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

**FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ**

**BAZE DE DATE SI TEHNOLOGII SOFTWARE**

**Lucrare de Disertație**

**SOLUTIE 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](#_Toc104911705)

[2. Preliminarii 6](#_Toc104911706)

[2.1. Internet of Things 6](#_Toc104911707)

[2.2. Publish/Subscribe 7](#_Toc104911708)

[2.3. MQTT 8](#_Toc104911709)

[2.3.1 Protocolul MQTT 8](#_Toc104911710)

[2.3.2 Protocolul WebSocket 8](#_Toc104911711)

[2.3.2 MQTT Broker 9](#_Toc104911712)

[2.3.2 MQTT Client 9](#_Toc104911713)

[2.4 HTTPS 9](#_Toc104911714)

[2.5. Spring Boot, Spring Security 10](#_Toc104911715)

[2.6. Docker, Containerizare si izolare 10](#_Toc104911716)

[2.7. AI/ML ? 10](#_Toc104911717)

[3. Descrierea soluției 11](#_Toc104911718)

[3.1. Arhitectura soluției 11](#_Toc104911719)

[3.2. Lista de componente 14](#_Toc104911720)

[3.3. Descrierea nodului IoT 16](#_Toc104911721)

[3.5. Descrierea Gateway-ului 17](#_Toc104911722)

[3.6. Microservicii dezvoltate 18](#_Toc104911723)

[3.7. Solutii ML folosite 24](#_Toc104911724)

[3.8. Interfata Grafica 25](#_Toc104911725)

[3. Concluzii 29](#_Toc104911726)

[Bibliografie 30](#_Toc104911727)

# Introducere

Conectivitatea a devenit omniprezenta in viată moderna, transformând monitorizarea si controlul de la distanta 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 creste pana la 50% in 2023 fata de anul 2018 când aceasta înregistra un procent de 34%, ceea ce înseamnă o schimbare semnificativa a cotei de piața datorita popularizării Internet of Things. Aceste obiecte pot lua forma unui senzor de temperatura si umiditate conectat la Internet, o camera de monitorizare a copilului, pana la electrocasnice inteligente precum mașina de spălat, aerul condiționat sau robotul aspirator.

Scopul acestei lucrări este de a proiecta o soluție modulara, extensibila si ușor de folosit pentru o casa inteligenta. In dezvoltarea acesteia, s-au îmbinat atât concepte Internet of Things prin arhitectura distribuita in noduri si gateway, cat si de inteligenta artificiala pentru a procesa datele captate si de a produce diferite perspective pe care o persoana le-ar fi omis. De asemenea, soluția dezvoltata este cloud native, susținând decuplarea diverselelor componente si oferind un sistem definit de reziliența. Datele sunt transmise de către noduri folosind protocolul MQTT către un broker găzduit pe gateway, păstrând transmisia datelor in Intranet. La nivel de gateway, ruleaza si o aplicație dezvoltata in NodeJS ce este abonata la topicul ce conține date de la senzori si le va scoate in Internet prin intermediul HTTPS, asigurând un transport securizat al încărcăturii. Destinația acesteia este un microserviciu web-facing gazduit intr-o instanta cloud, care constituie rolul de dispecer spre diferitele microservicii internet responsabile. Printre ele se afla si microserviciul ce ofera persistenta datelor prin operatiuni de tip CRUD, acestea fiind salvate intr-o baza de date Oracle Express Edition 21c. Alte microservicii notabile sunt cele de Data Warehouse si cel de inteligenta artificiala, decuplând astfel stocarea in rapoarte de procesarea datelor folosind algoritmi de învățare automata.

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

Capitolul 2, concepte de natura tehnica, ce prezinta succint conceptele necesare realizării lucrării precum concepte de baza Internet of Things, o descriere a protocolului MQTT precum si modelul folosit de acesta, Publish-Subscribe, concepte specifice de containerizare si izolare a proceselor. Alte concepte atinse sunt si cele de Spring Boot, Spring Security, o introducere in inteligenta artificiala si o scurta descriere a protocolului HTTPS.

Capitolul 3, descrierea solutiei, prezinta arhitectura conceputa precum si fluxul datelor. De asemenea, se ofera mai multe detalii despre nodurile IoT, gateway, microserviciile dezvoltate, interfata grafica disponibila utilizatorului, cat si algoritmi de inteligenta artificiala folositi.

