pay-per-use.*
* *Private Cloud*: Infrastructura cloud-ului este utilizată pentru o singură organizație iar managementul acesteie poate fi realizată și de o altă organizație. Denumit și *internal-cloud* sau *on-premise cloud*, acesta se bazează pe virtualizarea infrastructurii deja existentă în organizație, acest fapt conducând la utilizarea mai eficientă a resurselor. Acest model implică cheltuieli de capital.
* *Community cloud*: Format din organizații multiple care își partajează infrastructura/serviciile cloud pentru atingerea unui țel comun. Se încearcă asigurarea beneficiilor obținute într-un cloud public (multi-tenant, pay-as-you-go), însă crescând nivelul de *privacy* si securitate la un nivel existent în *cloud*-urile private.
* *Hybrid cloud*: Infrastructura este formată din *cloud*-uri multiple (*private*, *community*, public ) ce rămân entități unice însă sunt legate între ele de tehnologii standardizate sau proiectate care asigură portabilitatea datelor și aplicațiilor. [15]

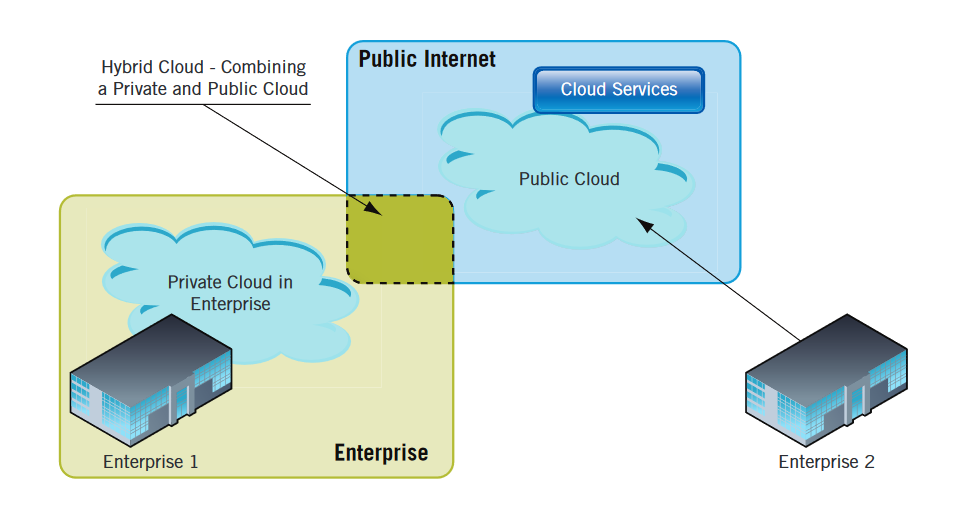


Figura 6. Modele de deployment in Cloud Computing

https://www.dialogic.com/~/media/products/docs/whitepapers/12023-cloud-computing-wp.pdf

## 2.4 Cloud of Things

Ne îndreptăm către Web 3.0, rețeaua de calcul ubicuu. Numărul de dispozitive conectate va crește vertiginos și cantitatea de date va crește în același ritm. Stocarea datelor local nu mai este posibilă. Astfel apare nevoie de închiriere a unui spațiu de stocare. De asemenea, această cantitate imensă de date trebuie folosită ca atare. Datele nu trebuiesc doar procesate pentru a preconiza niște viitoare evenimente, acestea trebuie să se transforme în cunoaștere sau informație care mai departe să formeze cunoaștere. Acesta presupune mai multă putere de procesare, iar acest lucru nu poate fi realizat de dispozitivele IoT, deoarece acestea au specificații hardware foarte reduse. În aceeași ordine de idei, apare nevoie de a închiria capacitate de procesare a datelor. Toate acestea vor fi posibile cu ajutorul cloud computing-ului. [7]

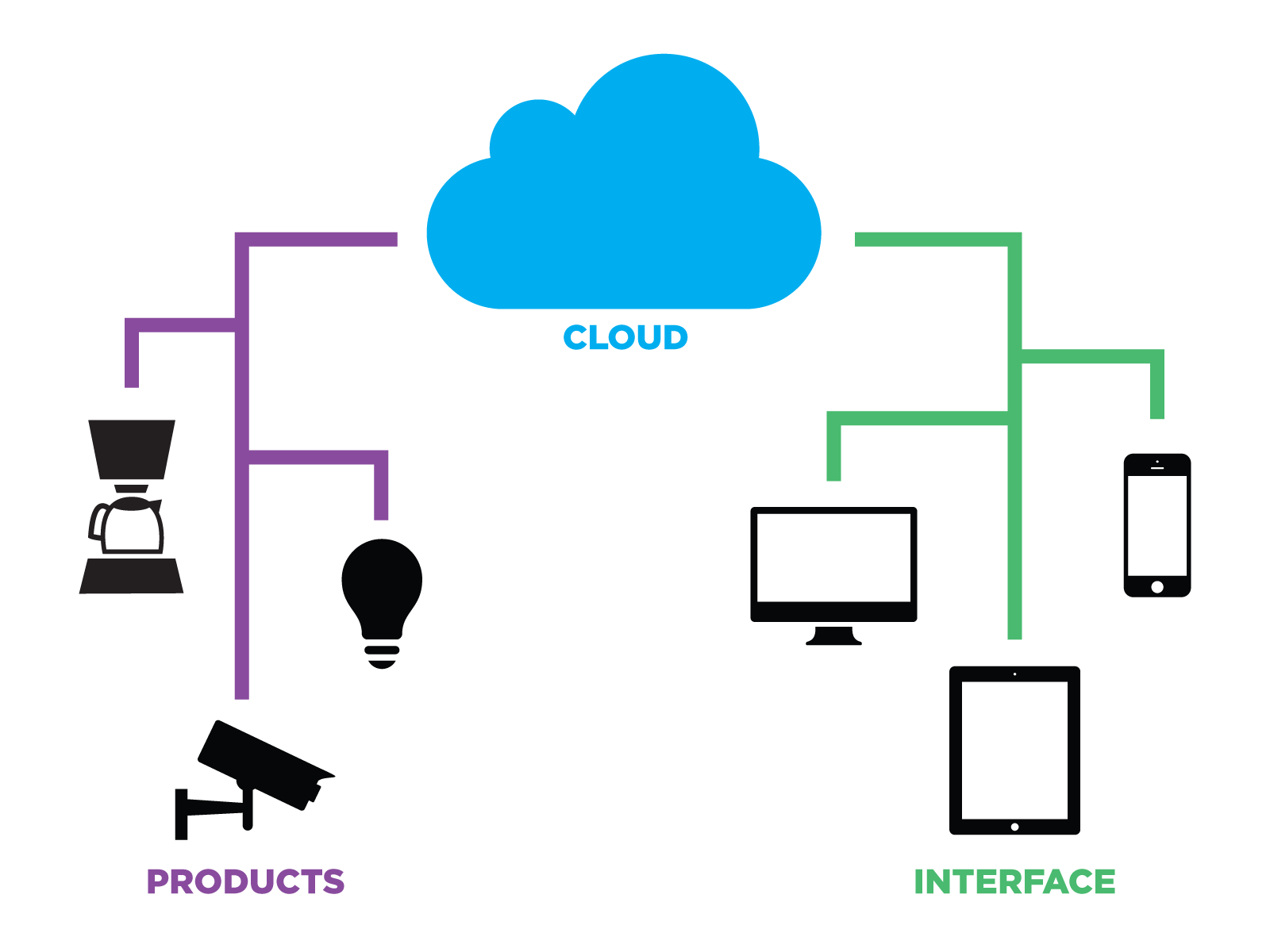


Figura 7. Rolul Cloud-ului in IoT

http://everthingcloud.com/images/cloud-and-things.png

## 2.5 MQTT(Message Queueing Telemetry Transport)

MQTT este un protocol de transport al mesajelor bazat pe modelul publisher/subscriber. Cei care se abonează pentru a primi mesaje (*subscribers*) au posibilitatea de a-și exprima interesul pentru un eveniment sau o serie de evenimente iar ulteriori sunt notificați despre orice eveniment, care corespunde interesului lor, generat de un emițător (*publisher*). Un eveniment este propagat asincron către toți cei care prezintă un interes pentru acest eveniment. În cadrul acestui model, cei care emit mesaje (*publishers*) nu specifică un anumit destinatar (*subscribers*), în schimb, se publică mesaje fără a ști cine s-a abonat pentru a le primi. Similar, receptorii primesc mesajele de care sunt interesați fără a ști cine le-a publicat [16]. Nu necesită multe resurse, este open-source, simplu și este proiectat pentru a fi ușor de implementat. Datorită acestor caracteristici este ideal pentru conexiunile M2M (Machine-to-Machine) care au multiple constrângeri de putere de calcul și resurse, inclusiv capacitate limitată de comunicare. [17]

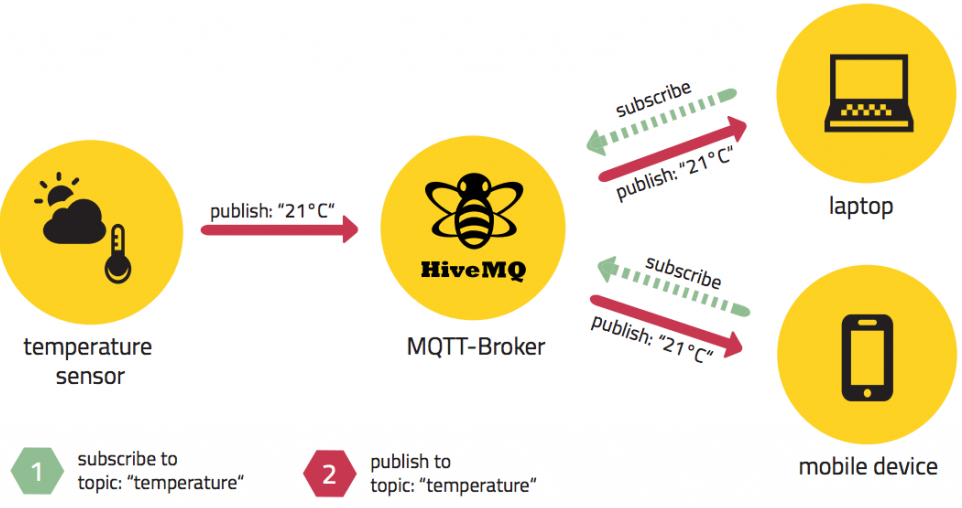
Așa cum am afirmat și mai sus, MQTT este un protocol bazat pe modelul publisher-subscriber ce permite ca dispozitivele de la periferia rețelei să publice către un broker (agent intermediar). Clienții se conectează la broker iar un principal scop al acestuia este să medieze comunicarea dintre două dispozitive. Fiecare dispozitiv are posibilitatea de a se abona la anumite *topic*-uri. În momentul în care un client publică un mesaj la un anumit topic, *broker*-ul transmite mai departe mesajul către tot cei care sunt abonați la acel topic. [18]

Figura 8. Modelul publisher-subscriber in MQTT

http://www.hivemq.com/wp-content/uploads/pub-sub-mqtt-1024x588.png

Acest protocol este bidirecțional și menține starea sesiunii. „Last Will and Statement” (LWT) este o caracteristică utilizată de MQTT pentru a notifica ceilalți clienți că un nod din rețea și-a pierdut conexiunea. Fiecare nod poate specifica un *„Last will message”* (un mesaj MQTT cu un subiect specificat) în momentul în care se conectează la broker. Dacă unul din dispozitivele de la periferia rețelei pierde conexiunea, broker-ul va trimite mesajul la toți clienții ce sunt abonați la topic-ul specificat în mesaj. Mesajul *LWT*  va fi șters în momentul în care primește de la client un mesaj DISCONNECT.[19]

Necesarul redus de resurse și eficiența protocolului MQTT permite monitorizarea și controlul unui volum mare de date. Spre deosebire de alte protocoale bazate pe modelul publisher/subscriber, care tind să suprasatureze conexiunile cu date nealterate, modelul publisher/subscriber folosit de MQTT reduce latența, maximizează lățimea de bandă disponibilă, datorită dimensiunii reduse al unui mesaj MQTT (antet fixat de 2KB), eliminării transmiterii de date nealterate și a modului în care a fost proiectat modelul.[20]

### 2.5.1 Client

Atât cel care publică mesaje cât și cel care se abonează la anumite topic-uri sunt considerați clienți. Aceștia pot îndeplini funcția de *publisher* și *subscriber* simultan. Un client MQTT poate fi orice de la un micro-controller până la un server complet, ce conține o librărie de MQTT și se conectează la un broker prin intermediul oricărei rețele. Pentru a primi mesaje, un clientul trebuie să se aboneze la toate topic-urile ce îl interesează.

### 2.5.2 Broker

Broker-ul se alfă în centrul fiecărui protocol bazat pe modelul *publisher/subscriber*. Acesta poate să gestioneze mii de clienți conectați simultan, acest fapt depinzând de implementare. De asemenea, menține sesiunile de conectare a tuturor clienților inclusiv abonările și mesajele ratate. O altă responsabilitate a acestuia o reprezintă autentificarea și autorizarea clienților. De cele mai multe ori, un broker este de asemenea extensibil, ceea ce permite integrarea facilă a unei autentificări particulare, a unei autorizări sau a unei modificări în sistemul de *back-end*. Integrarea este un aspect esențial deoarece, în cele mai multe cazuri, broker-ul este componenta care este direct expusă accesului prin intermediul Internet-ului, gestionează o multitudine de clienți și apoi trimite mesajele mai departe către sisteme de analiză și procesare. Per total, broker-ul este nodul central prin care trece fiecare mesaj. Ca urmare, este foarte important ca acesta să fie puternic scalabil, capabil de a fi integrat în sistemele de *back-end*, ușor de monitorizat și tolerant la defecte.

Protocolul MQTT se bazează pe TCP/IP iar clientul și broker-ul trebuie sa implementeze această stivă.

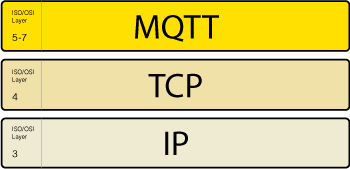


Figura 9. Stiva MQTT TCP IP

**http://www.hivemq.com/wp-content/uploads/mqtt-tcp-ip-stack.png**

O conexiune MQTT se realizează între un client și un broker. Conexiunea este inițiată de către client, trimițând broker-ului un mesaj *CONNECT*. Broker-ul îi răspunde cu *CONNACK* și un cod de stare (*status code*). Din momentul în care conexiunea a fost stabilită, broker-ul o va ține deschisă până va primi o cerere de deconectare din partea clientului sau se va pierde conexiunea. [21]

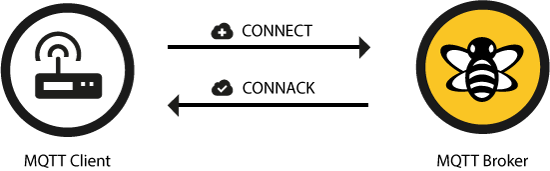


Figura 10. Realizarea conexiunii între un clienti și un broker

http://www.hivemq.com/wp-content/uploads/connect-flow.png

### 2.5.3 Topics

Un *topic* este șir de caractere folosit de broker pentru filtrarea mesajelor fiecărui client conectat. Acesta este format din mai multe *topic level*-uri. Fiecare *topic level* este urmat de către un slash (e.g. “/”). În contextul IoT-ului, utilizarea topic-urilor este mai adecvată, datorită eficienței, în comparație cu o coadă de mesaje. Clientul nu trebuie să creeze topic-ul respectiv înainte să publice un mesaj sau să se aboneze la acel topic datorită faptului că broker-ul acceptă orice topic valid, fără o inițializare anterioară. Topic-urile sunt case-sensitive, ceea ce face ca *bucatarie/lumina* și *Bucatarie/Lumina* să reprezinte două topic-uri diferite.[22]

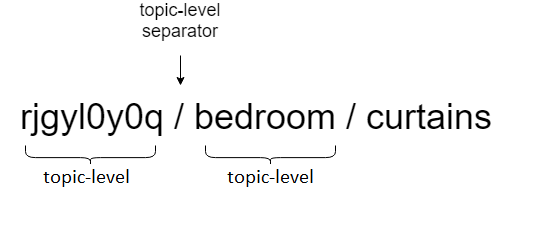


Figura 11. Exemplu de topic MQTT

# 3. Tehnologii folosite

## 3.1 Node.js

Node.js este un mediu multiplatformă, open-source, pentru construirea aplicațiilor web și de rețea. Aplicațiile Node.js sunt scrise în Javascript și pot fi rulate pe platforme precum Windows, OS X, Linux și altele.

Node.js oferă o arhitectură bazată pe evenimente și un I/O API ne-blocant, ce optimizează capacitatea de procesare și scalabilitatea aplicației [10]. Aplicațiile care au la bază Node.js rulează într-un singur fir de execuție, scalabilitatea unui astfel de server este dată de folosirea de *clustere* ce permit utilizarea fiecărui procesor disponibil pe mașină în cauză, pentru a face față numărului ridicat de cereri.

