### A picture containing logo Description automatically generatedA picture containing logo Description automatically generatedUNIVERSITATEA DIN BUCUREȘTI

**FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ**

**BAZE DE DATE ȘI TEHNOLOGII SOFTWARE**

**Lucrare de Disertație**

**SOLUȚIE CLOUD PENTRU MONITORIZAREA CASEI INTELIGENTE**

### Absolvent

### CHIRIGIU DELIA MARIA

**Coordonator științific**

**Lect. dr. Letiția Marin**

**București, iunie 2022**

#### **Rezumat**

Scopul acestei lucrări este de a proiecta o soluție modulară, extensibilă și ușor de folosit pentru o casă inteligentă. În dezvoltarea acesteia, sunt îmbinate atât concepte Internet of Things prin arhitectura distribuită în noduri și gateway, cât și de inteligența artificială pentru a procesa datele captate și de a produce diferite perspective pe care o persoană le-ar fi omis. De asemenea, soluția dezvoltată este cloud native, susținând decuplarea diverselor componente și oferind un sistem definit de reziliență. Datele sunt transmise de către noduri folosind protocolul MQTT către un broker găzduit pe gateway, păstrând transmisia datelor în Intranet.

Cuprins

[1. Introducere 4](#_Toc105454818)

[2. Preliminarii 6](#_Toc105454819)

[2.1. Internet of Things 6](#_Toc105454820)

[2.2. Publish/Subscribe 7](#_Toc105454821)

[2.3. MQTT 8](#_Toc105454822)

[2.3.1 Protocolul MQTT 8](#_Toc105454823)

[2.3.2 Protocolul WebSocket 8](#_Toc105454824)

[2.3.2 MQTT Broker 9](#_Toc105454825)

[2.3.2 MQTT Client 9](#_Toc105454826)

[2.4. Spring Boot, Spring Security 9](#_Toc105454827)

[2.5. Docker, Containerizare si izolare 9](#_Toc105454828)

[2.6. AI/ML ? 9](#_Toc105454829)

[3. Descrierea soluției 10](#_Toc105454830)

[3.1. Arhitectura soluției 10](#_Toc105454831)

[3.2. Lista de componente 13](#_Toc105454832)

[3.3. Descrierea nodului IoT 15](#_Toc105454833)

[3.5. Descrierea Gateway-ului 16](#_Toc105454834)

[3.6. Microservicii dezvoltate 17](#_Toc105454835)

[3. Concluzii 34](#_Toc105454836)

[Bibliografie 35](#_Toc105454837)

# Introducere

Conectivitatea a devenit omniprezentă în viața modernă, transformând monitorizarea și controlul de la distanță un lucru facil, îmbunătățind calitatea vieții prin creșterea calității aerului sau prin automatizarea diferitelor treburi casnice.

Potrivit Cisco [1], numărul de conexiuni machine-to-machine va crește până la 50% în 2023 față de anul 2018 când aceasta înregistra un procent de 34%, ceea ce înseamnă o schimbare semnificativă a cotei de piața datorită popularizării Internet of Things. Aceste obiecte pot lua forma unui senzor de temperatură și umiditate conectat la Internet, o camera de monitorizare a copilului, până la electrocasnice inteligente precum mașina de spălat, aerul condiționat sau robotul aspirator.

Scopul acestei lucrări este de a proiecta o soluție modulară, extensibilă și ușor de folosit pentru o casă inteligentă. În dezvoltarea acesteia, s-au îmbinat atât concepte Internet of Things prin arhitectura distribuită în noduri și gateway, cât și de inteligență artificială pentru a procesa datele captate și de a produce diferite perspective pe care o persoana le-ar fi omis. De asemenea, soluția dezvoltata este cloud native, susținând decuplarea diverselor componente și oferind un sistem definit de reziliență. Datele sunt transmise de către noduri folosind protocolul MQTT către un broker găzduit pe gateway, păstrând transmisia datelor în Intranet. La nivel de gateway, rulează și o aplicație dezvoltată în NodeJS ce este abonată la topicul ce conține date de la senzori și le va scoate în Internet prin intermediul HTTPS, asigurând un transport securizat al încărcăturii. Destinația acesteia este un microserviciu web-facing găzduit într-o instanță cloud, care constituie rolul de dispecer spre diferitele microservicii internet responsabile. Printre ele se află și microserviciul ce oferă persistența datelor prin operațiuni de tip CRUD, acestea fiind salvate într-o bază de date Oracle Express Edition 21c. Alte microservicii notabile sunt cele de Data Warehouse și cel de inteligență artificială, decuplând astfel stocarea în rapoarte de procesarea datelor folosind algoritmi de învățare automată.

Structura lucrării este păstrata simplă pentru brevitate, conținând 4 capitole:

Capitolul 2, concepte de natură tehnică, ce prezintă succint conceptele necesare realizării lucrării precum concepte de bază Internet of Things, o descriere a protocolului MQTT precum și modelul folosit de acesta, Publish-Subscribe, concepte specifice de containerizare și izolare a proceselor. Alte concepte atinse sunt și cele de Spring Boot, Spring Security, o introducere în inteligența artificială și o scurtă descriere a protocolului HTTPS.

Capitolul 3, descrierea soluției, prezintă arhitectura concepută precum și fluxul datelor. De asemenea, se oferă mai multe detalii despre nodurile IoT, gateway, microserviciile dezvoltate, interfața grafică disponibilă utilizatorului, cât și algoritmii de inteligență artificială folosiți.

Capitolul 4, concluzii, include avantajele și dezavantajele acestei soluții, precum și lucruri de avut în vedere pentru dezvoltări viitoare.

# Preliminarii

## 2.1. Internet of Things

Conceptul de Internet of Things reprezintă modul prin care obiectele fizice incorporate cu senzori și software sunt folosite pentru a colecta și distribui prin internet date din mediul înconjurător către alte dispozitive, pentru a putea fi apoi prezentate și analizate.

Într-un ecosistem IoT vom găsi un dispozitiv smart care captează datele și le trimite mai departe către un IoT gateway, ce are capabilități de a colecta și analiza local datele sau de a le transmite la rândul său către o altă aplicație.

Diagram

Description automatically generated

Figură 1. Arhitectura IoT

Arhitectura IoT este formată din trei nivele de bază:

* Nivelul de percepție – este primul nivel ce integrează obiectele fizice: senzorii sau actuatorii. Obiectivul acestuia este de a colecta informația asociată și de a o transmite către nivelul superior.
* Nivelul rețea (de transmisie) – este nivelul intermediar destinat Gateway-ului pentru a procesa și transmite datele colectate de la nodurile IoT. Acesta face legătura între senzori și server.
* Nivelul de aplicație – este ultimul nivel și poate fi reprezentat prin mai multe servere sau un sistem Cloud unde datele pot fi afișate, analizate și agregate.

## 2.2. Publish/Subscribe

Modelul Publish-Subscribe este un mod de comunicare asincron serviciu către serviciu. Acesta este un design pattern ce oferă o cale de comunicare între publishers (host) și subscribers prin mesaje (evenimente).

Fiecare mesaj are asociat un Topic. Clientul Publisher trimite mesaje cu un anumit Topic, iar Clientul Subscriber abonat la acel topic va primi mesajele.

La baza acestui model de arhitectură stă un intermediar numit broker de mesaje cunoscut atât de publisher cât și de subscriber. Acesta primește mesajele publicate și le trimite către subscriberii ce s-au abonat topicului, păstrând identitatea fiecărei componente anonimă.

Diagram

Description automatically generated

Figură 2. Modelul Publish/Subscribe

Sursa <https://aws.amazon.com/pub-sub-messaging/>

Avantajele acestui model sunt:

* Decuplarea între componente – permite izolarea completă a componentelor, făcând sistemul ușor de menținut.
* Elasticitatea soluției – o soluție ce integrează acest model de mesagerie este elastic deoarece nu depinde de numărul de publisheri sau subscriberi abonați.
* Sporește receptivitatea - fiind o soluție asincrona, transferul de mesaje nu blochează publisherul, iar subscriberul este ocupat doar atunci când se abonează unui topic.

## 2.3. MQTT

### 2.3.1 Protocolul MQTT

MQTT (Message Queuing Telemetry Transport) este un protocol lightweight folosit în arhitecturile de tip publish/subscribe și destinat comunicării machine to machine în medii cu lățime de bandă redusă.