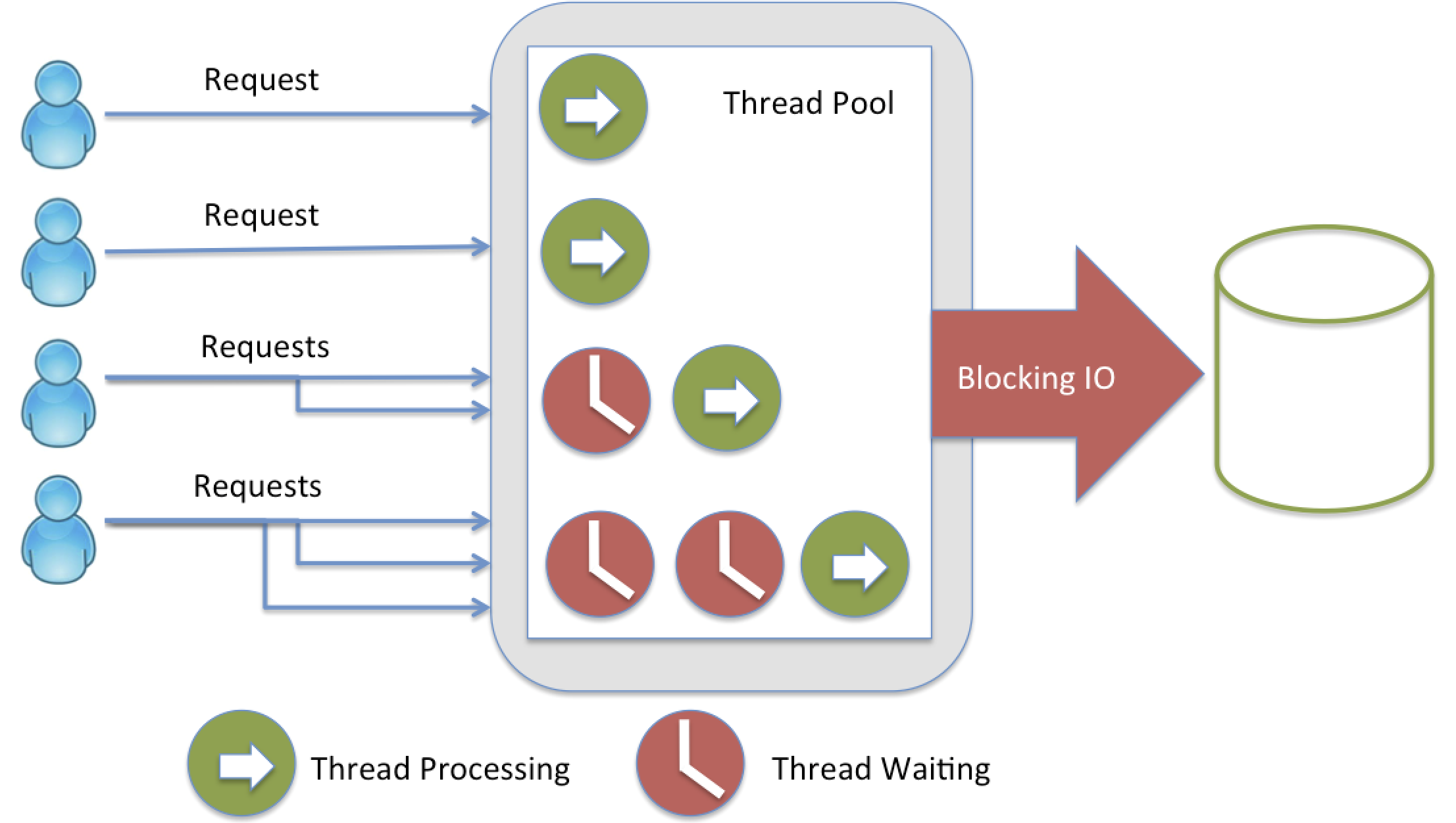


Figura 12.1. Serve Multi-Thread

https://strongloop.com/blog-assets/2014/01/threading\_java.png

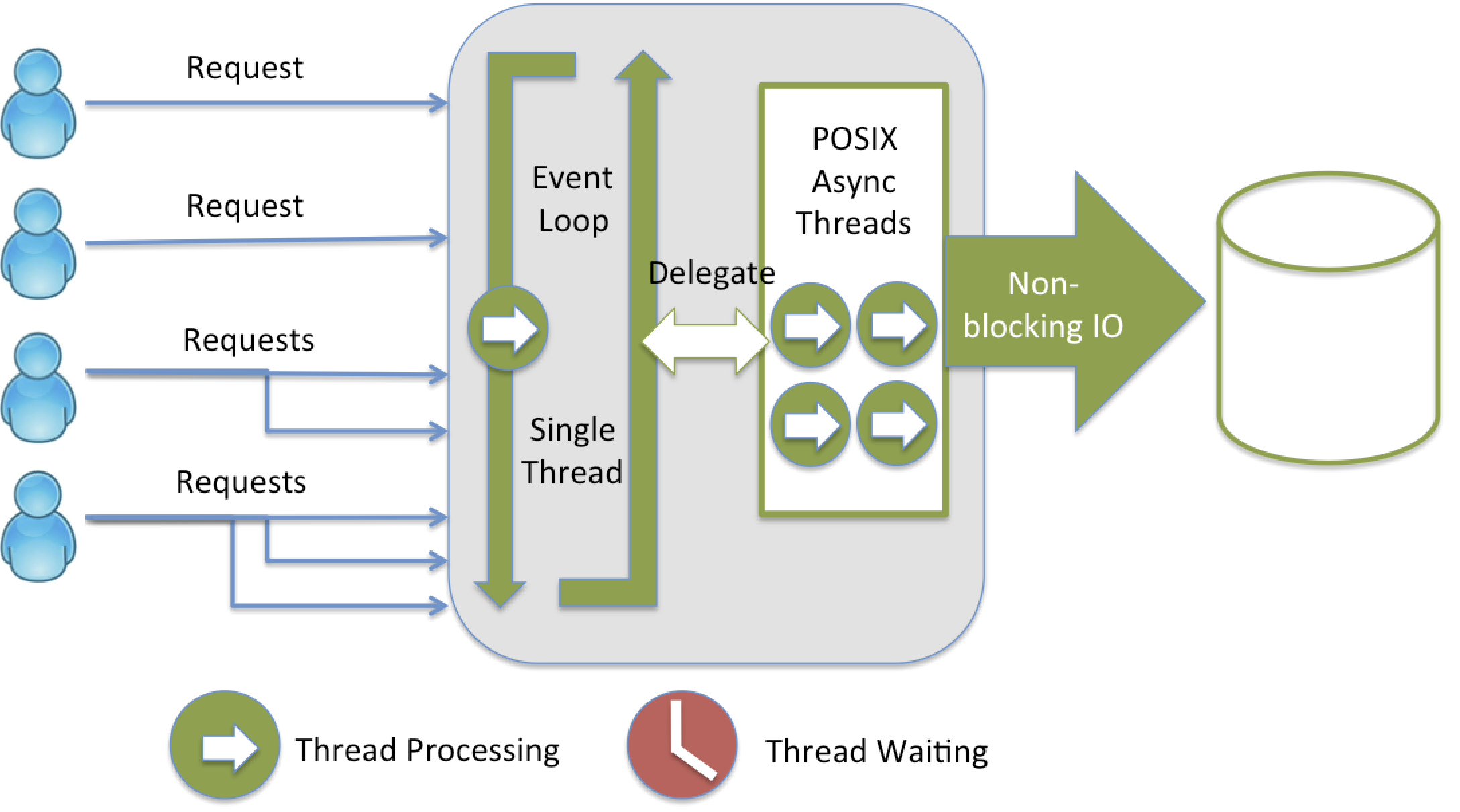


Figura 12.2. Server Node.js

https://strongloop.com/blog-assets/2014/01/threading\_node.png

## 3.2 AngularJS

AngularJS este un *framework* pentru aplicații web, ce facilitează crearea aplicațiilor dinamice (*Single Page Application – SPA)* având la bază o arhitectură MVC (model-view-controller) sau MVVM(model-view-view-model). Se ocupă de manipularea DOM-ului, procesarea apelurilor AJAX și menținerea informațiilor din DOM actualizate. Prin intermediul variabilei „*$scope”* modificările asupra unei structuri de date sunt vizibile peste tot unde aceasta este folosită, astfel fiind actualizată toate aplicația. Una dintre principalele caracteristici o reprezintă abstractizarea elementelor ce stau la baza aplicațiilor de tip *CRUD-Create Read Update Delete,* acestea reprezentând mare parte din aplicațiile disponibile în momentul de față și aducerea unor seturi noi de funcționalități ce conduc la accelerarea procesului de creare al unei aplicații. [23]

## 3.3 MongoDB

MongoDB este o bază de date *open-source*  orientate pe documente, ce face parte din familia sistemelor de baze de date NoSQL. O astfel de bază de date stochează informațiile în colecții, ce pot conține la rândul lor unul sau mai multe documente. Acestea sunt echivalentul tabelelor și înregistrărilor(rândurilor) din cadrul bazelor de date relaționale. Documentele sunt formate din mulțimi de perechi cheie-valoare și constituie elementul de bază în MongoDB.

Ca și celelalte baze de date *NoSQL,* MongoDB permite un design dinamic al schemelor unei colecții, permițând documentelor să aibă câmpuri, structuri și tipuri de date diferite (spre exemplu, două documente din cadrul aceleiași colecții pot conține câmpuri diferite sau un câmp poate avea tipuri de date diferite). Un document poate conține vectori și sub-documente. Documentele utilizează un format numit *BSON(Binary JSON)* ce oferă o reprezentarea binara a unui document *JSON*. BSON este un format de schimb de date folosit în principal ca mod de stocare și de transfer în rețea, în cadrul bazei de date MongoDB. Comparativ cu JSON, BSON este proiectat pentru a fi eficient atât din punct de vedere al dimensiunii spațiului de stocare alocat cât și al vitezei de scanare. Elementele de dimensiuni ridicate dintr-un document BSON sunt prefixate cu un câmp care specifică lungimea valorii următoare, pentru a facilita scanarea. În cazul datelor de dimensiuni reduse, acest format este mai ineficient decât JSON-ul. MongoDB oferă scalabilitate pe orizontală și disponibilitate ridicată prin intermediul replicării datelor și a mecanismului de *sharding*, distribuind datele pe mașini diferite facilitând astfel un număr ridicat de operații asupra unui număr foarte mare de date. Operațiile de join și tranzacțiile nu sunt suportate. Atomicitatea se obține numai la nivelul unui singur document.

În cadrul aplicație “Online Platform for IoT devices” s-a considerat NoSQL (în particular MongoDB) a fi o soluție potrivită, datorită scalabilității oferite în momentul în care volumul de date crește, datorită faptului ca entitățile stocate sunt dinamice, neavând câmpurile prestabilite, datorită diversității senzorilor ce pot fi înregistrați în sistem.

## 3.4 Firebase Cloud Messaging

Firebase Cloud Messaging (FCM) este o soluție oferită de Google, care permite trimiterea de notificări către dispozitivele mobile sau browsere. *Push notification* numit si *server push notification,* constă în trimiterea de informații de la o aplicație către un client fără ca acesta să fi făcut o cerere. Aplicația poate să trimită notificări oricând iar utilizatorul le va primi chiar și atunci când nu interacționează cu aplicația.[24]

Prin intermediul *FCM-ului* se pot trimite doua tipuri de mesaje către client:

* *Notification message*
* *Data message*

„*Notification message”*  este opțiunea cea mai *lightweight*. Nu necesită un număr mare de resurse, având o limită de 2 KB și un set predefinit de perechi cheie-valoare. Acestea sunt afișare în mod automat.

*Data message* permite trimiterea de mesaje până la 4 KB, formate din perechi cheie-valoarea ce pot fi personalizate. Aplicația este responsabilă cu procesarea informațiilor primite.

Pentru a se putea primi notificări, utilizatorul trebuie să permită browser-ului să accepte notificările primite de aplicație. Dacă utilizatorul a acceptat să primească notificări, *browser-ul* va reține această preferință. În caz contrat, de fiecare data când accesează site-ul, utilizatorului i se vor cere aceste permisiuni. FCM nu garantează ordinea de primire a notificărilor. [25]

*Push notification* se bazează pe *service workers* datorită faptului că aceștia rulează în background. Prin urmare, codul acestuia este executat în momentul în care se primește un *Data message* si se interacționează cu notificarea primită. [26]

Procesul de notificare a utilizatorilor se poate observa in imaginea de mai jos (Figura 13).

Am considerat că *Push Notification* este o tehnologie potrivită deoarece se permite notificarea utilizatorilor chiar dacă aceștia nu interacționează cu aplicația în momentul în care s-a produs modificarea stării unui device.

## 

Figura 13. Procesul de notificare a utilizatorilor https://developer.xamarin.com/guides/android/application\_fundamentals/notifications/firebase-cloud-messaging/Images/04-topic-messaging.png

## 3.5 AWS IoT

AWS IoT pune la dispoziție o comunicare securizată și bi-direcțională între dispozitivele conectate la Internet (e.g senzori, actuatori, dispozitive embedded sau electrocasnice inteligențe) și AWS. Acesta permite colectarea datelor de la dispozitive multiple, stocarea și analiza lor. De asemenea, se pot crea aplicații care permit utilizatorilor să controleze dispozitivele prin intermediul *smartphone-ului* sau tabletei.

AWS IoT este compus din următoarele componente:

* *Device gateway*: Permite dispozitivelor să comunice într-un mod sigur și eficient cu AWS IoT
* *Message broker*: Pune la dispoziție un mecanism sigur pentru dispozitive și aplicațiile AWS IoT să publice și să primească mesaje. Se poate folosi protocolul *MQTT* sau *MQTT* împreună cu *WebSocket.*
* *Rules engine*: Permite procesare mesajelor și integrarea cu alte servicii AWS. Se poate folosi un limbaj bazat pe SQL pentru a extrage informații din conținutul mesajului primit și a le trimite la alte servicii cum ar fi: Amazon S3, Amazon DynamoDB și AWS Lambda.
* *Security and Identity service*: Dispozitivele trebuie să aibă grijă de cheile de autentificare astfel încât trimiterea informațiilor către broker să se realizeze într-o manieră securizată. Broker-ul şi *rules engine*-ul utilizează caracteristicile de securitate ale AWS-ului pentru a transmite datele către dispozitive sau alte servicii AWS intr-un mod sigur.
* *Thing shadow*: Constituie un document JSON folosit pentru a stoca și prelua informațiile despre starea curentă a unui dispozitiv.[27]

# 4. Scopurile și cerințele aplicației

## Justificarea necesității aplicației

Aplicația „Online Platform for IoT Devices” își dorește să faciliteze interacțiunea dintre utilizator și dispozitivele inteligente. IoT-ul își propune să aibă același succes ca Internet-ului și să formeze o rețea globală în cadrul căreia fiecare dispozitiv să fie interconectat. *„Internet of Things“* ar trebui mai degrabă să se poartă numele de *„Intranet of Things”* datorită grupurilor izolate de dispozitive ce nu au fost proiectate să comunice între ele. Ne dorim să avem control asupra dispozitivelor, oriunde ne-am afla, nu doar din interiorul rețelei.

Un context în care aplicația este utilă este următorul: o persoană dorește să își transforme locuința într-una inteligentă și în acest sens își cumpără o serie de senzori și actuatori. Își instalează toate dispozitivele achiziționate iar următorul pas constă în construirea unei platforme prin intermediul căreia să poată interacționa cu fiecare *device* în parte. Procesul construirii unei astfel de platforme ar fi unul de lungă durată. De asemenea, o astfel de platformă ar trebui să pună la dispoziția utilizatorului un dashboard format din widget-uri care să aibă posibilitatea să fie personalizate. Nu în ultimul rând, o locuință inteligentă trebuie să permită automatizarea anumitor acțiuni, acestea venind în ajutorul utilizatorului.

## 4.2 Modul în care aplicația rezolvă problema vizată

Un prim pas în realizarea acestei platforme a fost să căutăm soluții deja existente, pentru a ne putea contura scopul și a aduce ceva nou prin intermediul acestui proiect. Deși câteva astfel de platforme au fost deja realizate, acestea nu permit notificarea utilizatorului în momentul în care un dispozitiv și-a schimbat starea/valoarea chiar dacă acesta nu interacționează în acel moment cu aplicația. De asemenea, foarte puține dintre ele permit automatizarea anumitor acțiuni. Pasul următor a constat în determinarea unei soluții pentru al ține la curent pe utilizator cu modificările apărute în sistem și punerea în practică a unui sistem bazat pe reguli pentru a facilita automatizarea unor acțiuni. Interacțiunea dintre platformă și dispozitive se va realiză folosind protocolul MQTT care este special proiectat pentru comunicarea M2M (Machine-to-Machine).

În momentul în care utilizatorul accesează platforma, acesta este întâmpinat de pagina principală, unde îi este sugerat să se înregistreze sau să se conecteze dacă are deja cont. Înregistrarea se realizează prin introducerea unui nume, prenume, email și parolă.

După ce s-a autentificat, utilizatorului i se atribuie un *token* și i se permite interacțiunea cu platforma. Pentru a-și înregistra dispozitivele, utilizatorul trebuie să completeze un formular cu token-ul MQTT aferent *device-ului* și numele sub care dorește să apăra în sistem. Odată adăugate dispozitivele, clientul poate crea widget-uri pentru fiecare *device* în parte, în funcție de informațiile pe care le oferă. Toate aceste widget-uri vor fi afișate într-un *dashboard* ce va putea fi personalizat. Din momentul în care un dispozitiv are asociat un widget, întotdeauna când își modifică valoarea, utilizator va fi notificat iar *dashboard*-ul va fi actualizat. În caz contrar, nu se va primi nici o notificare.

Pentru automatizarea acțiunilor utilizatorului, acesta își poate seta anumite condiții ce vor forma o regulă. O regulă se va activa în momentul în care toate condițiile introduse de utilizator vor fi îndeplinite. Utilizatorul are la dispoziție o serie de opțiuni pentru fiecare regulă: se poate seta un anumit interval în care regula să fie activă (e.g de luni până vineri în intervalul 8:30-20:40), se poate seta o dată la care regulă să se activeze (e.g ora 18:45), se poate seta ca utilizatorul să fie notificat când o regulă este activată. Nu în ultimul rând, utilizatorul își poate seta anumite acțiuni ce vor fi realizate în momentul în care toate condițiile regulii sunt îndeplinite.

# 5. Analiza și proiectarea aplicației

## 5.1 Analiza problemei

### 5.1.1 Problema existentă

*„Internet of Things”* fiind un concept în plină dezvoltare, nu există încă un protocol standard care să fie utilizat pentru comunicarea M2M (Machine-to-Machine). Datorită ne-existenței unui standard și a numărului foarte ridicat de dispozitive pe care IoT-ul dorește să le interconecteze, utilizatorilor nu li se oferă posibilitatea de interacțiune cu dispozitivele inteligente dintr-un punct central. În acest caz, se poate ajunge ca utilizatorul să utilizeze câte o aplicație pentru fiecare device în parte.

### 5.1.2 Identificarea soluției

În soluția pe care ne propunem să o implementăm, dorim să oferim utilizatorilor posibilitatea de interacțiune cu dispozitivele inteligente, de oriunde s-ar afla, prin intermediul unei interfețe *web*, de a fi ținuți la curent cu starea întregului sistem și de a fi notificați în momentul în care un senzor/actuator își modifică starea. De asemenea, cum un mediu inteligent ar trebui să permită automatizarea unor anumite acțiuni, o posibilă soluție ar fi utilizarea unui sistem bazat pe reguli.

## 5.2 Proiectarea aplicației

### 5.2.1 Cerințele aplicației

Înainte de proiectarea propriu-zisă, este necesară prezentarea cerințelor pe care aplicația în cauza le va îndeplini. În continuare, aceste vor fi descrise într-un mod schematic:

* Cerințe funcționale:
  + Interacțiunea cu dispozitivele inteligente
  + Utilizarea platformei in diferite scopuri, scenarii (Exemplu: Smart House, Smart Hotel, Smart Parking, etc.)
  + Vizualizarea în timp real a dispozitivelor înregistrate în cadrul platformei și a stării acestora
  + Notificarea utilizatorilor in momentul în care un senzor/actuator și-a modificat starea, în orice situație (e.g. chiar dacă utilizatorii nu interacționează cu aplicația)
  + Automatizarea anumitor acțiuni ale utilizatorilor
  + Furnizarea de statistici referitoare la valorile provenite din partea dispozitivelor
  + Utilizarea acestei platforme atât de către un om cât și de către o mașina (interconectarea sistemelor)
* Cerințe privind interactivitatea:
  + Vizualizarea informațiilor provenite de la fiecare dispozitiv
  + Interacțiune cu actuatorii
  + Oferirea utilizatorului a unei interfețe fiabile care să fie accesibilă de pe orice tip de dispozitiv (computer, laptop, telefon, tabletă)
  + Utilizatorul va fi notificat în cazul în care valoarea unui senzor/actuator se va modifica
* Cerințe de securitate:
  + Datele fiecărui utilizator vor fi confidențiale
  + Existența unui mecanism de autentificare pentru a stabili identitatea fiecărui utilizator și pentru a împiedica accesul la dispozitivele care nu îi aparțin

### 5.2.2 Arhitectura generala

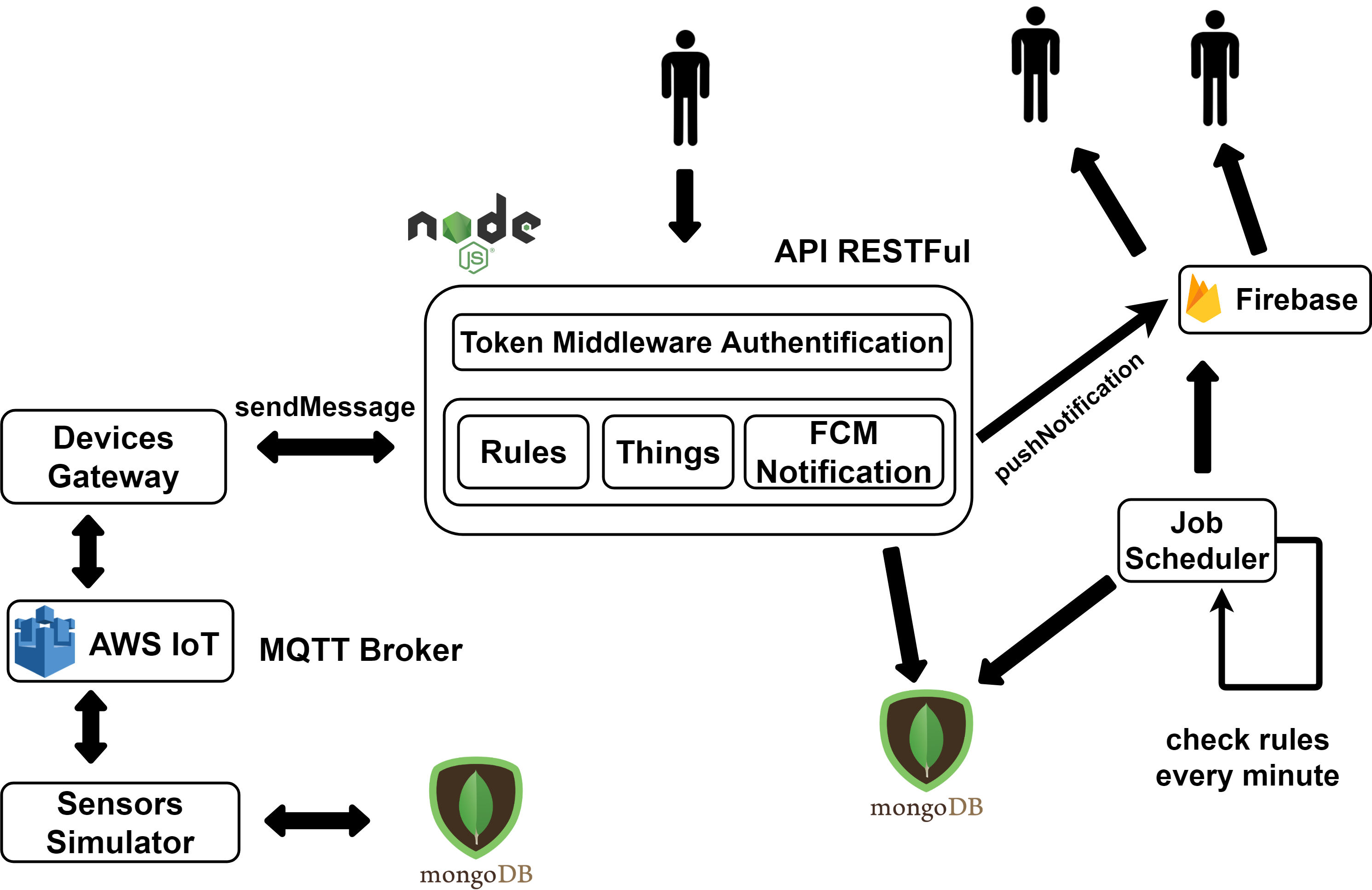
Un obiectiv general al acestei platforme constă în rezolvarea problemei interoperabilității dintre dispozitive și oferirea unui punct central prin intermediul căruia se facilitează accesul la acestea. Sistemul astfel creat oferă utilizatorilor posibilitatea de interacțiune cu dispozitivele, oriunde s-ar afla. De asemenea, li se permite crearea de scenarii care vor conduce la automatizarea anumitor acțiuni. Platforma va oferi utilizatorului posibilitatea de a adaugă dispozitive în sistem fără a-și face griji în legătură cu interoperabilitatea cu celelalte părți deja existente. În momentul conectării, fiecare dispozitiv trebuie să furnizeze serverului informații despre sine cum ar fi : atributele/proprietățile sale sub formă unor perechi <cheie, valoare>. Interacțiunea cu senzorii va fi simulată prin intermediul unei interfețe web. În continuare, puteți vedea o imagine generală a arhitecturii pe care ne propunem să o implementăm.

Figura 14. Arhitectura generală

### 5.2.3 Things

Principalul scop al acestui modul îl reprezintă interacțiunea cu dispozitivele din sistem, managementul acestora (adăugare/eliminare) și vizualizarea informațiilor aferente. Pentru adăugarea unui dispozitiv în cadrul platformei, utilizatorul va fi nevoit să introducă topic-ul MQTT și numele dispozitivului. Odată introduse, utilizatorul va avea acces la informații precum: starea curentă a dispozitivului, statistici referitoare la ultimele valori înregistrate, istoricul valorilor.

Procesul de funcționare al acestui modul este ilustrat în diagrama de secvența de mai jos:

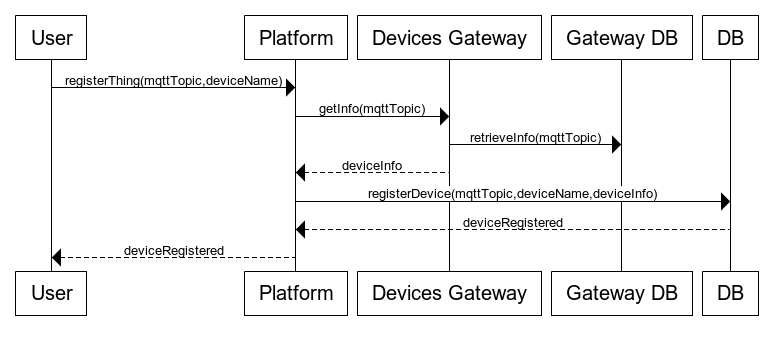


Figura 15.1. Adăugarea unui dispozitiv în sistem

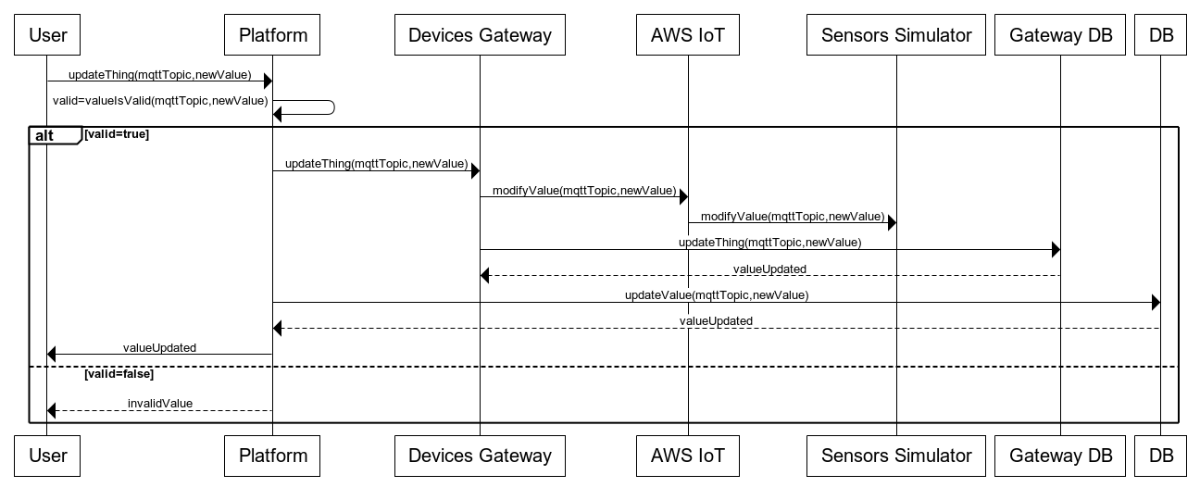


Figura 15.2 Modificarea valorii unui actuator

### 5.2.4 Rules

Acest modul se concentrează pe procesul de automatizare a acțiunilor utilizatorului, având la bază un sistem bazat pe reguli. Utilizatorul va avea posibilitatea de a seta anumite condiții ce vor declanșa regula, poate stabili un interval în care aceasta să fie activă sau o anumită oră la care regula să fie activată. Înainte de a fi introdusă în sistem, regula va fi verificată pentru a nu fi în conflict cu ea însăși sau cu alte reguli. De asemenea, fiecare regulă poate fi modificată și se oferă un istoric al acesteia, furnizându-se informații despre momentul în care a fost activată și ce a cauzat activarea ei. Tot în cadrul acestui modul, se verifică dacă o regula a fost activată, în momentul în care un dispozitiv și-a schimbat starea.

Procesul de adăugare a unei reguli este descris succint în diagrama de secvență de mai jos:

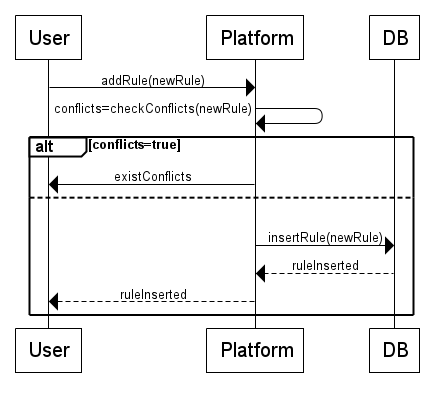


Figura 16. Adăugarea unei reguli în sistem

### 5.2.5 FCM Notifications

În cadrul acestui modul se inițiază notificarea clientului în momentul în care un device și-a modificat starea ori o regulă a fost activată. Deoarece este necesară existența unui mecanism de push notification, avem de ales între două opțiuni: *Web Sockets* și *HTML5 Push Notifications*. Datorită faptului că in dezvoltarea aplicației s-a optat pentru o arhitectură de tip *REST*, o soluție adecvată ar fi utilizarea API-ului “*PushNotifications”,* ce a apărut odată cu *HTML5* [10]. Optând pentru această soluție, i se permite astfel utilizatorului să primească notificări chiar și în momentul în care acesta nu interacționează cu aplicația.

Mecanismul de funcționare a acestui modul se poate observă în diagrama de secvența de mai jos:



Figura 17. Notifications Workflow

### 5.2.6 Job Scheduler

Constituie un daemon care rulează la un interval regulat de timp (un minut) ce parcurge toate regulile ce au setat un timp de activare și verifică dacă există vreo regulă ce s-a activat și realizează operațiile aferente.

### 5.2.7 Baza de date

Ca mediu de stocare s-a folosit o bază de date NoSQL, MongoDB, datorită caracteristicelor acesteia, a performanțelor oferite, fiind o soluție potrivită pentru acest tip de aplicație datorită datelor nestructurate din partea senzorilor/actuatorilor. Pentru interacțiunea cu baza de date s-a utilizat *Mongoose,* un ODM (*Object-Document-Mapping).* Fiecare colecţie din MongoDB se mapează ca o schemă, în cadrul căreia se pot modela conținutul documentelor. Fiecare schemă este utilizată prin intermediul unui model. [28]

Un exemplu de document aferent unui utilizator se poate observa in figura de mai jos.

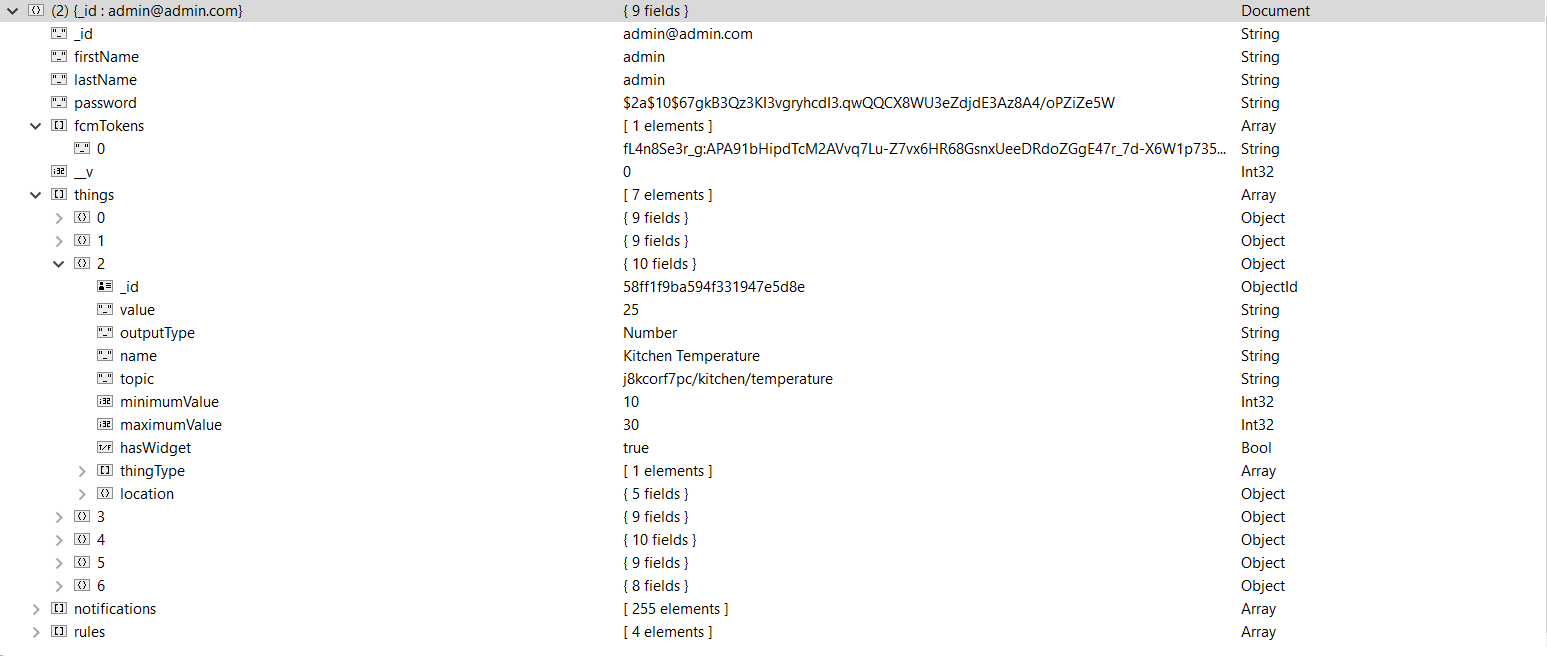


Figura 18. Exemplu de document aferent unui utilizator

Fiecare document este identificat prin intermediul câmpului *\_id*. Baza de date contine trei colecții: *users*, *ruleHistory*, *thingHistory*. Structura datelor în interiorul bazei de date este următoarea:

*Users -* \_id (email) , firstName, lastName, password, fcmToken, things(\_id, value, outputType, name, topic, hasWidget, location, thingType, notifications) (date, message, seen), rules(\_id, name, rule, devicesInvolved, active, ruleDescription)

*RuleHistory -* \_id (email-ul utilizatorului), rules (\_id, name, history (\_id, value, thing, date))

*ThingHistory -*  \_id (topic-ul MQTT al dispozitivului) , values (\_id,date,value)

Structura unui document poate să difere de la un utilizator la altul deoarece unii dintre ei au dispozitive înregistrate alții nu, unii au reguli iar alții nu. De asemenea, diferă și datorită informațiilor furnizate de către senzori (e.g. unele dispozitive furnizează valori numerice, altele furnizează informații sub forma unui șir de caractere, etc ).

### 5.2.8 Sensors Simulator

Această parte a aplicației are ca și scop simularea device-urilor. Acest modul este necesar deoarece în cadrul proiectului nu se realizează nici o interacțiune cu vreun dispozitiv real. Prin urmare, *Sensors Simulation* a fost dezvoltat pentru a simula o interacțiune cât mai reală între platformă și o suită de dispozitive. Comunicarea dintre dispozitivele virtuale și platformă se realizează cu ajutorul unui intermediar: „*Devices Gateway”*. Datorita faptului că device-urile sunt simulate, *Sensors Simulator* și *Devices Gateway*  constituie aceeași entitate.Dispozitivele vor transmite un mesaj MQTT către *Devices Gateway* folosind AWS IoT, cel din urma îndeplinind rolul de broker MQTT. Fiecare dispozitive v-a fi identificat în mod unic prin intermediul topic-ului MQTT aferent. Toate mesajele din partea dispozitivelor vor ajunge la *Devices Gateway* și vor fi trimise mai departe catre OPID. Persistența datelor este asigurată prin intermediul unei baze de date NoSQL. În momentul în care un utilizator modifică starea unui actuator, platforma va trimite nouă valoare către *Devices Gateway* iar acesta va transmite mesajul către device, folosindu-se de topic-ul MQTT. Pentru a receptiona mesajele, fiecare dispozitiv va trebui să se aboneze la topic-ul MQTT aferent. Astfel este simulată o interacțiune reală între un device și platformă în cauză. Într-un caz real, *Sensors Simulator* și *Devices Gateway* nu vor mai reprezenta aceeași entitate. Dispozitivul fizic va prelua atribuțiile simulatorului iar *gateway-ul* își va păstra rolul.