Acesta este un protocol de mesagerie în timp real, oferind o trimitere rapidă a datelor. Mesajul transmis pe rețea conține ca parametri:

* Payload-ul ce este reprezentat de încărcătura de date înregistrate de nod.
* QoS asigură trimiterea mesajului către subscriber. Un nivel înalt de QoS oferă o siguranță mai mare ca mesajul să fie trimis, dar și o latență și o lățime de bandă mai mari. Nivelele QoS sunt:
* Maximum o data (0), fără confirmare. Nu oferă o garanție a sosirii mesajului.
* Minimum o data (1), cu confirmare.
* Fix o data (2), cu un hasdshake realizat in 4 pași, fiind cel mai sigur, dar și cel mai încet.
* Colecție de proprietăți
* Numele topicului

MQTT este identificat ca fiind un protocol lightweight deoarece mesajele pot fi de minimum 2mb, respectiv 256mb maximum.

### 2.3.2 Protocolul WebSocket

WebSocket este un protocol de comunicare ce vine peste TCP/IP, folosit în comunicarea server-client. Acesta este un protocol bidirecțional, stateful, păstrând conexiunea între client și server deschisă până când unul dintre cei doi o va închide.

Acesta este folosit în special în aplicațiile de tip real time, oferind posibilitatea de a transmite datele încontinuu și mult mai rapid, folosind aceeași conexiune deja deschisă.

### 2.3.2 MQTT Broker

Serverul de mesaje sau MQTT Broker este responsabil de gestionarea conexiunilor clienților MQTT , de interceptarea request-urilor publish, subscribe, unsubscibe, cât și de distribuirea mesajelor.

Printre brokerele MQTT existente se află și Eclipse Mosquitto, un broker open source ce implementează versiunile 5.0, 3.1.1 si 3.1 ale protocolului MQTT. Mosquitto este un broker lightweight, fiind dezvoltat în limbajul de programare C. Acesta oferă și o bibliotecă în C pentru implementarea clienților MQTT .

### 2.3.2 MQTT Client

Clientul MQTT este o aplicație ce implementează MQTT peste TCP/IP pentru a trimite și primi mesaje. Clientul MQTT ce trimite mesajele către broker este Publisher, iar cel ce le primește de la broker este Subscriber. Fiecare client poate fi Subscriber, Publisher, dar și ambele in același timp.

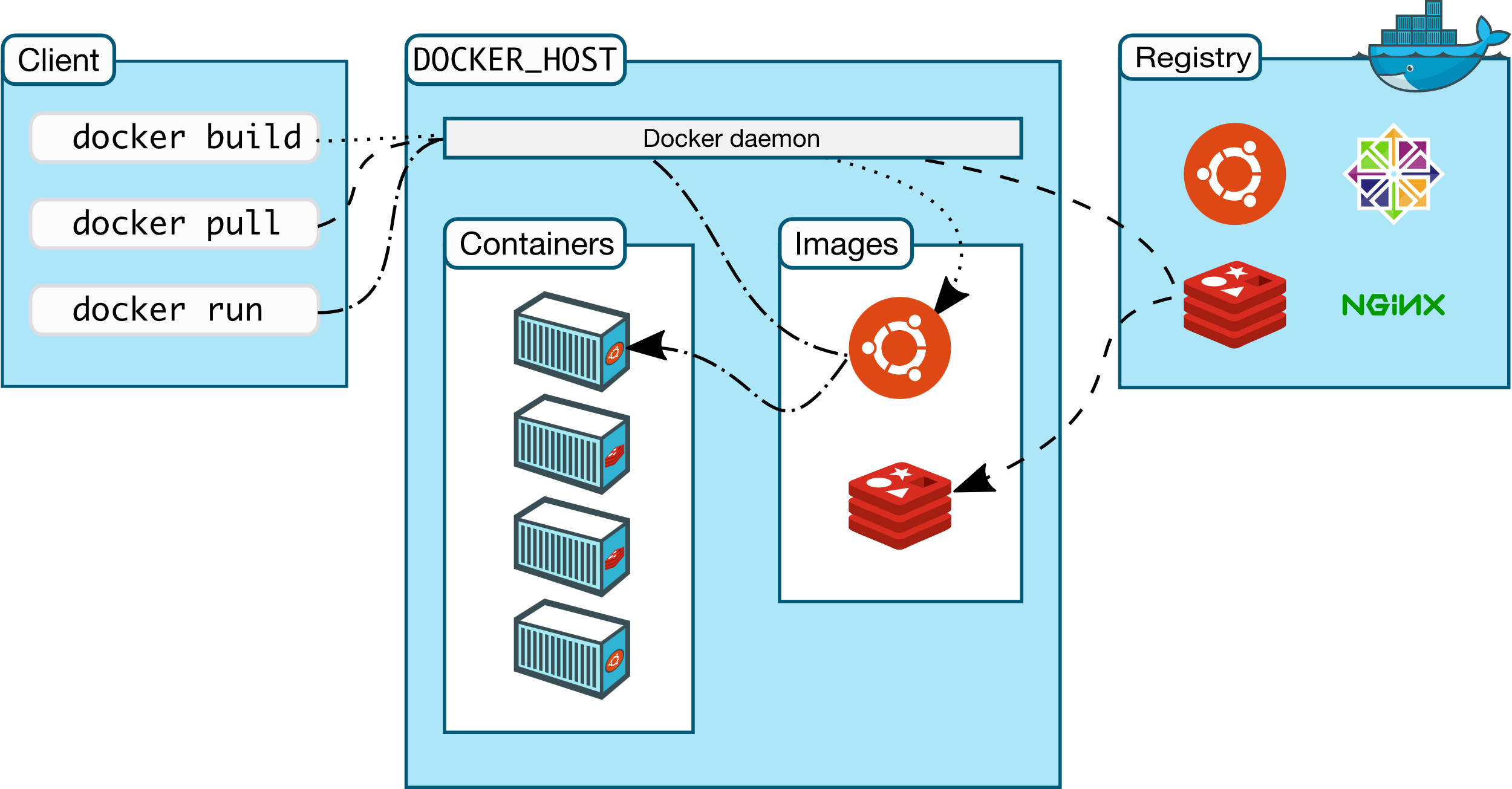
Eclipse oferă un client Paho MQTT ce se poate conecta la MQTT broker și implementează versiunile 5.0, 3.1.1, și 3.1 ale protocolului MQTT.

## 2.5. Docker, Containerizare si izolare

Docker este o platforma definita de mai multa componente, menita sa faciliteze dezvoltarea, livrarea si rularea de aplicatii intr-un mediu izolat. Acest lucru permite eliminarea diferentelor de sistem, asigurand o compatibilitate maxima si un timp de *setup* minim. Astfel, mediul este uniform intre masinile folosite in dezvoltare si cele din infrastructura.

Architectura folosita de catre Docker este client-server, unde un *daemon* folosit pentru impacheterea imaginilor, rularea containerelor si distribuirea lor este serverul, iar clientul este folosit pentru management. Interfata dintre acestia o reprezinta un API REST, fie expus pe socket-uri UNIX sau prin intermediul unui adaptor de retea.

Serverul este reprezentat prin procesul *dockerd*, ce asculta pentru noi cereri si administreaza obiecte Docker precum imagini, containere, retele si volume, insa poate si orchesta alti *daemons* in modul Docker Swarm pentru a rula distribuit containerele. Clientul *docker* este modul primar pentru a interfata cu serverul, folosirea de comenzi precum *docker run* fiind traduse in apelul REST corespunzator si apoi tratat de catre server. De asemenea, un client *docker* poate interfata cu mai multe servere.



Figură 3. Arhitectura Docker

Sursa: <https://docs.docker.com/get-started/overview/>

Serverul administrează mai mult obiecte precum imagini, containere, retele, volume si alte tipuri de obiecte.

Imaginile reprezinta sabloane folosite pentru a instantia containere. Acestea pot fi descarcate atat dintr-un Docker Registry precum Docker Hub, dar si construite prin folosirea unor fisiere de tipul *Dockerfile*, ce specifica o imagine de baza, de exemplu OpenJDK 17, comenzi aditionale ce trebuiesc rulate, cum ar fi copierea unei arhive *jar*, dar si comanda ce va fi executata la creearea unui container cu aceasta imagine, cum ar fi invocarea interpretorului *java*. Fiecare instructiune creeaza cate un strat al imaginii, permitand astfel *caching-ul* ce face procesul de reconstruire al imaginii mult mai rapid, dar si reutilizarea de imagini ce reduce seminificativ cerintele la nivelul spatiului de stocare.