Interacțiunea dintre aceasta și platformă este descrisă în diagrama de mai jos:

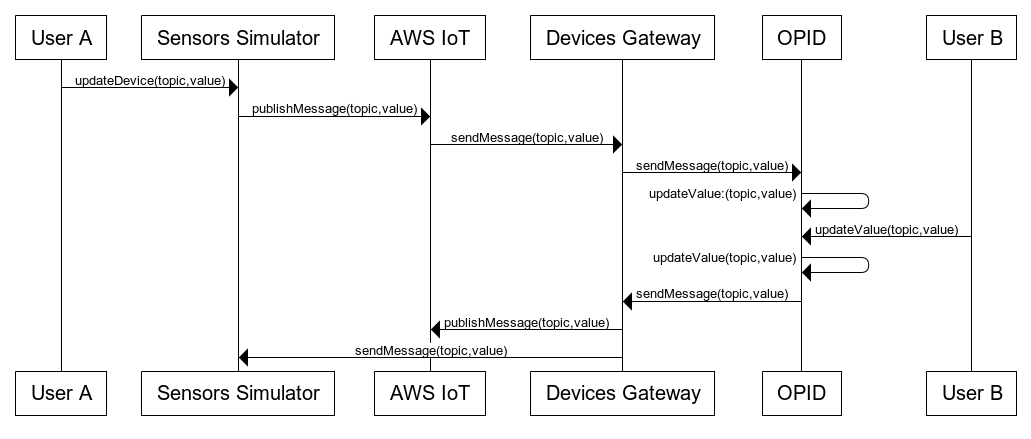


Figura 19. Interactiunea Dispozitiv <-> Utilizator

### 5.2.9 Devices Gateway

Acest modul este utilizat pentru a intermedia interacțiunea dintre utilizator și dispozitive. *Gateway*-ul se abonează la *topic*-ul la care dispozitivele își publică datele pentru a recepționa mesajele transmise de acestea. În momentul în care se dorește simularea unui nou device va fi necesară completarea unui formular cu informațiile aferente dispozitivului ( e.g. topic-ul MQTT, tipul dispozitivului, drepturi de scriere/citire,etc.). Datele vor fi stocate într-o bază de date pentru a fi utilizate în momentul în care platforma dorește să obțină informații despre dispozitivul pe care utilizatorul dorește să îl adauge în cadrul platformei.

# 6. Implementare

## Detalii tehnologice

Pentru a menține gradul complexității aplicației la un nivel scăzut, s-a mers pe principiul unei aplicații modulare. Comunicarea clientului cu serverul web se realizează prin intermediul unui REST API implementat în Node.js, utilizând framework-ul Express, ce interacționează cu baza de date MongoDB, prin intermediului ODM-ului Mongoose.

Aplicația este găzduită de cei de la *Cloud9* [29], aceștia oferind un mediu de dezvoltare constând dintr-o mașină virtuală cu un nucleu, 512 MB RAM și 2GB memorie.

Datorită faptului că s-a optat pentru o arhitectură REST, trei aspect au fost și sunt foarte importante în dezvoltarea acestei platforme:

* serviciul conține un URI de bază (identificarea resurselor se face pornind de la acest URI de bază – fiecare resursă având un identificator unic)
* tipurile de date suportate de serviciu fac de obicei parte din clasa tipurilor MIME
* setul de operații permise de către serviciu sunt aceleași cu metodele expuse de protocolul HTTP (POST,GET,PUT,DELETE,PATCH) [10]

Un exemplu de cerere HTTP ar fi următoarea:

*https://opid-server-cypmaster14.c9users.io/api/things/location*

Principalele resurse modelate de acest API sunt următoarele:

* /api/things
* /api/subscriptions
* /api/rules
* /api/notifications
* /api/FCMToken

Majoritatea resurselor modelate expun operații de tip CRUD (Create, Read, Update, Delete) echivalentul metodelor *POST, GET, PUT,DELETE*. De exemplu, metodele folosite pentru resursa */api/things* sunt:

*POST /api/things*

* Se folosește pentru adăugarea de dispozitive în sistem. Informațiile despre noul dispozitiv(*topic*-ul MQTT,numele) trebuie să se afle în corpul cererii. Dacă aceste date nu sunt prezente se va returna un mesaj cu codul de stare 400 (*Bad Request*). În cazul în care dispozitivul a fost înregistrat deja utilizatorul va primi un răspuns cu codul de stare 409 (*Conflict*). Dacă nu se găsesc informațiile suplimentare despre dispozitiv, clientului i se va răspunde cu un mesaj corespunzător având codul de stare 404 (*Not Found*). Dacă nu s-a produs nici o eroare, codul de stare returnat clientului va fi 201 (*Created*).

GET */api/thing*

* Se folosește pentru a obține lista de dispozitive ale unui utilizator. Aceștia vor fi identificați prin intermediul token-ului ce este trimis în cadrul fiecărei cereri HTTP. Dacă interogarea bazei de date se realizează cu succes, utilizatorul v-a primi răspunsul sub forma unui JSON, codul de stare fiind 200 (*OK*). Dacă se produce o eroare, codul de stare returnat va fi 500 (*Internal Server Error*).

GET /api/thing/:topic/history?limit=

* Se folosește pentru a obține istoricul valorilor înregistrare de un dispozitiv. Numărul de înregistrări trimise clientului poate fi limitat prin intermediul parametrului *limit*. În cazul în care nu se produce nici o eroare, informațiile dorite vor fi transmise sub forma unui JSON în corpul răspunsului, având codul de stare 200.

DELETE */api/things/:topic*

* Astfel se elimină un dispozitiv din cadrul platformei. Topic-ul senzorului/actuatorului este specificat în *URI*. Dacă acesta nu este transmis, clientului îi va fi transmis un răspuns având codul de stare 400. În momentul în care eliminarea din sistem s-a realizat cu succes, codul de starea al răspunsului va fi 200.

PATCH */api/things/:topic*

* Metoda *PATCH* va fi folosită pentru actualizarea valorii unui dispozitiv. Noua starea a actuatorului se va afla în corpul cererii HTTP. Dacă valoarea a fost actualizată cu succes, clientului îi va fi transmis un răspuns având codul de stare 200. Odată actualizată valoarea are loc și evaluarea regulilor active la acel moment.

Pentru resursa *„/api/rules”*, metodele utilizate sunt:

GET */api/rules/:ruleName*

* Se utilizează pentru a obține informațiile despre o anumită regulă. În cazul în care nu se produce nici o eroare și numele regulii a fost specificat în URI, clientul va primi informațiile dorite sub forma unui JSON

POST */api/rules*

* Se folosește pentru adăugarea unei reguli în sistem. Informațiile despre conținutul regulii trebuie să se găsească în cadrul cererii HTTP. Regula ce se dorește a fi introdusă se verifică dacă este în conflict cu ea însăși sau cu alte reguli din sistem. Dacă se găsește un astfel de conflic, utilizatorului i se va răspunde cu un mesaj corespunzător, având codul de eroare 409 (Conflicts). Dacă nu s-a produs nici o eroare și nu există nici un conflict, clientului i se va răspunde cu un mesaj având codul de stare 201 (Created).

PUT */api/rules/:ruleName*

* Metoda PUT va fi utilizată în cadrul procesului de actualizarea a unei reguli. Numele regulii ce se dorește a fi actualizată este specificată în URI iar noul conținut al regulii trebuie să se găsească în cadrul cererii HTTP. . Dacă vreuna dintre aceste două date lipsește, clientului i se va transmite un răspuns având codul de stare 400.

DELETE /api/rules/:ruleName

* Metodă DELETE va fi utilizată în cadrul procesului de eliminare a unei reguli. Numele regulii este specificat în URI. Dacă acesta nu va fi transmis, clientului i se va transmite un răspuns având codul de stare 400. În momentul în care eliminarea din sistem s-a realizat cu succes, codul de starea al răspunsului va fi 200.

PATCH /api/rules/:ruleName

* Se folosește pentru actualizarea status-ului unei reguli, pentru trecerea din starea activă în cea inactivă sau invers. Numele regulii, pentru care se dorește modificarea stării, trebuie să fie specificat în URI.În caz contrar, clientului i se va trimite un mesaj de eroare având codul de stare 400.

GET /api/rules/:ruleName/history

* Este utilizat pentru a se obține istoricul unei reguli. Aceste informații vor fi afișate în tab-ul „History” în cadrul paginii ce va conține informațiile despre o anumite regulă. Numele regulii trebuie să fie specificat în URI. În caz contrar, clientului i se va trimite un mesaj de eroare având codul de stare 400.

În cazul în care pentru o resursă modelată de API nu s-au furnizat toate datele necesare, clientul va primii un răspuns având codul de stare 400 (*Bad Request*). Datele trimise dinspre client spre server vor fi în format JSON iar răspunsul serverului la cererile clientului vor fi tot în același format.

Pentru a implementa efectiv serviciu REST, s-a folosit un mecanism oferit cei de la Express: *express.Router*. O instanță a aceste clase reprezintă un *middleware* complet. De asemenea, prin intermediul acestei clase, mecanismul de rutare pentru o resursă se realizează cu ușurință și poate fi extins în orice moment.

Mecanismul de rutare pentru resursa *things*, prin intermediul căreia se realizează managementul resurselor din sistem, se realizează în felul următor:

router.post("/", function (req, res, next) {});

router.get("/", function (req, res, next) {});

router.get("/:topic/history?limit=", function (req, res, next) {});

router.delete("/:topic", function (req, res, next) {});

router.patch("/:topic", function (req, res, next) {});

router.put("/location", function (req, res, next) {});

module.exports = router;

Această resursă este utilizate în cadrul serverului astfel:

const things = require("./Routes/things");

const app = express();

app.use("/api/things", things);

În același mod se realizează și managementul celorlalte resurse ale API-ului. După cum se poate observa mai sus, serviciul a fost dezvoltat într-o manieră modulară, fiind ușor de extins.

Pentru securizarea anumitor resurse ale API-ului și autentificarea clienților s-a utilizat JWT (JSON Web Tokens) [30]. Utilizatorul obține acest token în momentul în care autentificarea a fost realizată cu succes. Autenticitatea și proveniența token-ului poate fi verificată prin intermediul semnăturii pe care o deține. Token-ului îi va fi aplicată o semnătură digitală pentru a dovedi autenticitatea sa. Pentru aplicarea semnăturii digitale a fost utilizat algoritmul de criptare RSA. Se generează o pereche de chei, una privată și una publică. Token-ul este semnat folosind cheia privată iar semnătură se verifică cu ajutorul cheii publice. Astfel se asigură faptul că nimeni altcineva, în afară de autoritatea ce deține cheia privată, nu va putea furniza semnături valide. În momentul în care token-ul expiră, utilizatorul trebuie să reia procesul de autentificare pentru a-i fi atribuit un nou token.

Pentru a avea acces la resursele protejate ale API-ului , clientul trebuie să trimită token-ul în cadrul fiecărei cererii HTTP. Token-ul va fi căutat în *cookie*-uri, în conținutul cererii sau în antetul cereri. Pentru a restricționa accesul persoanelor neautorizate, se verifică dacă semnătura token-ului a fost aplicată de către acest serviciu și de asemenea se verifică dacă token-ul a expirat sau nu. În cazul în care validarea token-ului nu s-a realizat cu succes, utilizatorii vor fi redirecționați către pagina de *log-in.*

La baza automatizării acțiunilor utilizatorului se află un sistem bazat pe reguli. Pentru realizarea acestei caracteristici a sistemului, s-a utilizat pachetul *JSON-Rules-Engine* [31]. În cadrul acestui pachet, regulile au structură unui JSON, fiind ușor de citit și înțeles. Datorită acestei structuri, regulile pot fi transformate cu ușurință într-un JSON și stocate în interiorul unei baze de date. Formatul unui astfel de structuri se poate observa în fragmentul de mai jos:

{

"name": "Complex rule",

"rule": {

"conditions": {

"any": [

{

"all": [

{

"fact": "Bathroom Fan",

"operator": "equal",

"value": "Off"

},

{

"fact": "Bathroom Light",

"operator": "equal",

"value": "Off"

}

]

},

{

"all": [

{

"fact": "Attic Temperature",

"operator": "equal",

"value": 22

},

{

"fact": "Garage Light",

"operator": "equal",

"value": "On"

}

]

}

]

},

"event": {

"do": {

"Garage Light": "Off"

},

"type": "message",

"sendNotification": true,

"sendSms": false

},

"triggerInterval": {

"start": {

"time": "Thu, 01 Jan 1970 05:30:00 GMT",

"date": "1" // Duminica 0 => Sambata 6

},

"end": {

"time": "Wed, 31 Dec 1969 07:55:00 GMT",

"date": "5"

}

},

"triggerTime": "Wed, 31 Dec 1969 06:35:00 GMT",

"ruleDescription": "Complex rule description",

"active": true,

"devicesInvolved": [

"Bathroom Fan",

"Bathroom Light",

"Attic Temperature",

"Garage Light"

]

}

Valoare cheii *„conditions”* reprezintă expresia logică ce a fost introdusă de utilizator ( (Bathroom Fan == Off And Bathroom Light == Off ) Or (Attic Temperature == 22 And Garage Light == On) ). Obiectul asociat cheii „*event*” constituie evenimentele ulterioare momentului în care regula a fost evaluată cu succes (e.g. va fi modificată valoarea dispozitivului *„Garage Light* ” și se va trimite un mesaj de notificare). Se observă că regula are un interval în care aceasta să fie activă (Luni 05:30 GMT – Vineri 07:55 GMT) și o dată la care să se evalueze în mod automat ( 06:35 GMT).

De asemenea, trebuiesc definite datele ce vor fi folosite de *engine* pentru evaluarea unei reguli. Un astfel de set de date se pot observa în fragmentul de mai jos:

{

"Back door":"Locked",

"Garage Light":"On"

}

Regula introdusă de utilizator va fi transformată într-o expresie booleana, din care se va obține arborele asociat formulei, care va fi asociat cu valoarea cheii *conditions* din JSON-ul ce prezintă structura unei reguli. Abordarea în care se realizează evaluarea regulii este una bazată pe Promises. Următoarea porțiune evaluează expresia și verifică dacă regulă a fost activată:

function checkRule(data, facts, next) {

let engine = new Engine();

let rule = new Rule(data);

engine.addRule(rule);

engine

.run(facts)

.then(triggeredEvents => {

if (triggeredEvents.length > 0) {

next("Regula a fost activata");

} else {

next(null);

}

})

.catch(reason => {

logger.error("Failed because:", reason);

next(reason);

});

}

Înainte de adăugarea unei reguli în sistem, aceasta trece printr-un proces de verificare  pentru a identifica dacă se află în conflict cu ea însăși sau cu alte reguli (e.g. o altă regulă are ca și consecință închiderea ușii, iar nouă regulă are ca și consecință deschiderea ușii). În cazul în care există un astfel de conflict, utilizatorul va fi notificat pentru a modifica conținutul regulii.

Modalitatea de verificare a existenței unui conflict în interiorului unei reguli ce se dorește a se introduce în sistem, se realizează în modul următor:

* se obține configurația curentă a dispozitivelor și se modifică astfel încât regula să se poată activa
* se aplică acțiunile ce trebuiesc realizate în momentul în care regula se activează
* se verifică dacă regula se poate aplica sau nu
* dacă regula se poate activa, atunci avem un conflict, altfel regula este introdusă în sistem

În momentul în care un senzor/actuator își modifică starea, sistemul verifică toate regulile ce sunt active și au în componența lor dispozitivul în cauza. Apoi, acestea sunt evaluate pentru a se vedea dacă s-a activat vreuna. O regulă se poate afla în două stări: activă sau inactivă. În momentul în care se află în starea inactivă, acesta va fi omisă în procesul de verificare.

De asemenea, se oferă posibilitatea de a avea reguli care să se activeze la un moment dat și într-un anumit interval de timp. În acest sens, s-a utilizat pachetul *node-schedule* [32] care permite executarea unor funcții la un interval regulat de timp. În acest sens, s-a creat un program care va rula din minut în minut pentru a verifica dacă există reguli care trebuie evaluate în acel moment.

Pentru expedierea notificărilor s-a folosit un serviciu furnizat de către cei de la Google, Firebase Cloud Messaging. Pentru utilizarea acestui serviciu, fiecărui utilizator îi va fi atribuit un token FCM pentru a fi identificat în mod unic. În momentul în care utilizatorul permite browser-ului să primească astfel de notificări, un token îi va fi atribuit și va fi trimis serverului.

Dacă utilizator are asociate mai multe astfel de token-uri, se va trimite o notificare către fiecare token în parte. În momentul în care utilizatorul se deconectează de pe un dispozitiv, token-ul FCM aferent dispozitivului va fi șters.

Un utilizator va primi notificări într-unul din următoarele scenarii:

* în momentul în care s-a activat o regulă
* acțiunile ce sunt realizate în momentul în care se activează o regulă
* în momentul în care un senzor/actuator și-a modificat valoarea

În cazul în care un device își modifică valoarea, va fi notificat fiecare utilizator ce are un widget asociat dispozitivului. Notificările pot fi expediate în două moduri: către un anumit client (folosind token-ul FCM al clientului) sau către dispozitive multiple (trimițând o notificare către un anumit topic). În cadrul platformei, un topic FCM asociat unui dispozitiv, se bazează pe topic-ul MQTT al acestuia. Transformarea unui topic MQTT într-unul FCM se realizează în modul următor:

*ow59y68me/attic/temperature => topics/ow59y68me\_attic\_temperature*

Un utilizator se abonează (face *subscribe*) la notificările din partea unui device în momentul în care îi atribuie un widget. Pentru transmiterea notificărilor s-a utilizat pachetul *fcm-push* [33] care va facilita transmiterea de notificări. Pentru configurare, este necesar *SERVER\_KEY-ul* asociat unei aplicații Firebase, acesta obținându-se din consola web oferită de aceștia.

Formatul unui mesaj ce este transmis se poate vedea mai jos:

const message = {

to: fcm\_topic,

data: {

topic: fcm\_topic,

thingName: thingName,

body: `Has new value:${newValue}`,

value: newValue,

date: Date.now(),

icon: "https://cdn4.iconfinder.com/data/icons/google-i-o-2016/512/google\_firebase-2-128.png"

}

};

„*To*” reprezintă destinatarul acestei notificări, „*topic*” și „*thingName*” identifică dispozitivul ce și-a schimbat starea, „*body*” reprezintă conținutul mesajului notificării ce este afișat destinatarului/destinatarilor și „*value*” este valoare curentă a senzorului/actuatorului.

Transmiterea notificării se realizează în modul următor:

const FCM = require("fcm-push");

const fcm = new FCM(SERVER\_KEY);

pushNotification: function (message, next) {

fcm.send(message, function (err, response) {

if (err) {

logger.error("Something has gone wrong!", err);

next(err, null);

return;

}

next(null, response);

});

}

Așa cum am afirmat mai sus, comunicarea dintre dispozitive și platformă se realizează cu ajutorul unui intermediar, *Devices Gateway*. Dispozitivele transmit informațiile la topic-ul „*thingModifyValue” iar* *gateway-ul* se va abona la acesta pentru a ține platforma la curent cu toate modificările apărute. *Devices Gateway* va transmite informația mai departe către platformă prin intermediul unui cereri HTTP.

Modalitatea în care dispozitivele transmit mesaje este prezentată mai jos.

const client = awsIOT.device(credentials);

function publishMessage(topic, value, callback) {

console.log("[Publisher] Publish data:", topic, value);

const data = {

topic: topic,

value: value

};

client.publish("thingModifyValue", JSON.stringify(data));

callback(null);

}

Modul în care se realizează abonarea la topic-uri și primirea mesajelor de actualizare a stării unui dispozitiv sunt prezentate mai jos.

client.on('connect', function () {

logger.info("Client connected to broker");

User.find({}, {

'things.topic': 1,

'\_id': 0

}, function (err, res) {

for (let document of res) {

if (document.things.length > 0) {

for (let result of document.things) {

logger.info(`Subscribe to:${result.topic}`);

client.subscribe(result.topic);

}

}

}

});

});

Maniera în care sunt tratate mesajele venite din partea dispozitivelor este una asincronă.

client.on('message', function (topic, message) {

message = message.toString();

const thingTopic=message.topic;

const newValue=message.value;

updateTopicValue(thingTopic, newValue, function (err) {

if (err) {

console.log(err);

res.locals.message = "Failed to update the value from DB";

next();

return;

}

});

});

MongoDB este o bază de date ce lucrează cu colecții (asemănătoare ca și concept cu tabelele) ce rețin documente, bazate pe indecși generați automat. Fiecare document este identificat prin intermediul câmpului *\_id*. OPID are în spate trei colecții: users, ruleHistory, thingHistory. Conectarea la baza de date este o conexiune stabilă – este inițiată la momentul pornirii serviciului.

S-a optat pentru o astfel de structură –imbricată- datorită faptului că în MongoDB nu se pot realiză join-uri între colecții și deoarece s-a dorit diminuarea duratei de răspuns a unei cereri. Interfața de lucru cu baza de date se rezumă în principal la metode de căutare în baza de date, metode de actualizare a unor câmpuri din documentele interne și transformarea obiectelor *Javascript* în obiecte sau vectori în format JSON.

## 6.2 Interfața cu utilizatorul

Arhitectura *front-end*-ului este construită din entități specifice *framework*-ului AngularJS: controllers, models, directives, services, views. Sistemul de dirijare este realizat prin intermediul modulului intern *ngRoute*, permițând crearea stărilor ce înglobează un *controller*, un model și un view, astfel abordându-se paradigma arhitecturală MVC (model-view-controller) [10]. S-a optat pentru un tip de aplicație *SPA* (Single-Page-Application) [34] datorită avantajelor pe care le oferă. O aplicație de tip *SPA* încarcă toate resursele necesare navigării la încărcarea primei pagini. Pe parcursul interacțiunii clientului cu aplicația, resursele sunt încărcate în mod dinamic. De cele mai multe ori, aplicația va modifica URL pentru a simula aplicațiile web tradiționale însă nu se mai face nici o altă cerere pentru a încărca complet o pagină.

Clientul va fi informat de orice modificare survenită în sistem prin intermediul notificărilor. Pentru a beneficia de acest serviciu, clientul trebuie să permită browser-ului să primească notificări. În momentul în care s-a obținut un token FCM, acesta este trimis serverului pentru ca acesta să aibă posibilitatea de notificare a clientul.

Solicitarea permisiunii și înregistrarea token-ului FCM se realizează astfel:

const config = {

messagingSenderId: SENDER\_ID

};

firebase.initializeApp(config); // configurare Firebase

const messaging = firebase.messaging();

//Solicit clientului permisiunea de primire a notificarilor

messaging.requestPermission()

.then(function () {

console.log('Permission granted');

// token FCM atribuit utlizatorului

return messaging.getToken();

})

.then(function (token) {

// Trimit serverului token-ul FCM atribuit utilizatorului

const data = {token: token};

$http.post("https://opid-server-cypmaster14.c9users.io/api/FCMToken", data)

.then(function (response) {

$window.localStorage.setItem('fcmToken', token);

},

function (response) {

console.log(response);

});

});

În momentul în care o notificare ajunge la client, interfața se va modifica și va înștiința utilizatorul că a survenit o modificare. Clientul intră în posesia conținutului notificării astfel:

messaging.onMessage(payload => {

});

Dacă o notificare este primită în momentul în care utilizatorul interacționează cu aplicația, ea este procesată de către client, altfel este procesată de către un *service-worker*. Un *service-worker* este un script pe care browser-ul îl execută în fundal, separat de o pagină web, oferind anumite caracteristici ce nu necesită o pagină web sau interacțiunea unui utilizator. Caracteristica la care facem referire este *Push Notification* [26]. Atribuția service-worker-ului este aceea de a primi mesajul și a notifica utilizatorul că a survenit o modificare. O porțiune din script-ul executat de worker se poate vedea mai jos. [35]

importScripts('https://www.gstatic.com/firebasejs/3.5.2/firebase-app.js');

importScripts('https://www.gstatic.com/firebasejs/3.5.2/firebase-messaging.js');

//paramteri necesari configurării *Firebase-ului*

const config = {

apiKey: API\_KEY,

authDomain: AUTH\_DOMAIN,

databaseURL: DATABASE\_URL,

projectId: PROJECT\_IDD,

storageBucket: STORAGE\_BUCKER,

messagingSenderId: SENDER\_ID

};

firebase.initializeApp(config);

const messaging = firebase.messaging();

messaging.setBackgroundMessageHandler(function (payload) {

const title = payload.data.status;

const option = {

body: payload.data.body,

icon: payload.data.icon,

click: payload.data.click

};

return self.registration.showNotification(title, option);

});

Fiecărui client i se oferă un tablou de control pe care îl va putea personaliza. În cadrul acestuia, utilizatorul va avea posibilitatea de a vedea modificările survenite în sistem. Aceste modificări sunt primite numai dacă în cadrul tabloul de control senzorul/actuatorul ce și-a schimbat stare are asociat un widget. Gradul de personalizare este oferit prin intermediul modulului *angular-gridster* [36]. Acesta permite transformarea dashboard-ului într-o grilă în care widget-urile poate avea dimensiuni diferite și i se poate schimba locația. Widget-urile vor fi reprezentate sub forma unui *gauge*, ce este oferit de modulul *angular-gauge* [37]. Utilizarea celor două module se poate observa în porțiunea de mai jos :

<div gridster="gridsterOpts">

<ul>

<li class="dashboardWidget" ng-class="{'resize-trick':widget.resizable===false}" gridster-item="widget" ng-repeat="widget in widgets">

<div>

<div class="box-header widget-header" ng-show="showSaveIcon">

<div class="box-header-btns right-align">

<span class="right-align">

<i class="fa fa-trash fa-2x trashIcon" aria-hidden="true" ng-click="removeWidget(widget.topic)">

</i>

</span>

</div>

</div>

<div class="centeredWidget box-content" ng-click="!showSaveIcon&& widgetClick(widget.topic)">

<ng-gauge size="{{widget.size}}" type="full" thick="4" value="widget.value" cap="round" label="{{widget.name}}" foreground-color="#ECF0F1" background-color="#ecf0f1" ng-show="widget.outputType==='Number'">

</ng-gauge>

<ng-gauge size="{{widget.size}}" type="full" thick="7" value="widget.value" cap="round" label="{{widget.name}}" foreground-color="#e74c3c" append="%" background-color="#ecf0f1" ng-show="widget.outputType==='Percentage'">

</ng-gauge>

<ng-gauge size="{{widget.size}}" type="full" thick="4" value="widget.value" cap="round" foreground-color="#e74c3c" background-color="#ecf0f1" ng-show="widget.outputType==='String' || widget.outputType==='Boolean' ">

</ng-gauge>

</div>

<div class="center-align" ng-show="widget.outputType==='String' || widget.outputType==='Boolean'">

<h5>{{widget.name}}</h5>

</div>

</div>

</li>

</ul></div>

Restul componentelor aferente interfeței au fost create utilizând MaterializeCSS, ce se bazează pe conceptul de Material Design creat de cei de la Google.

Senzorii/actuatorii sunt simulați prin intermediul unei interfețe web. Odată modificată valoarea unui senzor din cadrul interfeței, un mesaj MQTT va fi trimis la *topic*-ul „*thingModifyValue*” la care este abonat „*Devices Gateway*”. Cel din urmă primește mesajul și îl trimite mai departe către platformă. Astfel este simulată o interacțiune, dintre un dispozitiv și platformă, cât mai aproape de realitate.

Pentru adăugarea unui dispozitiv, trebuiesc adăugate topic-ul MQTT aferent și numele device-ului. Prima parte a fiecărui topic reprezintă un șir *random* pentru a identifica unic fiecare dispozitiv. Un astfel de topic se poate observa în exemplul de mai jos:

în exemplul de mai jos:

*ow59y68me/attic/temperature*

# 7. Manual de utilizare

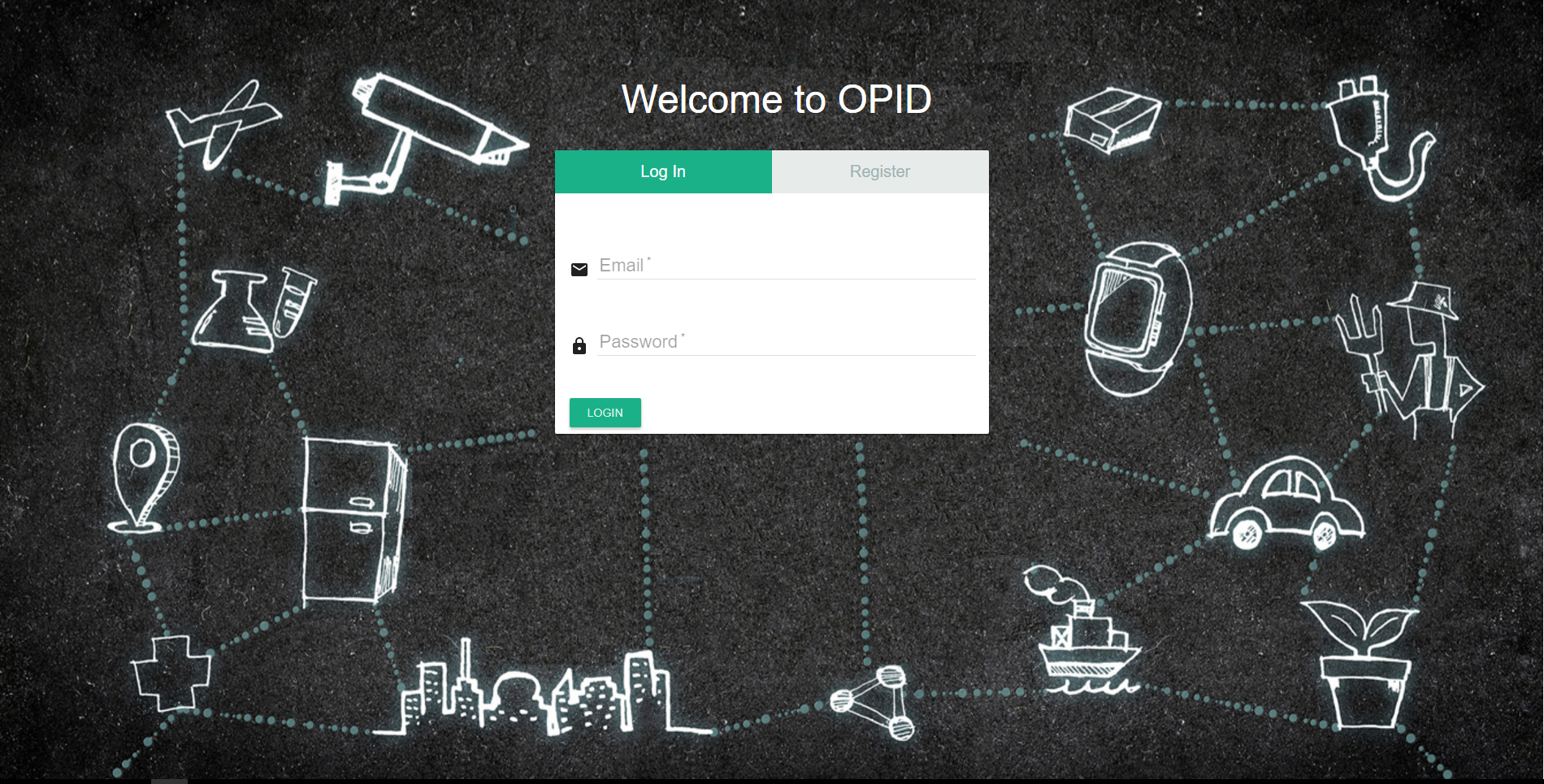
 La prima interacțiune cu aplicația, fiecare utilizator este întâmpinat pagina de start ce permite utilizatorului fie să se autentifice, fie să se înregistreze.

Figura 20. Pagina de start

Dacă utilizatorul are creat deja un cont, acesta va trebui să introducă email-ul și parola pentru accesarea platformei. În caz contrar, acesta va trebui să se înregistreze, furnizând unele informații cum ar fi: prenumele, numele, email-ul și parola (Figura 21). După ce s-a înregistrat, utilizatorul va fi redirecționat către formularul de autentificare.

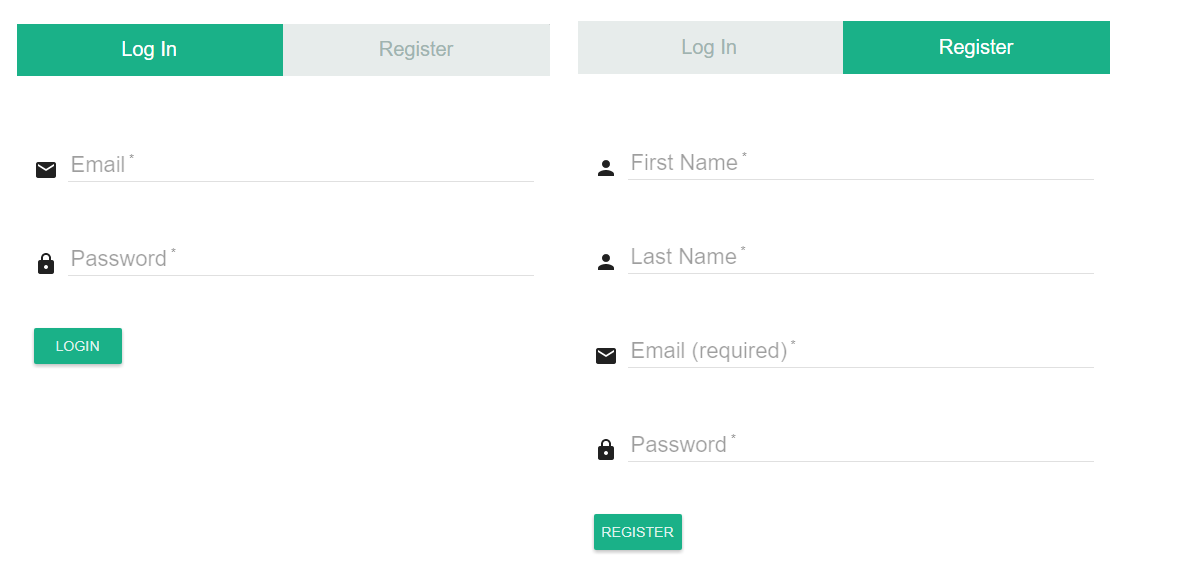


Figura 21. Autentificarea şi înregistrarea

Odată autentificat, utilizatorul este redirecționat către pagina principală a aplicației ce este formată din trei mari componente: *dashboard*, *things*, *rules*. In cadrul *dashboard*-ului, utilizatorul va avea la dispoziție următoarele facilități: atribuirea de widget-uri dispozitivelor înregistrate (Figura 22) și interacțiunea cu actuatorii prin intermediul widget-urilor. Utilizatorii vor avea posibilitatea de personalizare a *dashboard*-ul, vor putea poziționa widget-urile oriunde doresc iar pentru unele dintre ele le vor putea modifică dimensiunea. Pentru crearea de widget-uri, utilizatorul trebuie doar să introducă numele dispozitivului. Utilizatorul va beneficia de o interfață ce va fi actualizată în momentul în care un senzor/actuator își modifică valoarea și de asemenea, va fi notificat când se produce o astfel de schimbare în sistem.

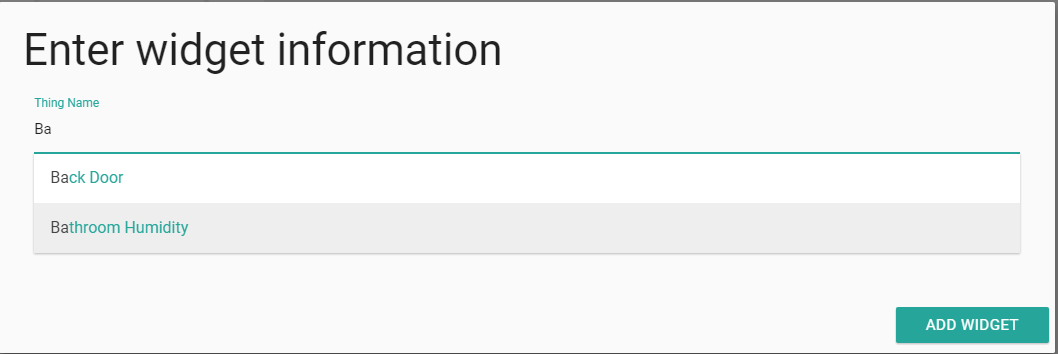


Figura 22. Adăugarea unui widget

Singura modalitate de a modifica starea unui actuator este prin intermediul widget-ului asociat. Odată acționat widget-ul, utilizatorul i se va deschide o fereastră de unde va putea alege o nouă valoare validă pentru dispozitiv (Figura 23). Persoanele care au asociate un widget pentru același dispozitiv vor fi notificate de schimbările realizate de către ceilalți utilizatori.