Containerul reprezinta instantierea unui imagini, ruland implicit in izolarea fata de sistemul gazda folosind functionalitatea de *namespaces* oferita de nucleul Linux. Aceasta izolarea poate fi configurata prin atasarea unui retele Docker, ce perminte comunicarea cu alte containere, sau prin atasarea la reteaua sistemului gazda sau atasarea unui volum prin montarea de directoare. Avantajul oferit de un container fata de solutiile clasice de virtualizare o consta in reutilizarea elementelor din sistemul gazda cum ar fi sistemului de operare si librarii esentiale ce duce la o reducere semnificativa a resurselor consumate. Pentru orchestrarea unui solutii bazate pe mai multe containere, folosirea lui *docker compose* este recomandata. Acesta este un instrument pentru definirea si rularea de aplicatii cu mai multe containere, unde un fisier YAML este folosit pentru a declara structura solutiei. Apoi, comanda *docker compose up* poate fi folosita pentru a ridica intreaga solutie intr-un mod facil si rapid.

## 2.6. Machine Learning

Machine Learning este stiinta programarii calculatoarelor astfel incat ele sa invete din date. [2] Pentru a fi util, un sistem Machine Learning trebuie prima data antrenat cu date de test, iar in functie de tipul sistemului pot fi folosite si etichete, valoarea corecta a prezicerii cu ajutorul careia sa fie efectuat antrenamentul. Clasificarea acestora poate fi facuta pe baza mai multor criterii precum:

* Necesitatatea supervizarii de catre un operator uman:
  + Supervizat: sunt folosite etichete in procesul de antrenare
  + Ne-supervizat: nu sunt folosite etichete in procesul de antrenare
  + Semi-supervizat: este antrenat initial in modul supervizat, apoi rafinat in modul ne-supervizat
  + Invatare ranforsata: invatare bazata pe recopemense
* Daca pot invata incremental:
  + Invatare online: este antrenat incremental folosit fie secvential cu date individuale, fie in mini-batch-uri
  + Invatare batch: sistem ce nu poate invata incremental, ce trebuie antrenat folosind toate datele disponibile
* Modul de invatare:
  + Bazata pe instanta: invata datele
  + Bazata pe model: sistemul creeaza un model pe baza datelor de antrenament

Invatarea bazata pe model este cea mai intalnita forma de Machine Learning, fiind disponibile modele atat simple precum regresia lineara, pana la modele complexe precum padurile stohastice. Antrenarea greutatilor se face fie folosind o functie de fitness, masurand nivelul cu care modelul surprinde caracteristicile datelor, scopul fiind maximizarea, fie o functie de cost, scopul fiind minimizarea. Pentru probleme ce necesita valori discrete, regresiile, o functie de cost este favorabila. Printre cele mai populare functii de cost se afla Eroarea Medie Absoluta, dar si Radacina Erorii Medii Patratice.

Retelele neuronale reprezinta evolutia modelelor clasice de Machine Learning, creeand o paralela intre modul in care functioneaza creierul uman si felul in care acest sistem proceseaza datele, conectand mai multi neuroni responsabili cu calcule simple. Unitatea de baza a unei retele neuronale este *perceptronul*, ce accepta o valoare de intrare, aplica o functie de activare apoi trimite valoare rezultata catre urmatorul neuron. In functie de necesitatea aplicatiei, exista mai multe functii de activare ce pot fi folosite, precum:

* Liniara: nu altereaza valoarea
* Functia de pas Heaviside: transforma valorile mai mici decat 0 in 0 si cele mai mare in 1
* Unitate liniara rectificata: transforma valorile mai mici decat 0 in 0, iar restul nu sunt alterate
* Functia sigmoid: folosita cand o probabilitate este necesara, avand valori intre 0 si 1
* Functia tangenta hiperbolica: similara cu functia sigmoid, insa cu valori intre -1 si 1

Similar unui sistem clasic de inteligenta artificiala, antrenamentul este necesar. Acesta este efectuata prin procesul de *back-propagation*, bazat pe o functie de cost. Concret, un *forward-pass*, ce reprezinta inmultire de matrici intre valorile de neuronilor si greutati, este efectuata cu greutatile initializate stohastic, iar apoi costul relativ cu eticheta de antrenament este calculat. Apoi, folosind procesul de *back-propagation*, greutatile sunt ajustate conform tehnicii de optimizare folosite, multiplicata cu rata de invatare.

Diagram

Description automatically generated

Retea neuronala simpla

Cazuri particulare ale acestor retele neuronale sunt retele neuronale convolutionale si cele recurente, ce oferta rezultate sporite pentru cazuri bine definite. Retele neuronale convolutionale sunt cel mai des folosite pentru clasificarea de imagini, fiind definite de stratul convolutional ce ofera invarianta spatiala. Retelele neuronale recurente sunt folosite adesea pentru reprezentari secventiale in timp, precum procesarea limbajului natural sau regresii pe serii de timp, fiind capabile sa memoreze date anterioare folosite in efectuarea prezicerii curente.

# Descrierea soluției

## 3.1. Arhitectura soluției

Scopul acestei lucrări de disertație este de a proiecta o soluție de tip casă inteligentă modulară și extensibilă. Arhitectura soluției constă în mai multe noduri IoT de design propriu, gateway IoT și mai multe micro-servicii lansate într-o instanță cloud de tip IaaS.

Diagram

Description automatically generated

Figură 3. Arhitectura soluției

Nodurile IoT folosite sunt plăci de dezvoltare ESP8266, conectate la rețeaua WiFi și au atașați diverși senzori pe interfața serială. Datele acestora sunt transmise în clar și transportate prin intermediul protocolului MQTT către broker-ul găzduit pe gateway-ul IoT. Gateway-ul folosit este un Raspberry Pi 4 cu 1GB RAM rulând sistemul de operare Raspberry Pi Lite.

La nivel de gateway, a fost configurat broker-ul Eclipse Mosquitto care ascultă pe portul TCP 1883 noi cereri de conectare. Broker-ul va distribui mesajele publicate către abonații acestora, având de asemenea posibilitatea păstrării ultimului mesaj pentru a fi distribuit noilor abonați, cât și ajustarea serviciului calității, iar autentificarea clienților se face pe bază de nume de utilizator și parolă.

În componența soluției se află de asemenea și o aplicație NodeJS ce deservește două scopuri, client MQTT și client HTTPS. Aceasta se abonează la broker pentru diverse topicuri la care sunt publicate datele senzorilor urmând a fi împinse către cloud prin intermediul HTTP POST. Pentru a rezuma, gateway-ul IoT va concentra toate datele primite în Intranet de la diversele noduri, urmând a le împinge în cloud pentru stocare și analizare. Clientul de MQTT ales este MQTT.js ce este un proiect Open-Source ce permite conectarea folosind protocolul TCP.

Diagram

Description automatically generated

Figură 4 Modelul publish/subscribe.

În cloud, datele sunt trecute printr-un microserviciu web facing, gateway-ul, urmând a fi trimise mai departe către microservicii private. Aceste microservicii sunt găzduite într-o instanță Oracle Cloud Instance și rulate în containere Docker pentru a facilita punerea în funcțiune a aplicației, cât și securitatea prin izolarea inherentă oferită de containerizare. De asemenea, modul Swarm este activat, ce determină un grad sporit de reziliență la diverse probleme ce pot apărea pe durata execuției, cât și un răspuns la cerere foarte rapid prin scalabilitatea oferită de către acesta. Pentru dezvoltarea microserviciilor private a fost folosit atât Spring Boot și Java, pentru serviciul OLTP si pentru cel de tip Data Warehouse, cât și Flask și Python pentru dezvoltarea microserviciului responsabil cu operațiile de tip Machine Learning. Pentru a facilita utilizarea acestei soluții, o interfață grafică dezvoltată în Next.js folosind componente din populara librărie de React Material UI a fost adăugată. Framework-ul acesta a fost folosit pentru performanța deosebit de ridicată pe care o oferă, cât și pentru posibilitățile de optimizare pentru motorul de căutare.

Toate cererile HTTP sunt adresate gateway-ului, care pe lângă rolul principal de rutare mai are și rol de autentificare și autorizare. Cererile către interfața grafică sunt permise și utilizatorilor ne-autentificați, iar pentru utilizarea microserviciilor, autentificarea este necesară. Autentificarea și autorizarea utilizatorilor este facilitată de utilizarea furnizorului de identitate Auth0, și este realizată folosind JSON Web Tokens, permițând decuplarea stării de autentificare de către server-ul unde a fost efectuată.

Această arhitectură a fost aleasă pentru flexibilitatea și extensibilitatea pe care le oferă, având de asemenea un grad sporit de securitate. Toate nodurile vor publica în același topic, având necesară zero configurare pentru introducerea de noi senzori în rețea. Pentru a putea accesa datele de la noduri, este necesară asocierea gateway-ului unui cont. Acest proces este facilitat de către interfața grafică dezvoltată în Next.js, iar de îndată ce împerecherea a fost încheiata cu succes, datele vor fi împinse în cloud către gateway ce va delega rezolvarea cererii către microserviciul OLTP dedicat stocării informației într-o bază de date relațională Oracle Express Edition 21c ce efectuează operații de tip CRUD. Acest pas va crea un strat de persistență și facilitează accesul la date printr-un format ușor de manipulat. De asemenea, acest microserviciu este folosit și în cadrul interfeței grafice pentru a popula date semnificative despre valorile actuale ale senzorilor, precum și date salvate de către utilizator despre locuința sa și amplasamentul senzorilor per cameră.