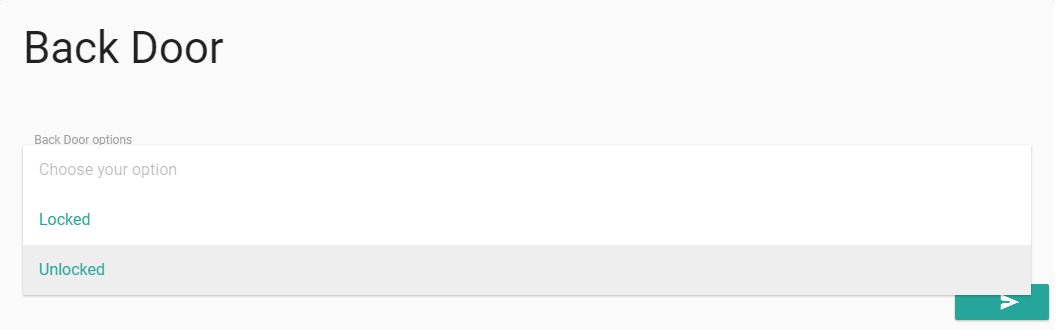


Figura 23. Modificarea stării unui dispozitiv

Pagina dedicată dispozitivelor poate fi accesată selectând opțiunea *Things*, din cadrul meniului lateral. În cadrul acestei pagini, utilizatorul poate să realizeze managementul senzorilor/actuatorilor înregistrați în sistem, poate adăuga și elimina dispozitive (Figura 24). De asemenea, informațiile despre un dispozitiv pot fi accesate ușor, prin intermediul unui simplu *click* pe figura ce conține numele acestuia.

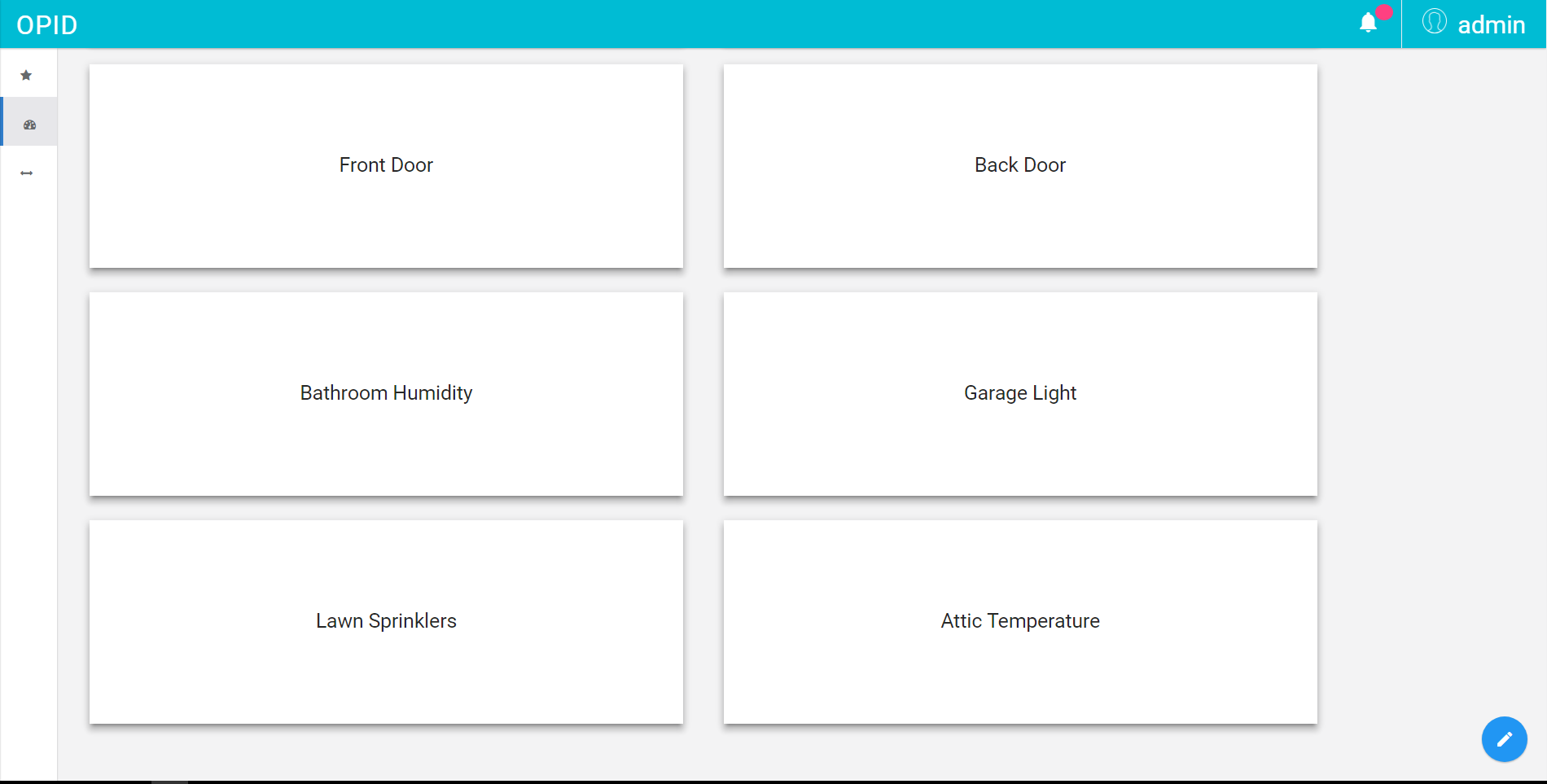


Figura 24. Dispozitivele utililizatorului

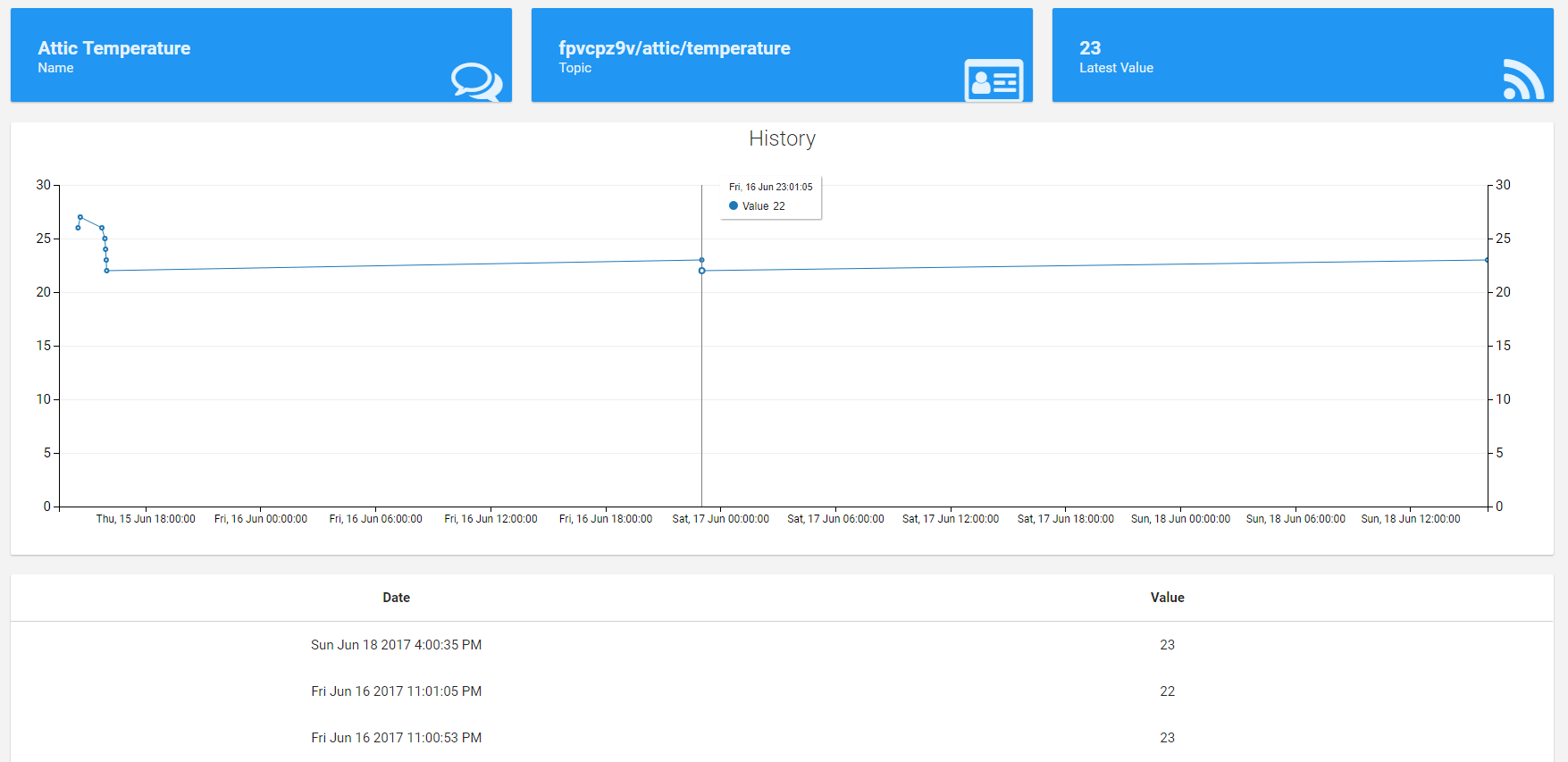
 Accesând pagina unui dispozitiv, utilizatorului i se vor prezenta o serie de informații cum ar fi: topic-ul MQTT aferent dispozitivului, numele acestuia și ultima valoarea înregistrată. De asemenea, utilizatorului i se va pune la dispoziție un grafic prin intermediul căruia va putea observa fluctuațiile valorilor înregistrate. Un tabel este pus la dispoziție utilizatorului unde poate observa întregul istoric al valorilor înregistrate de senzori/actuatori (Figura 25).

Figura 25. Informaţii despre dispozitiv

Pagina *Rules* oferă utilizatorului posibilitatea de a adăuga reguli în sistem și de a le vedea pe acelea deja introduse. În momentul în care o regulă a fost adăugată, aceasta este în starea inactivă. Pentru ca o regulă să fie evaluată în momentul în care un dispozitiv își schimbă valoarea, aceasta trebuie să fie în starea activă. Utilizatorul poate modificare starea unei reguli prin intermediul butonului aflată la baza figurii ce conține numele acesteia (Figura26).

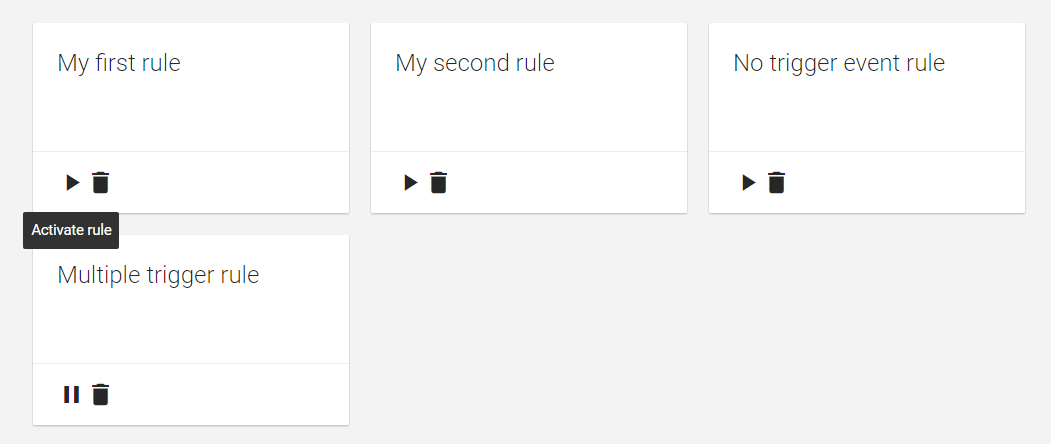


Figura 26. Regulile utilizatorului

Adăugarea unei reguli se face prin intermediul opțiunii aferente din cadrul butonului din partea de jos a paginii. Odată accesată această opțiune, utilizatorul va trebui să completeze un formular care va reprezenta configurația regulii. Clientul va avea posibilitatea de a-și seta: un interval în care regula să fie activă, o dată la care regula să fie evaluată și opțiunea de a fi notificat de fiecare dată când regula este evaluată cu succes. Se vor putea seta o serie de condiții ce vor fi verificate dacă sunt îndeplinite în momentul în care o regulă este evaluată. De asemenea, utilizatorul poate stabili anumite acțiuni ce vor fi realizate în momentul în care regula va fi evaluată cu succes.

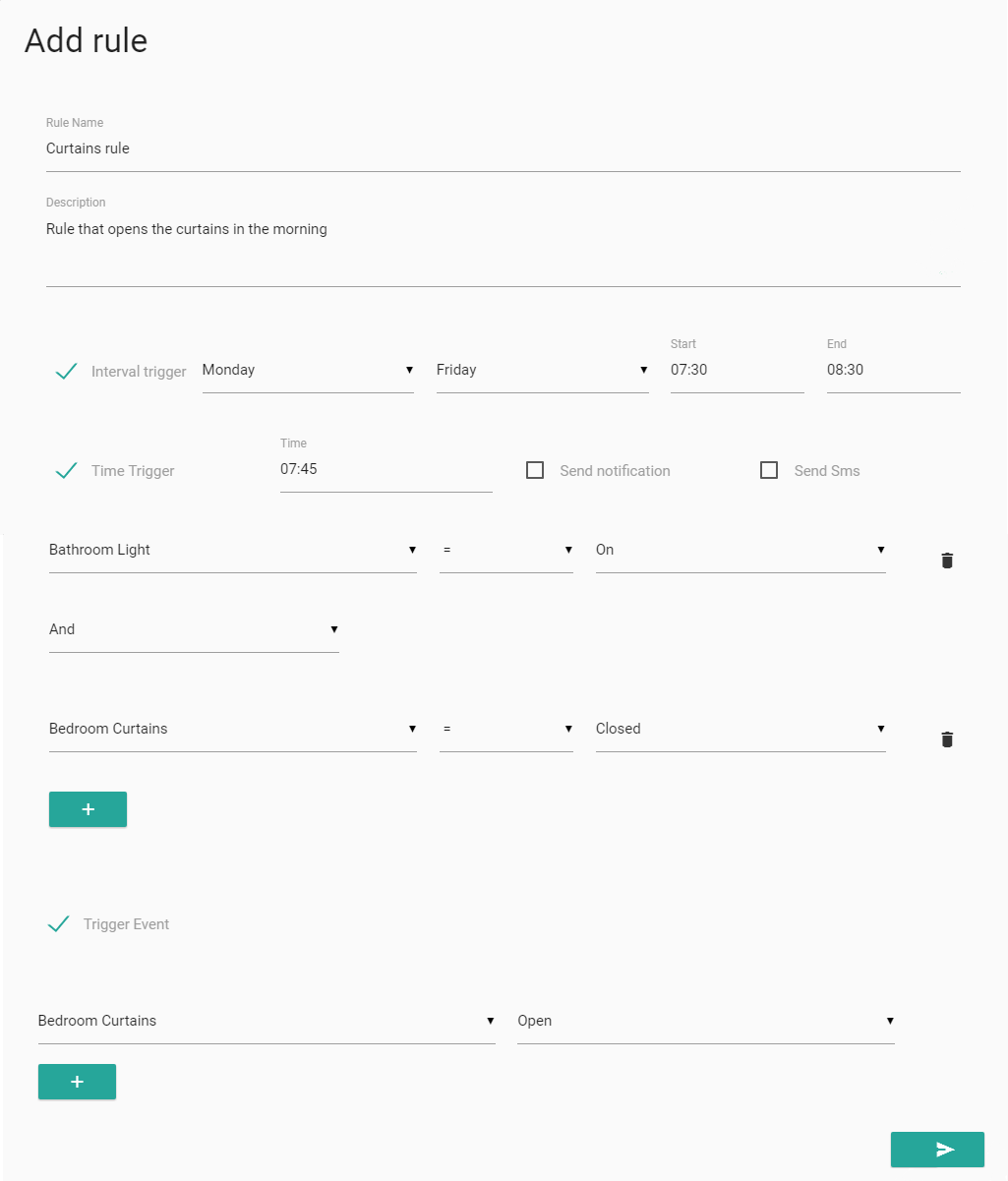


Figura 27.Adăugarea unei reguli

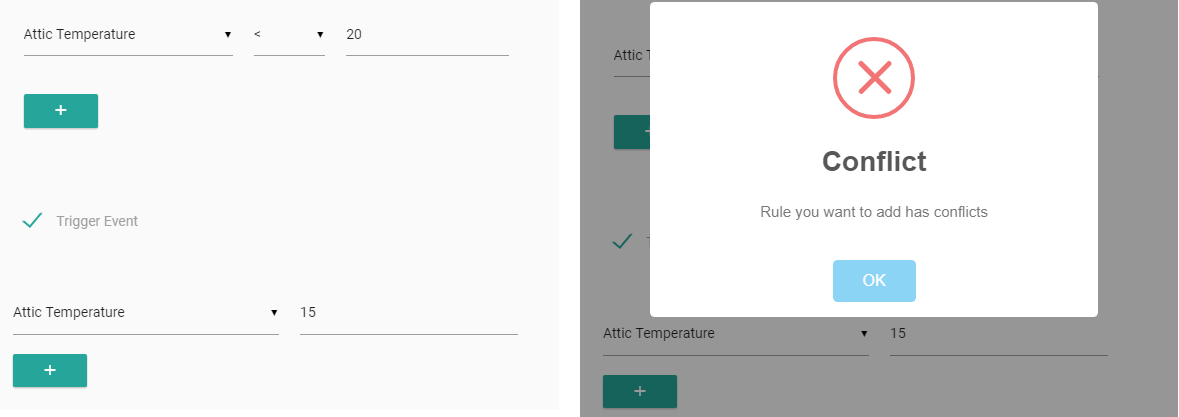
Dacă regula ce se dorește a se introduce conține conflicte, utilizatorul va fi notificat și va trebui să modifice conținutul regulii dacă va dori în continuare să o adauge în sistem (Figura 28).

Figura 28. Reguli ce conțin conflicte

Prin intermediul unui simplu click pe o regulă, utilizatorul va fi redirectat către pagina acesteia care este formată din două tab-uri: *details* si *events*. În cadrul primul tab, clientul va putea modifica conținutul regulii ( e.g intervalul în care o regulă este activă, data la care regulă să fie evaluată, etc.). În cazul în care nouă configurație a regulii va conține conflicte, utilizatorul va fi notificat că nu se poate realiza actualizarea. În cadrul celui de-al doilea tab este afișat istoricul regulii, data la care a fost activată și respectiv ce a cauzat activarea ei.

Informațiile puse la dispoziție de *widget*-urile ce sunt prezente în cadrul tabloului de control, se vor actualiza în mod automat în momentul în care se primește o notificare din partea serverului. Aceste notificări sunt primite în momentul în care un dispozitiv și-a modificat valoarea sau atunci când o astfel de modificare activează o regulă. Deoarece există posibilitatea ca utilizatorul să nu observe actualizarea valorii widget-ului, va fi afișat un mesaj cu conținutul notificării timp de trei secunde. Acțiunile realizate în momentul în care se primește o astfel de notificare pot fi observate în figura de mai jos. (Figură 29)

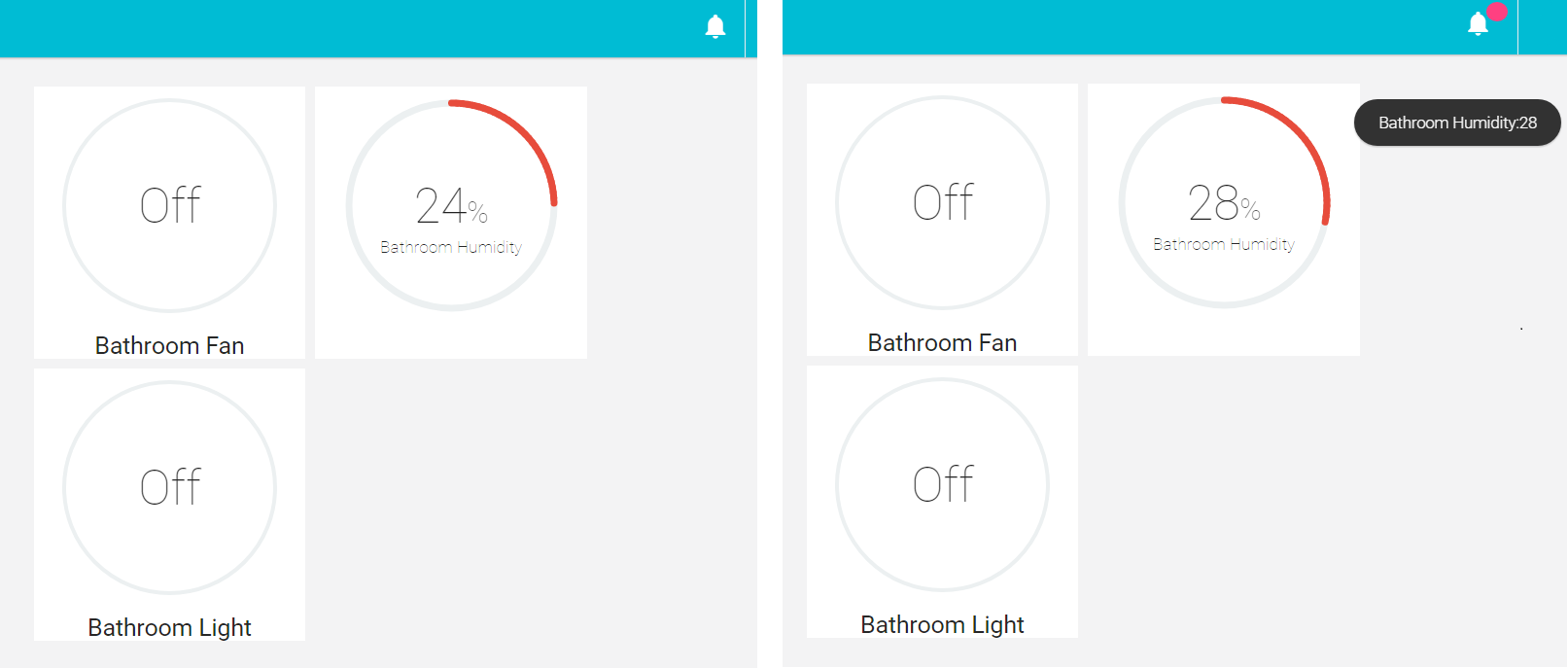


Figura 29. Actualizarea valorii unui dispozitiv

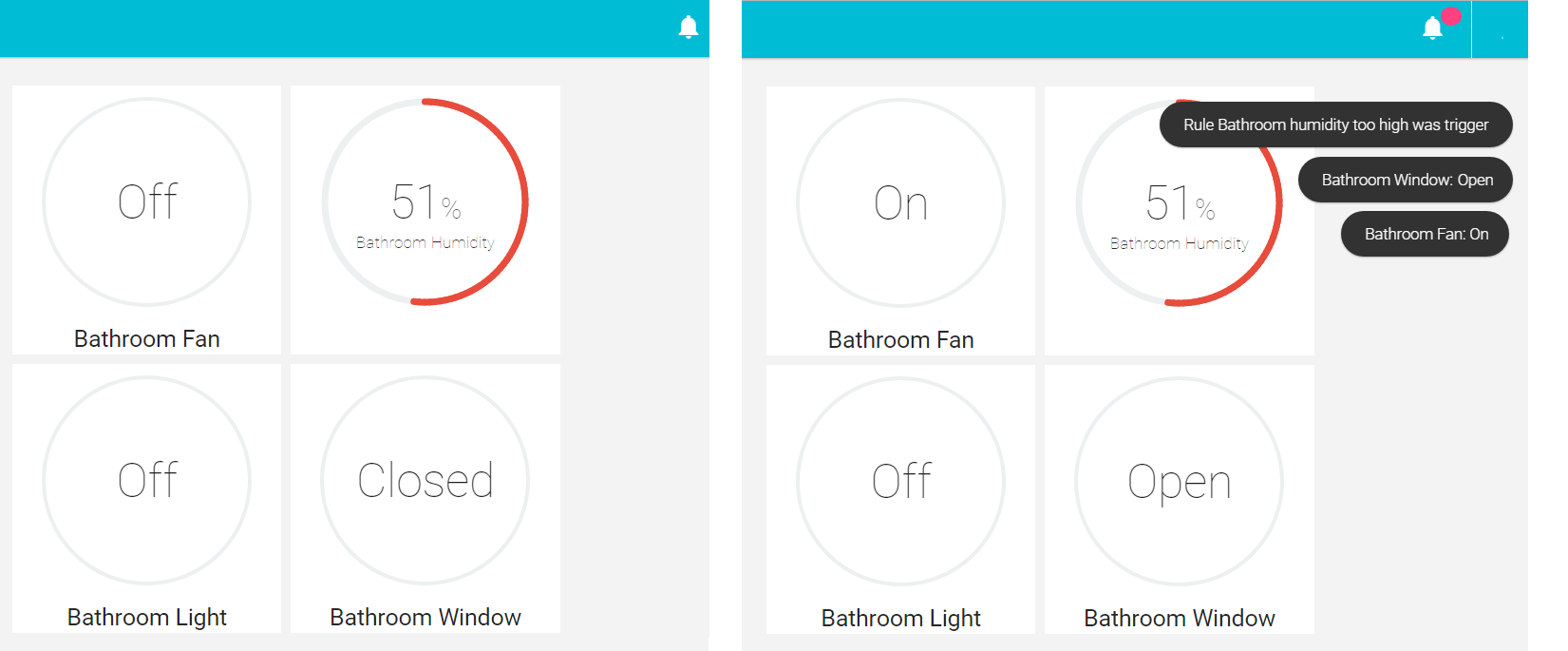
 Notificările referitoare la actualizarea valorilor unor dispozitive nu sunt singurele primite de utilizator. Acesta va primi notificări și în momentul în care o regulă este evaluată cu succes. De asemenea, se vor primi notificări și în legătură cu acțiunile ulterioare activării unei reguli (Figura 30). Să presupunem că avem o regulă de genul următor: „Atunci când umiditatea din baie depășește 50%, deschide geamul și pornește ventilatorul din baie”. Acțiunile ulterioare activării acestei reguli se pot observa în imaginea de mai jos.

Figura 30. Activare unei reguli și acțiunile ulterioare

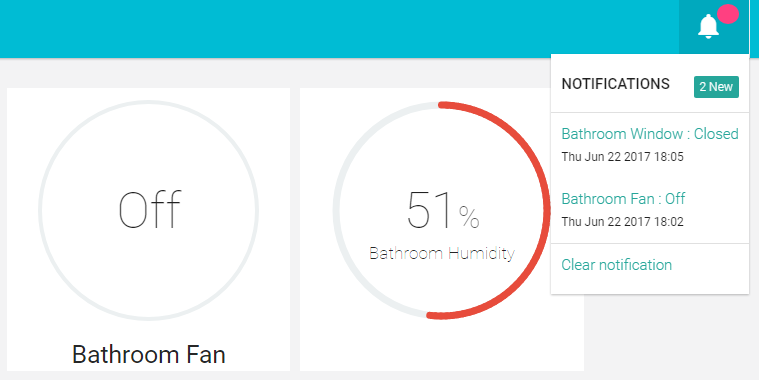
 Notificările primite de utilizatori vor putea fi observate și în meniul special destinat acestora (Figura 31), unde vor fi sortate descrescător în funcție de data primirii.

Figura 31. Bara de notificări

Dacă în momentul în care un dispozitiv își modifică valoarea, utilizatorul nu interacționează cu aplicația, aceasta va fi alertat prin intermediul unei notificări (Figura 32).



Figura 32. Notificarea utilizatorului

# 8. Studiu de caz

În continuare, se va prezenta un caz ipotetic de utilizare al acestei platforme.

Directorul unui liceu a achiziționat o serie de de senzori și actuatori (e.g. senzori de temperatură, becuri inteligente, mecanisme de închidere și deschidere a jaluzelelor în mod automat, încuietori inteligențe, mecanisme de deschidere a geamurilor în mod automat și multe altele). pentru a le pune în cadrul sălilor de curs. Acesta citește un articol despre OPID și despre soluțiile pe care le furnizează și se hotăraște să folosească această platformă în cadrul instituției pe care o reprezintă. După ce au fost instalate, dispozitivele trebuie adăugate în cadrul platformei. Se creează câte un cont pentru fiecare sală de curs (e.g. B311) și se adaugă fiecare dispozitiv utilizându-ne de topic-ul MQTT al acestora (Figura 33.1).

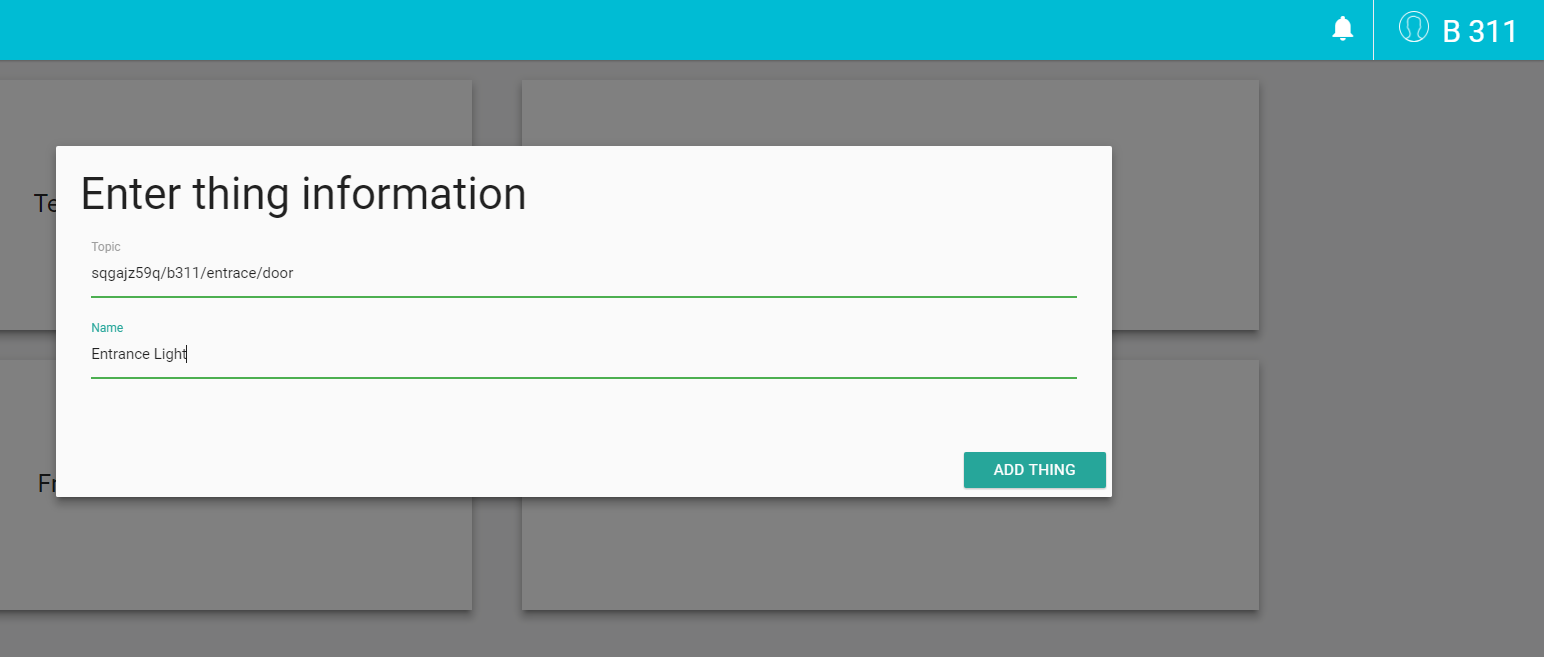


Figura 33.1 Adaugarea unui dispozitiv

Odată adăugate dispozitivele în cadrul platformei, pentru fiecare dintre ele se creează un widget deoarece cu scopul de a se primi notificări în timp real în legătură cu modificări ce se petrec în sistem. (Figura 33.2)

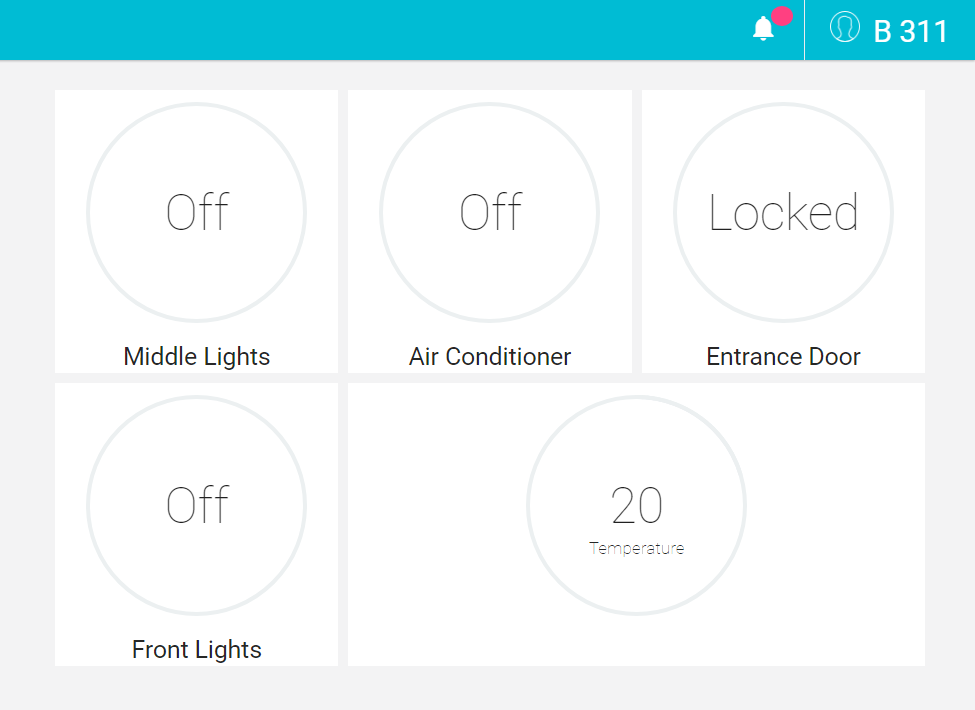


Figura 33.2 Panoul de control aferent sălii B 311

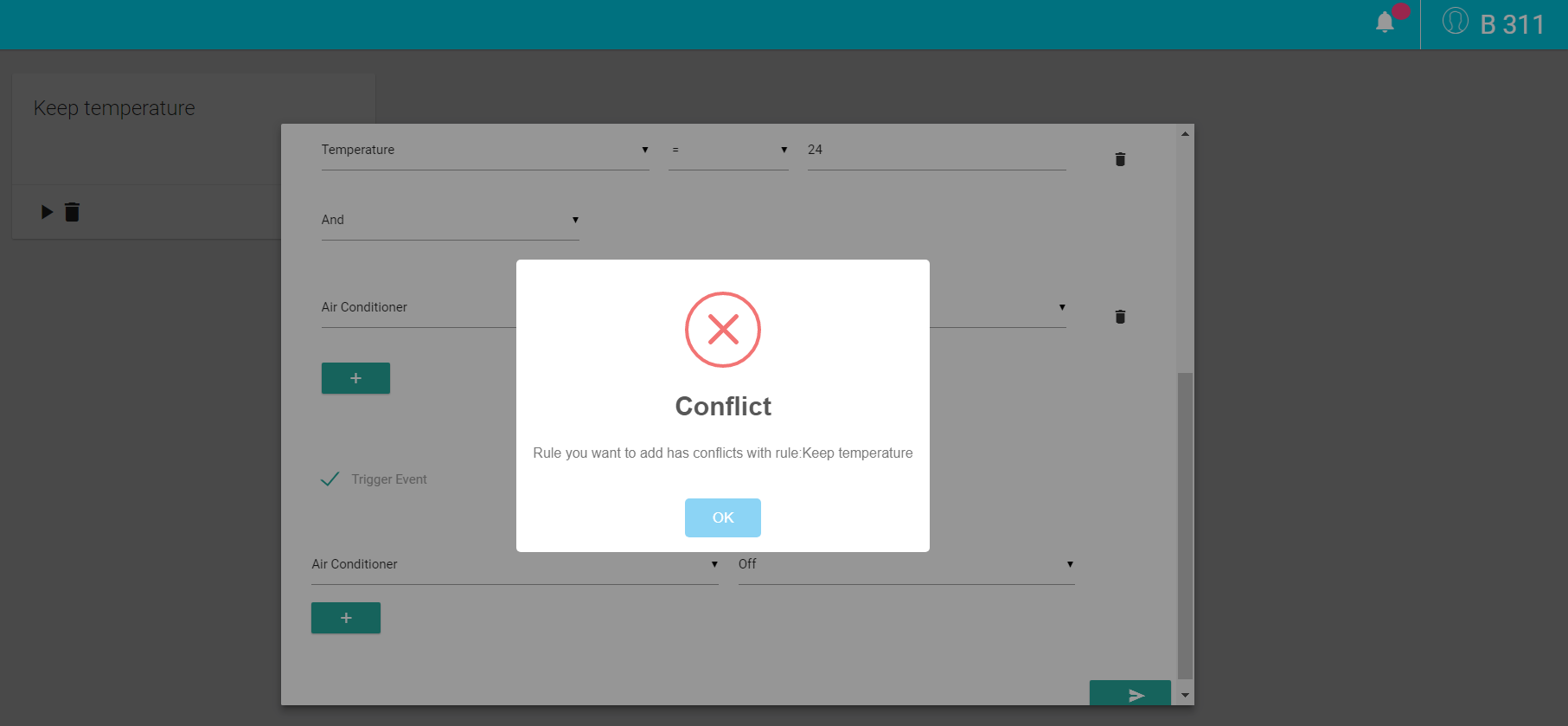
După adăugarea widget-urilor, pentru fiecare clasă se decide ce reguli ar trebui să fie introduse în sistem. Elevii stabilesc regulile și le trimit dirigintelui pentru a fi introduse în cadrul platformei. Introducând regulile, un diriginte este notificat dacă regula pe care dorește să o introducă în sistem, în acel moment, se află în conflict cu o altă regulă. (Figura 33.3) Regula: „Dacă temperatură din sală este mai mare de 22 de grade și aerul condiționat este închis, atunci pornește aerul condiționat” se află într-un conflict cu o altă regulă introdusă anterior care specifică următorul lucru: „Dacă temperatură în sala de curs este 24 grade iar aerul condiționat este pornit, atunci închide aerul condiționat.”