Pe lângă microserviciul OLTP, a mai fost implementat serviciul ce permite accesarea datelor structurate în modul Data Warehouse ce sunt stocate într-o bază de date Oracle Express 21c și servite prin intermediul protocolului HTTPS, dar și serviciul responsabil cu efectuarea operațiilor Machine Learning folosind un model pre-antrenat.

De asemenea, această soluție este ușor scalabilă și fiabilă, fiind concepută cloud-native și fault-tolerant. Modul Docker Swarm permite aprovizionarea rapidă de noi containere în cazul în care procesul inițial este ucis de către o eroare, dar și creșterea sau scăderea numărului de replici pentru a ajusta corespunzător nivelul de încărcare.

## 3.2. Lista de componente

Tabel 1. Lista de componente

|  |  |  |  |
| --- | --- | --- | --- |
| Componentă | Preț(lei) | Cantitate(buc) | Preț Total(lei) |
| Placă de dezvoltare ESP8266 | 36.99 | 2 | 73.98 |
| Senzor de temperatură și umiditate DHT11 | 8.98 | 2 | 17.96 |
| Raspberry Pi 4 1 GB RAM | 173 | 1 | 173 |
| Card MicroSD 32GB | 36 | 1 | 36 |
| Total(lei) | 300.94 | | |

1. Placa de dezvoltare ESP8266

ESP8266 este un microchip cu capabilități WiFi produs de către compania chineza Espressif. Acesta este definit de o arhitectură pe 32 de biți, având un procesor Tensilica Diamond Standard 106Micro tactat în mod implicit la 80MHz, însă care poate fi setat să ruleze și la o frecvență de 160Mhz. Ca și memorie volatilă, prezintă 32KB rezervați pentru instrucțiuni, și 80KB disponibili pentru rularea de programe de către utilizator.

Un factor important în alegerea unei plăci de dezvoltare cu acest chip a fost suportul pentru stiva TCP/IP precum și suportul pentru funcții criptografice cu o intensitate scăzută, pentru funcțiile de rezumat și calculul de coduri de autentificare a mesajelor.

Două plăci de acest tip au fost folosite pentru realizarea lucrării, ele având rolul de a transmite date de la senzori către gateway.

1. Senzor de temperatura si umiditate DHT11

DTH11 este un senzor de temperatură și umiditate ce produce un semnal digital pe pin-ul de date. Temperatura este măsurata folosind un termistor cu coeficient negativ iar umiditatea relativa este măsurata folosind un senzor capacitiv. Senzorul poate citi valori ale umidității în plaja 20-90% RH cu o acuratețe la măsurare de +/- 5% RH. Temperatura poate lua o valoare între 0 și 60◦C cu o acuratețe de +/-2◦C. Senzorul permite un voltaj de intrare în plaja 3.3V și 5V.

1. Raspberry Pi 4 1GB RAM

Raspberry Pi 4 este o placa de dezvoltare ce conține un procesor actualizat față de generațiile precedente, având o arhitectură pe 64 de biți și un procesor cu 4 nuclee tactate la 1.5GHz ce are incorporat un disipator de căldura din metal. De asemenea, pe partea de conectivitate acesta are 3 port-uri USB 3.0, placa de rețea Wi-Fi ce funcționează atât în modul 2.4GHz cât și 5GHz, Bluetooth 5.0 Low Energy, Gigabit Ethernet cu capabilități de Power over Ethernet, dar și porturi Micro HDMI ce suportă rezoluții până la 4K.

Versiunea folosită pentru realizarea proiectului este dotată cu 1GB de memorie volatilă și rulează sistemul de operare Raspberry OS Lite. Această placă de dezvoltare este folosită pe post de gateway, susținând infrastructura necesară conectării senzorilor și transmiterea de date atât de la senzor la gateway cât și de la gateway în cloud. Concret, gateway-ul găzduiește un broker Mosquitto MQTT cât și aplicația ce colectează datele publicate și le transmite securizat folosind HTTPS către Cloud.

1. Card MicroSD 32GB

Cardul MicroSD este folosit pentru a dota Raspberry Pi-ul cu memorie non-volatilă, alegându-se capacitatea de 32GB pentru a permite instalarea sistemului de operare cât și a tuturor aplicațiilor necesare.

Conectarea fizică a senzorilor cu placa de dezvoltare a fost efectuată conform tabelului de mai jos.

Tabel 2. MAPPAREA!?!?!

|  |  |  |
| --- | --- | --- |
| Componentă | Pin Componentă | Pin ESP8266 |
| DHT11 | VCC | 3V |
| DATA | D1 |
| GND | G |

Diagram, schematic

Description automatically generated

Figură 5. Schema electrică

Pentru stratul de percepție al nodului, senzorul de umiditate și temperatură DHT11 a fost conectat la placa de dezvoltare ESP8266. Aceasta placă de dezvoltare este dotată cu 17 pini GPIO, având suport pentru interfețele SPI, I2C, UART dar si Analog-to-Digital pe 10 biți. Alimentarea senzorului a fost efectuată folosind pinul de 3V prezent pe placă, voltaj permis conform fișei tehnice a acestuia. Pinul de date a fost ulterior conectat direct către pinul GPIO5, marcat pe placă ca și D1, iar împământarea a fost legată de pinul G.

## 3.3. Descrierea nodului IoT

Nodurile IoT sunt constituite din plăci de dezvoltare ESP8266 având conectate pe interfața serială diverși senzori. Un exemplu de acest fel ar fi DHT11, senzor de umiditate(%) și temperatură(◦C). Acestea comunică cu gateway-ul prin intermediul protocolului la nivel de aplicație MQTT, iar la nivel de transport TCP.

Acestea sunt conectate la rețeaua Wi-Fi de tip intranet, ele nefiind menite să se conecteze în internet din considerente de securitate. Datele recepționate de la senzori sunt trimise prin intermediul protocolului MQTT. Concret, nodurile publică pe același topic, fiind clienți MQTT. Plasarea nodurilor în intranet a fost aleasă strategic, permițând economisirea bateriei și limitarea consumului de resurse prin renunțarea implementării securității la nivel de transport prin TLS, bazându-se strict pe confidențialitatea oferită de către WPA3-PSK la nivelul legăturii de date. Autentificarea nodurilor la broker se face pe baza unor credențiale arse in EEPROM. De îndată ce nodurile sunt conectate la WiFi, acestea vor începe să trimită date către broker, nefiind necesară o configurare adițională.

Dezvoltarea componentei software aferente nodului IoT a fost făcută în framework-ul Arduino, ce permite folosirea limbajului C++ și o paradigmă orientată obiect. De asemenea, conform sugestiei din documentație [2], abstractizarea sistemului de asamblare a binarului și de gestionare a dependențelor a fost realizată cu ajutorul extensiei de Visual Studio Code numită PlatformIO. Această extensie permite de asemenea și un sistem avansat de completare și sugestie a codului, precum și suport pentru depanare și o portabilitate a codului ridicată. Compilatorul folosit este o versiune modificată a GCC (GNU Compiler Collection), ce suportă standardul C++ 11 și un subset al librăriei *std* pe lângă librăriile specifice framework-ului Arduino, dar și librării destinate accesării funcțiilor prezente pe ESP8266, precum Wi-Fi sau posibilitatea de a restarta nodul programatic. Pe langa librariile standard oferite, dependenta *PubSubClient* a fost adaugata prin intermediul managerului oferit de catre PlatformIO. Aceasta include un client de MQTT sub forma unei clase omonime cu numele librariei, ce primeste ca parametru al constructorului mediul peste care este efectuata comunicarea, in acest caz fiind Wi-Fi prin intermediul *WiFiClient*, ce face parte din libraria standard *ESP8266WiFi.h*.

Structura codului este marcata de cele 2 metode specifice framework-ului Arduino, *setup* si *loop*, unde *setup* reprezinta logica de initalizare a nodului ce este executa de fiecare data nodul este pornit, iar *loop* cum este si sugerat prin nume, ruleaza in bucla logica de business.

Diagram

Description automatically generated

Figură 6. Schemă inițializare nod

Initializarea nodului este marcata de 2 procese, conectarea la reteaua Wi-Fi si conectarea la broker-ul MQTT. Conectarea la reteaua Wi-Fi este efectuata folosind libraria standard Wi-Fi *ESP8266WiFi.h*, operatiunile fiind abstractizare prin obiectul *WiFi*. Placa de retea este setata in modul *Station* ce permite conectarea la puncte de access, iar credentialele sunt citite din EEPROM si folosite ca parametrii pentru metoda *begin*. Acest proces este efectuat in bucla cu o intarziere de 5 secunde pana ce conexiunea va fi stabilita cu success.

Text

Description automatically generated

In urma stabilirii conexiunii Wi-Fi, conectarea la broker-ul MQTT este incercata. Metoda *setServer* aferenta clientului MQTT este apelata, folosind ca parametrii IP-ul si portul broker-ului, iar apoi identificatorul unic format in urma concatenarii sirului de caractere “esp8266-“ si adresa MAC a placii de retea este creat, ce va fi ulterior utilizat ca parametru al metodei *connect*. Daca metoda intoarce un cod de eroare, nodul se va restarta si va reincerca conexiunea, altfel, initializarea va fi completa si metoda *loop* va prelua controlul executiei.

Text

Description automatically generated

Metoda *loop* executa logica de business in bucla, folosind aceeasi metoda *connectWifi* pentru a asigura permanenta conexiunii, urmand apoi sa foloseasca metoda *loop* a clientului MQTT. Apelarea acesteia este necesara deoarece clientul functioneaza in mod sincron, metoda fara de care schimbul de informatii dintre broker si client nu s-ar efectua. Nodul IoT manipuleaza transparent schimbul de date, colectarea de date din senzor fiind delegata catre o metoda specializata pentru a pastra flexibilitatea solutiei. Datele sunt publicate in topicul *data* sub format JSON, avand ca proprietati *origin*, ce reprezinta identificatorul nodului, tipul datei si valoarea acesteia.

Text

Description automatically generated

Logica de business este efectuata in bucla la o frecventa de 5 secunde, aceasta fiind impusa prin apelarea metodei *delay* specifice Arduino Framework.

## 3.5. Descrierea Gateway-ului

Gateway-ul IoT este constituit dintr-o placa de dezvoltare Raspberry Pi 4 cu 1 gigaoctet de memorie volatila si 32 de gigaocteti memorie non-volatila, ruland sistemul de operatre Raspberry Pi OS Lite. Acest sistem de operare are un consum scazut de resurse, fiind gandit pentru a fi folosit strict prin intermediul SSH, neavand vreun mediu de lucru instalat.

Acest dispozitiv gazduieste atat broker-ul MQTT, cat si o aplicatia NodeJS ce este subscrisa la topicul *data* si trimite toate mesajele mai departe catre Gateway-ului din cloud. Concret, scopul acestuia este sa centralizeze toate datele de la nodurile reprezentate de dispozitive cu putere de procesare redusa si sa le impinga pentru a fi procesate si persistate.

Broker-ul MQTT folosit este Mosquitto dezvoltat de catre Eclipse Foundation. Acesta este configurat sa asculte pe portul TCP 1883 pentru noi conexiuni, permitand conectarea fara a fi necesare credentiale.

Text

Description automatically generated

Conform protocolului MQTT, acesta suporta multiple strategii de trimitere a mesajelor, configurabil prin intermediului parametrului QoS (Quality of Service):

* 0, modul „fire and forget”, ce garantează trimiterea mesajului o singură dată, fără a mai fi necesară confirmarea primirii
* 1, modul „at least once”, ce garantează trimiterea mesajului cel puțin o dată, fiind necesară confirmarea
* 2, modul „exactly once”, ce garantează trimiterea mesajului exact o singură data, fiind necesară confirmarea și lipsa duplicatelor

Păstrarea ultimului mesaj cu scopul distribuirii către noii abonați este de asemenea importantă pentru topicurile actualizate rar.

Gateway-ul nu deschide portul 1883 pentru comunicatii din internet, fiind astfel restrictionata utilizata broker-ului doar de catre utilizatorii aceleasi retele, precum nodurile IoT.

Pe langa broker-ul MQTT, dispozitivul mai gazduieste si o aplicatie NodeJS scrisa in Typescript ce este abonata la topicul *data* si impinge mai departe datele in cloud prin intermediul protocolului HTTPS pentru a asigura confidentialitatea incarcaturii. Pentru dezvoltarea acestei aplicatii au fost folosite pachete precum *mqtt*, ce reprezinta un client MQTT bazat pe conceptul de evenimente, *npmlog*, ce permite inregistrarea de mesaje fie cu scopul informarii, fie de inregistrare a erorilor produse sau strict pentru depanare, dar si *node-fetch*, ce permite utilizarea Fetch API in mediul Node JS.

Datele necesare configurarii acestei aplicatii au fost externalizate in fisiere JSON precum *credentials.json*, unde sunt stocate informatii necesare conectarii catre broker, dar si *jwt.json* unde sunt stocate variabile necesare obtinerii unui JSON Web Token pentru autentificarea cererilor catre Cloud Gateway.

Conectarea catre broker este efectuata prin apelul functiei *connect* aferente obiectului *mqtt*, ce va declansa publicarea de evenimente pe bucla NodeJS. Eventimentului *connect* ii este asociat un callback ce efectuaza abonarea la topicul *data*, iar evenimentului *message* ii este asociat un callback ce trimite prin HTTPS POST datele catre Cloud Gateway.

Text

Description automatically generated

Autentificarea dispozitivului este efectuata prin intermediul unui token purtator obtinut in urma Client Credentials Flow oferit de catre Auth0, ce implica un HTTPS POST catre furnizorul de identitate cu scopul obtinerii unui JSON Web Token. Datele necesarii acestei cereri sunt citite din fisierul *jwt.json* ce contine date despre client dar si audienta ceruta.

Text

Description automatically generated

In urma conectarii la broker si obtinerii tokenului purtatorul, aplicatia va semnala disponibilitatea IoT gateway-ului catre cloud printr-un apel HTTPS POST. Daca raspunsul contine campul *pairedTo* gol, gateway-ul va fi pus in modul imperechere ce determina apelul acestei metoda in bucla pana ce un user asociaza aparatul folosind codul de imperechere specific. Dupa imperechere, aplicatia va trimite in cloud toate inregistrarile publicate pe topicul *data*.

## 3.6. Microservicii dezvoltate

Partea integrantă a soluției este constituită de microserviciile dezvoltate folosind fie framework-ul Spring Boot in Java, fie Flask in Python, ce sunt găzduite pe o instanță de cloud Oracle. Arhitectura aceasta a fost folosită în detrimentul monolitului din prisma beneficiilor multiple ce sunt oferite pe partea de distribuire și mentenanță, în ciuda efortului de dezvoltare asociat.

Funcțional, acestea realizează fie rol de aplicație OLTP, asociată unei baze de date relațională Oracle Express Edition 21c, fie rol de aplicație Data Warehouse împerecheată cu o aceeași bază de date menționată anterior, însă folosind altă schemă, dar și o aplicație specializată pentru machine learning. Toate cererile sunt securizate prin intermediul protocolului HTTPS, și autenficate folosind JSON Web Tokens emise și verificate de către autoritatea emitentă Auth0.

Diagram

Description automatically generated

Figură 7. Diagrama microserviciilor

Tehnic, toate aceste cereri sunt trecute printr-un gateway care prima dată autentifică în funcție de calea folosită și apoi rutează către microserviciul responsabil. Partea de autentificare este facilitată de folosirea starter-ului Spring Cloud Security dar și al starter-ului ce verifica transparent JSON Web Token-urile primite în câmpul Authorization al cererii, numit OAuth 2.0 Resource Server. Transparența aceasta este obținută prin folosirea datelor din fișierul de configurare pentru a injecta un Bean ce folosește URL-ul autorității emitente pentru a verifica validitatea token-ului. Pentru a modifica setările implicite care obligă autentificarea fiecărui request, a fost creat un *WebFilter* ce permite acces neautentificat către interfața grafică pentru a permite utilizatorilor să-și creeze cont. Crearea de utilizatori, emiterea și verificarea de token-uri sunt delegate către Auth0, permițând simplificarea arhitecturii de securitate.

Text

Description automatically generated

Figură 8. Setări de securitate aferente Gateway-ului

Partea de rutare este implementată folosind Spring Cloud Gateway, însă și alte microservicii destinate infrastructurii au trebuit să fie adăugate. Printre acestea se numără un microserviciu de Service Discovery, dar și un microserviciu destinat servirii de configurații dintr-un repository de GitHub.