Figura 33.3 Adăugarea unei reguli ce conține conflicte

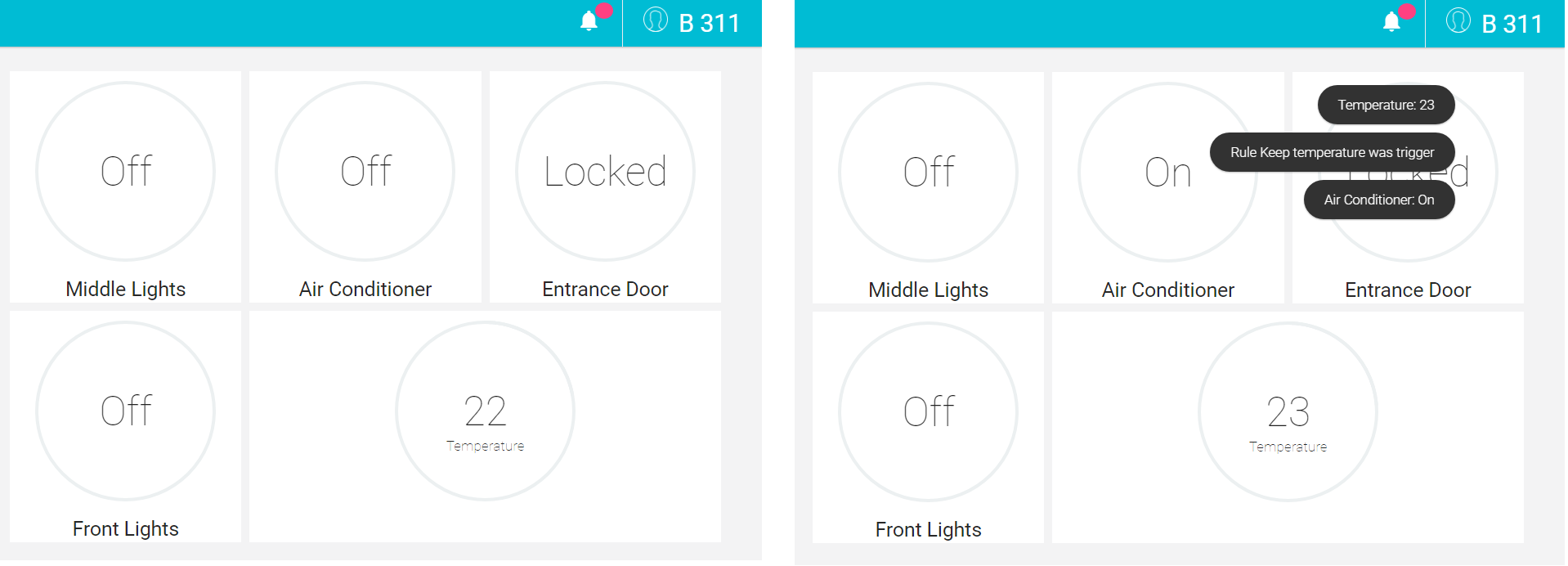
 În zilele toride de la sfârșitul lunii iunie, elevi pot observa că în sălile de curs ambientul este unul plăcut, temperaturile fiind ținute la un nivel satisfăcător. (Figura 33.4)

Figura 33.4 Creșterea temperaturii și activarea regulii

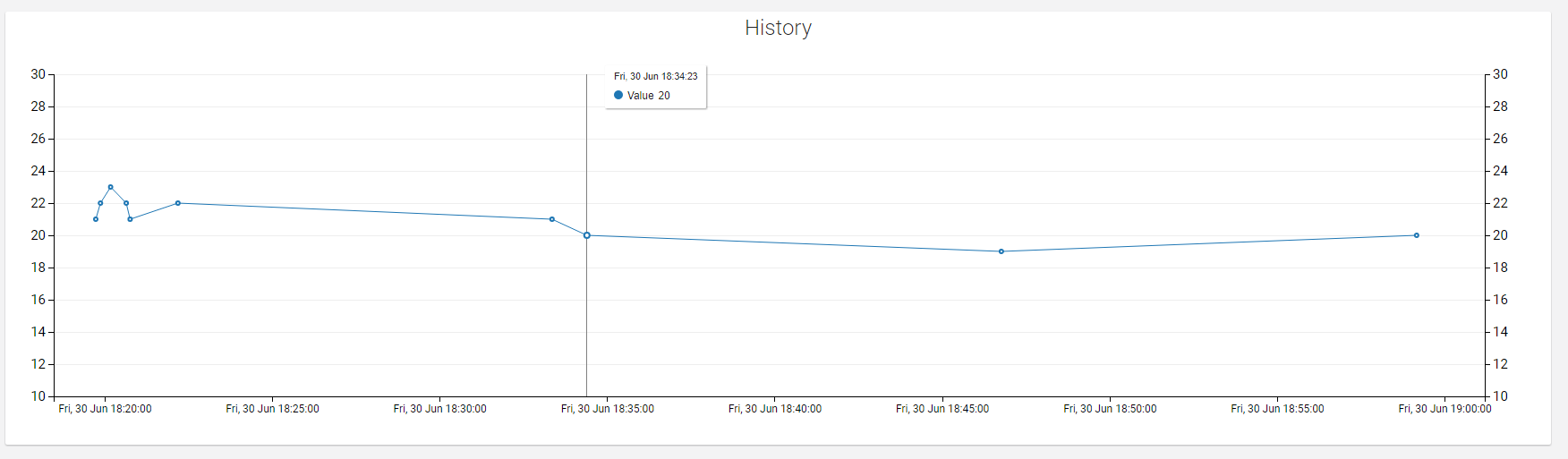
 Elevii și dirigintele clasei ce își desfășoară orele în sala B 311 pot vedea fluctuația valorilor înregistrate de dispozitive, respectiv temperaturile înregistrate și momentul din zi în care se înregistrează temperaturi ridicate (Figura 33.5).

Figura 33.5 Fluctuația temperaturii în sala de curs

Un paznic ce patrulează în tura de noapte, primește o notificare în legătură cu faptul că ușă de la sala B102 a fost deschisă (Figura 33.6). Observând asta, el se duce repede să verifice și observă că doi indivizi au forțat ușa și au intrat în cadrul laboratorului de informatică cu scopul de a sustrage echipamente IT. Paznicul imobilizează cei doi răufăcători și sună la poliție.

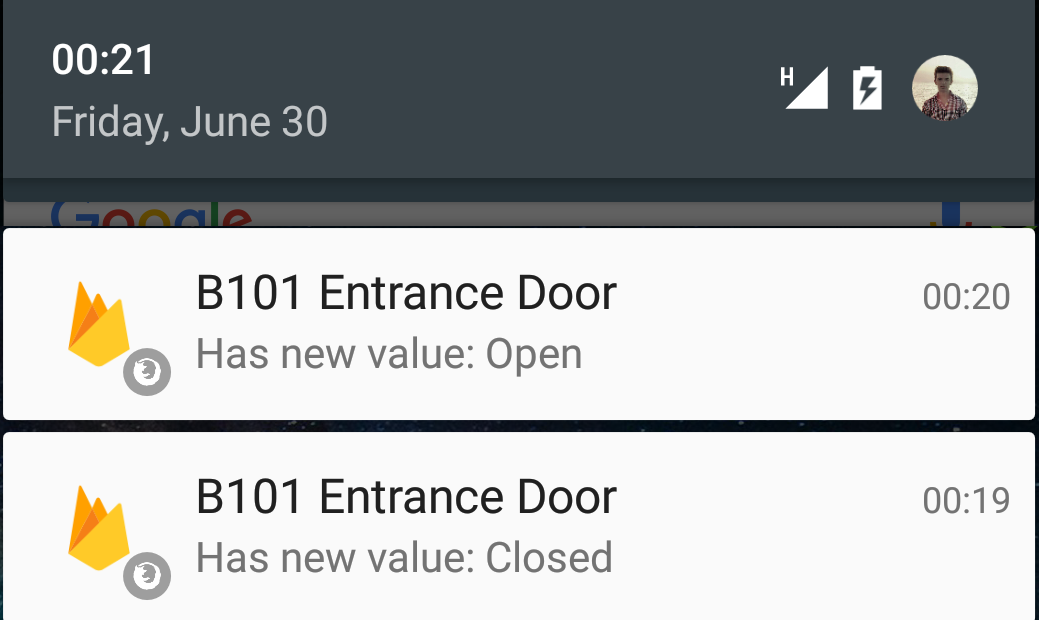


Figura 33.6 Notificările în legătură cu sala B101

Directorul liceului observând cât de utilă este o astfel de platformă, o recomandă și colegilor săi care doresc să introducă dispozitive inteligente în cadru sălilor de curs.

# 9. Concluzii

„*Online Platform for IoT Devices”* a fost dezvoltată cu scopul de a facilita interacțiunea dintre utilizator și dispozitivele IoT și de a rezolva problema interoperabilității între device-urile inteligente. Pentru comunicarea între dispozitive și platformă s-a folosit AWS IoT ce pune la dispoziție un mecanism sigur de transmitere a mesajelor. Transferul acestor tipuri de mesaje se realizează folosind protocolul MQTT special dezvoltat pentru comunicarea M2M (Machine-to-Machine). O evaluare a performanței platformei se găsește la Anexa 1.

Aplicația finală este o platformă web ce oferă utilizatorului posibilitatea de interacțiune cu actuatorii în orice moment și de oriunde s-ar afla. Datorită manierei și tehnologiilor folosite în dezvoltarea serverului *, API RESTful* și *Push Notifications*, se pot dezvolta cu ușurință aplicații native pentru orice platformă. Una dintre direcțiile viitoare de dezvoltare ar fi dezvoltarea de aplicații Android și iOS, fără ca serverul să necesite anumite modificări. Un avantaj foarte important îl reprezintă abilitatea de a folosi mecanismul de PushNotification și în cadrul aplicațiilor native.

Pentru automatizarea acțiunilor utilizatorilor s-a utilizat un sistem bazat pe reguli. Procesul de evaluare a unei reguli constă în parcurgerea arborelui asociat formulei logice. Deci, complexitatea algoritmului de satisfacere a regulilor este O(x+y), unde x este numărul de operanzi și y numărul de operatori. Aceștia vor putea stabili anumite condiții care trebuiesc îndeplinite pentru activarea unei reguli. Condițiile vor fi transformate într-o expresie logică, iar cu ajutorul acestora se va construi arborele asociat formulei logice care va fi utilizat în evaluarea expresiei. Utilizatorul va avea posibilitatea de a stabili anumite acțiuni ce vor fi executate în momentul în care rezultatul evaluării regulii este unul pozitiv. De asemenea, se poate seta o dată exactă la care să se evalueze regula și se poate specifica un interval temporal în care regula să fie activă. O direcție de dezvoltare în această privință poate fi utilizarea unor algoritmi de învățare automată sau a unei rețele neuronale pentru a identifica anumite acțiuni pe care un utilizator le realizează în mod frecvent. După ce s-a identificat o astfel de acțiune, ea poate fi automatizată prin intermediul introducerii unei noi reguli în sistem. De asemenea, o altă îmbunătățire ar fi adăugarea unei mecanism de evitare a apariției unui ciclu în momentul în care se adaugă o nouă regulă în sistem.

Pentru ca utilizatorii să fie ținuți la curent cu ultimele modificări survenite în sistem, s-a utilizat serviciul de *Push Notification* furnizat de către cei de la Firebase. Optând pentru acestă tehnologie, introdusă odată cu HTML5, utilizatorii vor fi notificați în momentul în care un senzor/actuator își modifică valoarea sau atunci când se activează o regulă. Utilizatorii vor primi astfel de notificări chiar dacă nu interacționează cu platforma sau în cazul în care nu există nici o instantădeschisă a aplicației.

Alte direcții viitoare de dezvoltare, ar fi creșterea numărului de tipuri de dispozitive acceptate de platformă. În acest sens, aplicația a fost dezvoltată într-o manieră modulară astfel încât inserarea unor noi tipuri de dispozitive să se realizeze ușor.

# 10. Bibliografie

[1] John Romkey, *„Toast of the IoT: The 1990 Interop Internet Toaster”,* IEEE Consumer Electronics Magazine*,* Ianuarie 2017

[2] O. Vermesan, P. Friess, P. Guillemin, H. Sundmaeker, et al., „*Internet of Things Strategic Research Agenda*”, Chapter 2 in Internet of Things – Global Technological and Societal Trends, River Publisher, 2011, ISBN 978-87-92329-67-7

[3] A.Sankara Narayanan, M.Syed Khaja Mohideen, M.Chithik Raja*, „IPv6 Tunneling Over IPV4*”, IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 2, No 2, March 2012

[4] Ovidiu Vermesan, Peter Friess, ,,*Internet of Things Applications: From Research and Innovation to Market Deployment*”, River Publishers, 2014

[5] Rafiullah Khan, Sarmad Ullah Khan, Rifaqat Zaheer, and Shahid Khan, *,,Future Internet: The Internet of Things Architecture,Possible Applications and Key Challenges”*, in the proceedings of 10th International Conference on Frontiers of Information Technology, Islamabad, Pakistan, 17-19 Decembrie, 2012.

[6] Miao Wu et. al., *„Research on the architecture of Internet of things”,* in the proceedings of 3rd International Conference on Advanced Computer Theory and Engineering, 20-22 August, 2012,Beijing, China,

[7] Mohammad Aazam, Imran Khan, Eui-nam Huh, *„Cloud of Things: Intergrating Internet of Things and cloud computing and the issues involved”*, 2014 11th Intenational Bhurban Conference on Applied Sciences & Technology (IBCAST) Islamabad, Pakistan

[8] \* \* \*, *Meet the Nest Learning Thermostat*: https://nest.com/thermostat/meet-nest-thermostat/

[9] \* \* \*, *Samsung smart TVs will all be IoT:* http://www.zdnet.com/article/samsung-smart-tvs-will-all-be-iot-compatible-from-2016/

[10] Sabin Buraga , „*Cursurile de Tehnologii Web și Dezvoltarea aplicaţiilor Web*

*la nivel de client*”, 2015-2016

[11] M. Jones, J. Bradley, N. Sakimura, „*The OAuth 2.0 Authorization Framework*”,  Internet Engineering Task Force (IETF), Mai 2015

[12] H. Tshofenig, J. Arkko, D. Thaler, D. McPherson*, „Arhitectural Considerations in Smart Object Networking”*(RFC 7452), Martie 2015

[13] Ian Foster, Yong Zhao, Ioan Raicu, Shiyong Lu , *„Cloud Computing and Grid Computing 360-Degree Compared”,* Grid Computing Environments Workshop, November 2008

[14] *Ahmed Mohamed Gamaleldin*, „*An Introduction to Cloud Computing Concepts” ,* Software Engineering Competence Center, 2013

[15] Lenuta Alboaie , „*Cursul de Cloud Computing*”, 2017

[16] P. Th. Eugster, P.A. Felber, R. Guerraoui, A.M. Kermarrec, *„The Many Faces of Publish/Subscribe”,* Journal ACM Computing Surveys, Volume 35 Issue 2, June 2003   
Pages 114-131

[17] \* \* \*, *MQTT Version 3.1.1:* http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html

[18] \* \* \*, *MQTT Essentials: Part 1 – Introducing MQTT:* http://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt

[19] \* \* \*, *MQTT Essentials Part 9: Last Will and Testament*: http://www.hivemq.com/blog/mqtt -essentials-part-9-last-will-and-testament

[20] \* \* \*, *Mqtt*: https://www.mqtt.com/features

[21] \* \* \*, *MQTT Essentials Part 3: Client, Broker and Connection Establishment*: http://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment

[22] \* \* \*, *MQTT Essentials Part 5: MQTT Topics & Best Practices:* http://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices

[23] \* \* \*, *AngularJS: Developer Guide: Introduction:* https://docs.angularjs.org/guide/introduction

[24] \* \* \*, *About FCM Messages:* https://firebase.google.com/docs/cloud-messaging/concept-options

[25] \* \* \*, *Push Notifications Explained:* https://www.urbanairship.com/push-notifications-explained

[26] \* \* \*, *Web Push Notifications:* https://developers.google.com/web/fundamentals/engage-and-retain/push-notifications/

[27] \* \* \*, *What is AWS IoT?* http://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html

[28] \* \* \*, *Mongoose Schema*: http://mongoosejs.com/docs/guide.html

[29] \* \* \*, *Cloud 9 - About Cloud9:* https://c9.io/site/about

[30] \* \* \*, *RFC 7519 - JSON Web Token (JWT):* https://tools.ietf.org/html/rfc7519

[31] \* \* \*, *json-rules-engine:* https://www.npmjs.com/package/json-rules-engine

[32] \* \* \*, *node-schedule:* https://www.npmjs.com/package/node-schedule

[33] \* \* \*, *fcm-push:* https://www.npmjs.com/package/fcm-push

[34] \* \* \*, *Why a single page Application, What are the benefits? What is a SPA? :* http://blog.angular-university.io/why-a-single-page-application-what-are-the-benefits-what-is-a-spa/

[35] \* \* \*, Service Workers: an Introduction: https://developers.google.com/web/ fundamentals/getting-started/primers/service-workers

[36] \* \* \*, *angular-gridster*: https://www.npmjs.com/package/angular-gridster

[37] \* \* \*, *angularjs-gauge:* https://www.npmjs.com/package/angularjs-gauge

# Anexa 1

Evaluarea performanței aplicației s-a realizat simulând o interacțiune între dispozitive și platformă. Timp de un minut s-au realizat câte 20 de cereri pe secundă. Din totalul de 1200 de cereri toate au fost realizate cu succes. Timpul mediul al unei cereri a fost de 1.3 secunde. Pe tot parcursul acestei interacțiuni, gradul de utilizare a procesorului a fost, în mare parte din timp, la 100%. Mai multe informații pot fi observate în rezultatul evaluării de mai jos.