Microserviciul destinat configurațiilor este dezvoltat tot în Spring Boot, folosindu-se de starter-ul Spring Cloud Config Server. Acesta folosește fișierul de configurare pentru a-și extrage informațiile necesare creării unui server ce încarcă în timp real din GitHub fișiere YAML ce urmează a fi servite către alte microservicii. Pentru rezolvarea fișierului corect, formatul *nume-profil.yml* trebuie să fie folosit, exemplul fiind *gateway-prod.yml*, unde *prod* este numele profilului. Pentru a evita eventualele probleme asociate pornirii multiplelor servicii, portul 8800 a fost folosit. Concret, fișierele de configurare pentru fiecare microserviciu au fost încărcate într-un repository privat, iar credențialele de autentificare au fost plasate în fișierul *application-prod.yml* ca și nume de utilizator și token de acces generat prin intermediul setărilor destinate dezvoltatorilor din cadrul interfeței web.

Rezolvarea serviciilor la momentul rulării este realizată prin intermediul microserviciului de Service Discovery, dezvoltat de asemenea în Spring Boot și folosind starter-ul Netflix Eureka Server. Având în vedere existența server-ului de configurări, aceasta este încărcată la momentul inițializării, declarând în fișierul împachetat cu microserviciul doar numele aplicației și URL-ul serverului. În cadrul fișierului servit din repository-ul de GitHub, portul a fost setat să fie 8761, iar clientul de Eureka configurat pentru a evita înregistrarea serverului.

Text

Description automatically generated

Figură 9. Configurarea microserviciului de Service Discovery

Un alt aspect notabil aparținând infrastructurii o constituie folosirea de Zipkin, un sistem distribuit de tracing ce permite o mult mai ușoară vizualizare a cererilor, precum și depanarea problemelor de latență întâlnite.

Revenind la rutarea cererilor de către gateway spre celelalte microservicii, aceasta este făcută declarativ prin intermediul fișierului de configurare.

Text

Description automatically generatedText

Description automatically generated

Figură 10. Configurarea Gateway-ului

Librăria Spring Cloud Gateway a fost aleasă în detrimentul altor soluții similare datorită integrării puternice pe care o oferă cu server-ul de Service Discovery Eureka, permițând rezolvarea serviciului cu cea mai mică încărcătura pentru a servi cererea, dar și integrarea cu librării ce aplică design pattern-ul Circuit Breaker. Acest pattern vine in completarea celui de Retry și este util în cazurile în care erorile apar din cauza unor evenimente ce nu pot fi anticipate, permițând folosirea unei metode de fallback în caz de circuit „deschis”. Librăria Resilience4j a fost aleasă pentru a îndeplini această sarcină, specificând un circuit breaker pentru fiecare cale disponibilă. Ca și client de Service Discovery, starter-ul Netflix Eureka Client a fost folosit, configurarea acestuia făcându-se prin intermediul fișierului YAML, specificând URL-ul serverului, în speță microserviciul de Service Discovery, dar și faptul că acest client n-ar trebui să se înregistreze server-ului. Pentru o mai rapidă rutare a cererilor către serviciul destinație, URL-urile rezolvate de Eureka sunt păstrate într-un cache manageriat prin abstracția specifică Spring oferind implementarea concretă Caffeine injectată în context prin intermediul unui Bean.

Text

Description automatically generated

Figură 11. Configurarea cache-ului din Gateway

Microserviciul OLTP este adresabil folosind prefixul /*iot*, iar scopul acestuia este de a persista datele aferente senzorilor dar și despre locuința utilizatorului și încăperile acesteia. Serviciul este dezvoltat folosind starter-ul Web, ce permite convenție asupra configurării prin folosirea de annotari precum *@RestController* sau *@Service*. Acesta suportă de asemenea și tracing prin configurarea Zipkin, dar și rezolvarea fișierelor de configurare la momentul inițializării prin Spring Cloud Config. Acesta se înregistrează la registrul de aplicații prin clientul Eureka pentru a putea fi identificat de către Gateway. De asemenea, toate cererile sunt autentificate și autorizate prin folosirea de JSON Web Tokens, folosind același starter OAuth 2.0 Resource Server, autoritatea emitentă fiind tot Auth0. Pentru îmbunătățirea performanței, startul de abstractizare a cache-ul oferit de către Spring Boot a fost folosit în conjuncție cu implementarea Caffeine, iar pentru îmbunătățirea rezilienței la erori a soluției a fost activat management-ul tranzacțiilor.

! INSERARE POZA CONFIGURARE

Acesta se folosește de o bază de date Oracle Express Edition 21c pentru a persista datele, folosind driver-ul de JDBC thin. Schema bazei de date a fost generată folosind Hibernate, setând proprietățile aferente în fișierul de configurare a profilului *dev*, urmând a fi creat un fișier SQL la pornirea serviciului.

Text

Description automatically generated with low confidence Diagram

Description automatically generated

Spring Data JPA a fost folosit pentru a crea stratul corespunzător accesului de date, reducând semnificativ codul boiler-plate și îmbunătățind viteza de dezvoltare. Entitățile sunt reprezentate de clase annotate cu *@Entity* iar membri acestora pot reprezenta atât câmpuri, dar și chei primare sau de legătură.Accesul la date este facilitat prin folosirea de repository-uri reprezentate de interfețe ce extind tipuri definite de Spring precum *CrudRepository* sau *PagingAndSortingRepository*, dar și prin declararea de metode folosind limbajul specific domeniului ce vor fi generate la momentul compilării proiectului.

Text

Description automatically generatedText

Description automatically generated

Microserviciul este structurat conform stilului arhitectural REST, fiind dispuse diferite endpoint-uri reprezentate de clase annotate cu *@RestController*. Acestea sunt găsite și injectate de către Spring Boot în context, ulterior fiind expuse printr-un container Tomcat. Au fost create controllere pentru toate entitățile, aplicându-se conceptul *Separation of Concerns,* structurând proiectul în controllere, service-uri și repository-uri, îmbunătățind astfel lizibilitatea proiectului și ușurând viitoarea mentenanță. De asemenea, serviciul nu are nicio dependință exceptând cele de infrastructură, fiind astfel ușor de distribuit și scalat în funcție de încărcătură.

Diagram

Description automatically generated

Fluxul de folosire al acestui serviciu implică crearea de țări și orașe de către administratorul aplicației, urmând ca utilizatorul să-și creeze o casă înainte de orice acțiune de împerechere cu un gateway. Identificarea acestuia se face prin token-ul purtător, urmând să aibă valabilă posibilitatea de salvare a unei case, regăsire a detaliilor acesteia dar și editare a numelui. Pentru a asocia o casă unui user se folosește câmpul *sub* din JWT, iar la salvarea casei se creează și o cameră implicită unde urmează a fi salvați senzorii la momentul împerecherii.

Controller-ul IotGateway are o metodă *signal* destinată a fi apelată de către dispozitivul IoT pentru a semnala disponibilitatea acestuia, trimițând în corpul cererii detalii precum codul de împerechere, tipul acestuia dar și senzorii instalați. Nodul este identificat prin intermediul câmpului *sub* al token-ului trimis, iar dacă nodul este deja împerecheat, este întors ID-ul utilizatorului; altfel, detaliile acestea sunt salvate într-o mapă concurentă, marcând astfel disponibilitatea dispozitivului de a fi împerecheat, urmând să cheme această metodă în buclă până va primi ID-ul unui utilizator. Endpoint-ul *pair* este destinat pentru a fi apelat de către utilizator, trimițând codul mașinii și codul de împerechere pentru a-și însuși dispozitivul, fiind astfel scos din pool-ul dispozitivelor pregătite pentru împerechere. În urma acestor operațiuni, utilizatorul poate edita numele gateway-ului, șterge un dispozitiv împerecheat, dar și întoarcerea tuturor gateway-urilor împerecheate. Un gateway are unul sau mai mulți senzori, editabili folosind endpoint-ul *sensor,* aceștia putând fi mutați în alte camere definite de către utilizator prin intermediul endpoint-ului *room*. În privința endpoint-ului *record*, acesta permite salvarea de noi intrări de către dispozitivul IoT, acesta fiind identificat prin intermediul token-ului purtător, dar și întoarcerea de înregistrări către utilizator în regim paginat, identificarea fiind analog făcută ca și la endpoint-ul precedent.

Microserviciul de Data Warehouse găsit la calea *dw* este introdus în proiect pentru a întruni nevoia de stocare a volumului mare de date istorice asupra cărora se pot dezvolta rapoarte și analize. Acesta folosește o instanță de baza de date Oracle Express Edition 21c, unde s-au agregat datele relevante din schema OLTP.

Diagram

Description automatically generated

Pentru modelarea depozitului de date s-a folosit proiectarea schemei sub formă de stea, având în vedere performanța pe care o aduce stocarea întregii informații într-o singură înregistrare.

Centrul acesteia constă într-o tabelă de fapte, *Record,* aceasta fiind subiectul principal al aplicației și sursa de generare a diferitelor analize. Tabelele dimensiune ce stochează informația adițională tabelului de fapte sunt: *Sensor,* *Address*, *Home*, *Time*.

Diagram

Description automatically generated

Toate tabelele dimensiune au fost rezultate din operații de JOIN, denormalizând schema OLTP astfel:

Tabela *Sensor* stochează datele cu privire la senzor *id, name, description*, dar regăsește și informațiile Gateway-ului cu care este asociat. Așadar acesta are 3 atribute suplimentare *machine\_id, gateway\_friendly\_name, type,* aduse prin relația dintre *Sensor* și *Gateway* din OLTP.

Tabela *Address* menține informații cu privire la oraș și țară, având atributele *id, city, country*, fiind o asociere între tabelele *City* și *Country* din OLTP.

Tabela *Home* stochează date cu privire la locuință, incluzând aici atât home\_name adus din tabela Home, cât și room\_name adus din tabela Room din OLTP.

Tabela dimensiune *Time* este adusă în această schemă pentru a descrie temporal informația, menținând ca atribute *reading\_id, full\_date, year, month, day, quarter*.

Microserviciul de Data Warehouse este o aplicație Spring Boot, utilizând pentru partea de acces la baza de date Spring Data JPA pentru a avea mai multă flexibilitate în vederea creării cererilor de informații ce stau la baza generării analizelor. Schema bazei de date a fost generată folosind Hibernate, urmând a fi creat fișierul SQL al acesteia la pornirea serviciului. Acest microserviciu respectă aceleași caracteristici ca cel OLTP, permițând tracing-ul prin configurarea Zipkin, rezolvarea fișierelor de configurare la momentul inițializării prin Spring Cloud Config și înregistrarea acestuia prin clientul Eureka pentru a putea fi găsit de către Gateway. De asemenea, cererile acestuia sunt autentificate și autorizate folosind JSON Web Tokens, prin starter-ul OAuth 2.0 Resource Server, iar pentru a-i oferi o performanță sporită s-a adăugat implementarea Caffeine.

Astfel, aplicația conține cereri pentru 3 analize: valoarea medie, valoarea minimă și valoarea maximă în funcție de id-ul senzorului și id-ul gateway-ului la care acesta trimite valorile înregistrate, id-ul senzorului fiind primit ca parametru în path-ul requestului aferent.

Text

Description automatically generated

Datele din Data Warehouse sunt colectate din OLTP folosind proceduri stocate, migrarea acestora făcându-se zilnic printr-un job automat, ce se rulează la ora 22:00. Procedura migrate\_*data()* inserează valorile aferente pentru a popula tabelele *Address*, *Home,* *Sensor* și *Record* aplicând multiple operații de JOIN pentru aceasta din urmă pentru a colecta toate valorile atributelor necesare. Procedura insert\_*time*(*)* inserează valorile coloanelor din tabela Time.

Procedurile stocate sunt chemate în metodele din repository-ul *RecordRepository*, care sunt la rândul lor apelate în metodele din *RecordService*, iar mai apoi în cele din *RecordController*, mapând fiecare apel de analiză într-un serviciu de tip *GET* și fiecare apel de inserare într-un serviciu de tip *PUT*.

Text

Description automatically generated

Pe lângă serviciile de OLTP și Data Warehouse, mai sunt prezente și cele de ML și inferență care sunt într-o strânsă legătura.

Serviciul ML este dezvoltat folosind Spring Boot și este disponibil la calea *nn*, fiind de asemenea și client de Eureka ce îi permite Cloud Gateway-ului să ruteze cererea prin rezolvare la run-time a URL-ului. Alte aspecte notabile sunt folosirea starter-ului OAuth 2 Resource Server pentru a valida JWT-urile cu scopul autorizării și autentificării cererilor, rezolvarea proprietăților la run-time prin Spring Cloud Config, folosirea stratului de abstracție a cache-ului împreună cu implementarea Caffeine pentru a îmbunătăți performanța, dar și includerea Spring Data pentru a facilita accesul la baza de date. Persistența este oferită de o instanță de Oracle Express Edition 21c, partajată cu seviciul Data Warehouse pentru a accesa datele formatate conform diagramei stea. Serviciul expune un singur endpoint *predict*, primind ca variabilă a căii ID-ul senzorului pentru care să fie făcută inferența. La nivelul serviciului, acesta verifică existența unui senzor cu acel ID, apoi selectează din baza de date înregistrări aferente senzorului pe ultimele 24 de ore. Acestea sunt apoi formatate ca matrice cu două coloane, oră și valoarea, și trimise către serviciul de Inferență, prin intermediul unui client Feign. Acest client permite un stil declarativ în folosirea de client REST, abstractizând detaliile ce țin de implementare.

Text

Description automatically generated

Serviciul de Inferență este dezvoltat în Python folosind framework-ul Flask pentru a crea un server HTTPS, iar partea de machine learning este construită folosind Tensorflow, Keras și Sci-Kit Learn. Acest serviciu nu necesită autentificare, fiind disponibil doar în Intranet și apelat de către serviciul ML, ne-existând rută a Gateway-ului care să aibă legătura directă cu acest microserviciu. Există un singur endpoint *predict*, apelat folosind metoda HTTP POST, având ca încărcătură 24 înregistrări reprezentând valori din trecut, iar ca răspuns va trimite rezultatul inferenței folosind un model și un scaler pre-antrenat, la care sunt folosite date prelucrate. Din matricea originala ce este caracterizată de două coloane, *ora* și *valoare*, mai sunt introduse coloanele *ora\_cos*, cosinusului orei ajustată, și *ora\_sin*, rezultatul funcției sinus aplicată orei ajustate, urmând a fi îndepărtată coloana *ora*.

Modelul și scaler-ul au fost pre-antrenate si exportate de către altă aplicație Python, fiind definit un model secvențial Keras ce reprezintă o rețea neuronală recurentă de tipul Long Short Term Memory (LSTM) cu 40 de unități și optimizată folosind Adam bazată pe metrica de cost Mean Squared Error (MSE). Rețeaua neuronală recurentă este indicată pentru regresii efectuate pe serii de timp, permițând informațiilor să fie persistate pe parcursul buclei de dimensiune *unități*. Adițional, folosirea de LSTM evită apariția problemei de învățare în cazul dependențelor pe termen lung. După compilare, acesta a fost exportat folosind greutăți inițializate aleatoriu folosind o distribuție normală.

Text

Description automatically generatedDiagram

Description automatically generated with medium confidence

Modelul a fost antrenat pe un dataset constituit din 500 de înregistrări istorice provenite de la senzor, fiind încărcate ca *Dataframe Pandas* dintr-un fișier CSV, prelucrate analog cu procesul efectuat de către serviciul de Inferență, adăugând coloanele *ora\_cos* și *ora\_sin* și eliminând coloana *ora*, urmând a fi ulterior separate în dataset destinat antrenării și dataset destinat testării pentru a preveni overfitting-ul modelului.

Chart, scatter chart

Description automatically generated

Ulterior, scaler-ul din Sci-Kit Learn *MinMaxScaler* este folosit pentru a plasa valorile pe o scară de la 0 la 1, acesta fiind antrenat pe datele de antrenament și aplicat atât pe set-ul de antrenament cât și cel test. După prelucrarea datelor, modelul exportat anterior a fost instanțiat folosind *load\_model* din Keras iar datele redimensionate corespunzător. Antrenamentul a fost accelerat prin intermediul unității grafice folosind plugin-ul Metal. După mai multe încercări, 35 de epoci cu un split de 10% din date folosite pentru validare a rezultat în cele mai mici valori ale funcției de cost. În urma efectuării antrenamentului, modelul a fost evaluat pe datele de test cu rezultate bune ce sugerează un fit robust. Având greutățile setate și scaler-ul antrenat, acestea au fost exportate pentru uz ulterior.

Text

Description automatically generatedText

Description automatically generated

Serviciul de Inferență folosește atât modelul cât și scaler-ul, pe care le încarcă la pornirea server-ului Flask. Înainte de a efectua predicția, date sunt prelucrate conform mențiunilor anterioare, transformate de către scaler și redimensionate corespunzător. Inferența este apoi efectuată, iar rezultatul acestuia este readus la scară normală prin metoda *inverse\_transform* a scaler-ului. Acest rezultat este convertit în format JSON și întors serviciului ML, care îl întoarce apoi către utilizator.

Pentru a facilita utilizarea acestei soluții, a fost dezvoltată o interfață grafică folosind framework-ul Next.js. Componentele folosite în crearea view-urilor sunt extrase din librăria React Material UI. Next.js a fost folosit pentru gradul înalt de calitate a experienței utilizatorului, pentru performanța deosebit de ridicată pe care o oferă, cât și pentru posibilitățile de optimizare pentru motorul de căutare.

Graphical user interface, application, table

Description automatically generated

Pagina de *dashboard* este configurată să suporte atât un utilizator ne-autentificat, cât și un utilizator autentificat. Un utilizator ne-autentificat se va putea conecta în aplicație selectând funcția de *Login,* fiind astfel redirecționat către platforma Auth0 unde va putea fie crea un cont în cazul în care utilizatorul este nou, fie se va putea loga în cazul unui utilizator existent.

Graphical user interface, application

Description automatically generatedGraphical user interface, text, application, chat or text message

Description automatically generated

Odată ce utilizatorul s-a autentificat, revenind în pagina *dashboard* aceasta va prelua denumirea utilizatorului și o va sugera. De asemenea, vor fi disponibile variantele de vizualizare a profilului și de delogare a utilizatorului.

Pagina de *settings* permite utilizatorului autentificat să își modifice setările cu privire la casă și camere. Apăsând grid-ul *Home* se va deschide grid-ul ce îi va permite utilizatorului să creeze o casă, iar în cazul în care casa este deja creată, aceasta se poate redenumi.

Graphical user interface, application

Description automatically generated

În același fel, apăsând grid-ul *Rooms*, grid-ul ce permite modificarea locuinței se va închide și se va deschide grid-ul ce permite menținerea camerelor. Acesta vine preîncărcat cu camera *Default*, aceasta fiind necesară asocierii unui dispozitiv înainte de a-i alege camera. În continuare însă se pot adăuga camere noi și se pot șterge respectiv modifica(redenumi) camere existente.

Graphical user interface

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Odată introdusă o cameră, aceasta va putea fi vizualizată atât în pagina de setări cât și în dashboard, specificând de asemenea și numele locuinței din care aceasta face parte.

Graphical user interface, website

Description automatically generated

Pagina de *devices* permite utilizatorului să își adauge un device IoT folosind un machine ID, acesta fiind id-ul gateway-ului și codul de împerechere cu acesta. Tot aici se pot redenumi device-urile și se vor afișa informațiile utile precum temperatura și umiditatea actuală ce sunt preluate din schema OLTP, cea medie în urma ultimelor 24 ore, cea maximă și cea minimă, ultimele trei fiind preluate folosind serviciile aflate în aplicația bazei de date depozit, folosindu-se schema acesteia.

Graphical user interface, text, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Odată adăugat un device, în dreapta apar grid-urile cu privire la senzorii conectați la acesta. Pe lângă cererile din depozitul de date și OLTP, este prezentă și valoarea prezisă pentru următoarea oră în ceea ce privește capabilitatea senzorului. Aceasta este oferită prin apelarea endpointului din microserviciul de ML.

Tot in această pagină, apăsând butonul de *Show More,* pentru fiecare senzor se va deschide un grafic cu privire la datele acestuia istorice, prezentând valoarea pe axa X și ora la care aceasta a fost citită pe axa Y.

Chart, line chart, box and whisker chart

Description automatically generated Chart, line chart, box and whisker chart

Description automatically generated

În pagina principală *dashboard*, utilizatorul își poate adăuga dispozitivele împerecheate în lista de dispozitive preferate. Aceasta este o listă persistată local în cache, iar la fiecare golire a acestuia, ea va fi ștearsă. Utilizatorul are opțiunea de a adăuga maximum 6 device-uri la favorite. Apăsând butonul de *Add* se va deschide dialogul în care se poate selecta dispozitivul dorit. În urma procesului de adăugare, acesta va apărea în grid-ul ales, specificând denumirea acestuia.

Graphical user interface, application

Description automatically generated

Graphical user interface, website

Description automatically generated

# Concluzii

In concluzie, proiectarea unei solutii IoT modulare si extensibile este posibila folosind tehnologii disponibile comercial pentru un pret redus, fapt confirmat si prin expansiunea rapida a acestui segment.

Solutia finala este constituita din mai multe noduri IoT reprezentati de placi de dezvoltare ESP8266 avand diversi senzori atasati care transmit date prin intermediul protocolului MQTT care gateway-ului IoT pentru a fi centralizate si trimise in cloud. Gateway-ul IoT este constituit de un Raspberry Pi 4 cu un gigaoctet de RAM ce gazduieste un broker Mosquitto, dzar si o aplicatie NodeJs ce colecteaza datelele publicate pe topicul *data* si apoi le trimite in cloud prin intermediul HTTPS POST. Acest request este autentificat printr-un token purtator obtinut de la furnizarul de identitate Auth0 prin intermediul Client Credentials Flow. Gazduirea broker-ului local a fost aleasa atat din considerente de securitate, nemaifiind necesara securizarea incarcaturii nici la nivel transport, nici la nivel aplicatie, dar si de consum de energie, protoculul MQTT consumand mai putine resurse decat cel HTTP.

Multiple microservicii au fost dezvoltate, gazduite intr-o instanta Oracle Cloud cu arhitectura ARM, fiind expus doar serviciul Cloud Gateway, restul serviciilor de infrastructura, dar si cel OLTP, Data Warehouse sau Machine Learning fiind accesibile doar din interiorul retelei. Serviciile dezvoltate sunt cloud-native, dar si destinate rularii in containere Docker, permitand lansarea pe noi gazde in doar cateva minute. Aceasta solutie include si componenta de machine learning ce permite prezicerea de noi valori pe baza unei serii de timp, inferenta fiind posibila prin apelarea serviciului Flask. Toate cele 3 servicii functionale Spring Boot sunt legate la o baza de date Oracle Express Edition 21c, gazduite de asemenea in Cloud, insa pe alta instanta Oracle cu arhitectura x86-64, fiind singura platforma suportata de catre imagine Docker oficiala.

Accesul la functionalitatile acestei solutii este facilitat de o interfata grafica NextJs servita de asemenea prin intermediului Cloud Gateway, ce permite creerea de locuinte, camere, imperecherea de gateway-uri IoT, vizualiarea de date istorice, afisarea de statistici din serviciul Data Warehouse, dar si predictii oferite de catre serviciul Machine Learning.

Posibila extensie a acestei solutii poate fi reprezentata de adaugare de noi modele de Machine Learning, dar si de dezvoltare a mai multor endpoint-uri in serviciul de Data Warehouse pentru a genera diverse rapoarte si statistici.

# Bibliografie

|  |  |
| --- | --- |
| [1] | Cisco, „Cisco Annual Internet Report (2018–2023) White Paper,” 9 March 2020. [Interactiv]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html. [Accesat 10 03 2022]. |
| [2] | Espressif, „ESP8266 Configuration,” [Interactiv]. Available: https://arduino-esp8266.readthedocs.io/en/latest/ideoptions.html#note-about-platformio. [Accesat 15 03 2022]. |
| [3] | MQTT, „MQTT,” [Interactiv]. Available: https://mqtt.org/. [Accesat 07 04 2022]. |
| [4] | A. Domingus, „Medium,” 02 05 2020. [Interactiv]. Available: https://adriennedomingus.medium.com/distributed-systems-an-introduction-to-publish-subscribe-pub-sub-6bc72812a995. [Accesat 07 04 2022]. |
| [5] | Next.js, „Next.js,” [Interactiv]. Available: https://nextjs.org/docs/getting-started. [Accesat 10 05 2022]. |
| [6] | Spring.io, „Spring,” [Interactiv]. Available: https://spring.io/projects/spring-cloud-security. [Accesat 09 04 2022]. |
| [7] | baeldung, „Baeldung,” [Interactiv]. Available: https://www.baeldung.com/spring-cloud-netflix-eureka. [Accesat 01 04 2022]. |
| [8] | auth0, „auth0,” [Interactiv]. Available: https://auth0.com/docs/. [Accesat 20 04 2022]. |
| [9] | A. Parajuli, „The IoT Projects,” 18 04 2020. [Interactiv]. Available: https://theiotprojects.com/esp8266-dht11-dht22-temperature-humidity-with-local-web-server. [Accesat 18 03 2022]. |
| [10] | Colah, „Understanding LSTM Networks,” 27 08 2015. [Interactiv]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Accesat 27 04 2022]. |
| [11] | Oracle, „Data Warehousing Guide,” [Interactiv]. Available: https://docs.oracle.com/cd/A87860\_01/doc/server.817/a76994/schemas.htm. [Accesat 29 05 2022]. |
| [12] | A. Geron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, O'Reilly. |