Capitolul 4, concluzii, include avantajele si dezavantajele acestei solutii, precum si lucruri de avut in vedere pentru dezvoltari viitoare.

# Preliminarii

## 2.1. Internet of Things

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

Intr-un ecosistem IoT vom gasi un dispozitiv smart care capteaza datele si le trimite mai departe catre un IoT gateway, ce are capabilitati de a colecta si analiza local datele sau de a le transmite la randul sau catre o alta aplicatie.

Diagram

Description automatically generated

Figură 1. Arhitectura IoT

Arhitectura IoT este formata din trei nivele de baza:

* Nivelul de perceptie – este primul nivel ce integreaza obiectele fizice: senzorii sau actuatorii. Obiectivul acestuia este de a colecta informatia asociata si de a o transmite catre nivelul superior.
* Nivelul retea (de transmisie) – este nivelul intermediar destinat Gateway-ului pentru a procesa si transmite datele colectate de la nodurile IoT. Acesta face legatura intre senzori si server.
* Nivelul de aplicatie – este ultimul nivel si poate fi reprezentat prin mai multe servere sau un sistem Cloud unde datele pot fi afisate, analizate si agregate.

## 2.2. Publish/Subscribe

Modelul Publish-Subscribe este un mod de comunicare asincron serviciu catre serviciu. Acesta este un design pattern ce ofera o cale de comunicare intre publishers (host) si 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 arhitectura sta un intermediar numit broker de mesaje cunoscut atat de publisher cat si de subscriber. Acesta primeste mesajele publicate si le trimite catre subscriberii ce s-au abonat topicului, pastrand identitatea fiecaruei componente anonima.

Diagram

Description automatically generated

Figură 2. Modelul Publish/Subscribe

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

Avantajele acestui model sunt:

* Decuplarea intre componente – permite izolarea completa a componentelor, facand sistemul usor de mentinut.
* Elasticitatea solutiei – o solutie ce integreaza acest model de mesagerie este elastic deoarece nu depinde de numarul de publisheri sau subscriberi abonati.
* Sporeste receptivitatea - fiind o solutie asincrona, transferul de mesaje nu blocheaza publisherul, iar subscriberul este ocupat doar atunci cand se aboneaza unui topic.

## 2.3. MQTT

### 2.3.1 Protocolul MQTT

MQTT (Message Queuing Telemetry Transport) este un protocol lightweight folosit in arhitecturile de tip publish/subscribe si destinat comunicarii machine to machine in medii cu latime de banda redusa.

Acesta este un protocol de mesagerie in timp real, oferind o trimitere rapida a datelor. Mesajul transmis pe retea contine ca parametri:

* Payload-ul ce este reprezentat de incarcatura de date inregistrate de nod.
* QoS asigura trimiterea mesajului catre subscriber. Un nivel inalt de QoS ofera o siguranta mai mare ca mesajul sa fie trimis, dar si o latenta si o latime de banda mai mare. Nivelele QoS sunt:
* Maximum o data (0), fara confirmare. Nu ofera o garantie a sosirii mesajului.
* Minimum o data (1), cu confirmare.
* Fix o data (2), cu un hasdshake realizat in 4 pasi. Cel mai sigur si cel mai incet.
* Colectie de proprietati
* 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 in comunicarea server-client. Acesta este un protocol bidirectional, stateful, pastrand conexiunea intre client si server deschisa pana cand unul dintre cei doi o va inchide.

Acesta este folosit in special in aplicatiile de tip real time, oferind posibilitatea de a transmite datele incontinuu si mult mai rapid, folosind aceeasi conexiune deja deschisa.

### 2.3.2 MQTT Broker

Serverul de mesaje sau MQTT Broker este responsabil de gestionarea conexiunilor clientilor MQTT , de interceptarea request-urilor publish, subscribe, unsubscibe, cat si de distribuirea mesajelor.

Printre brokerele MQTT existente se afla si Eclipse Mosquitto, un broker open source ce implementeaza versiunile 5.0, 3.1.1 si 3.1 ale protocolului MQTT. Mosquitto este un broker lightweight, fiind dezvoltat in limbajul de programare C. Acesta ofera si o biblioteca in C pentru implementarea clientilor MQTT .

### 2.3.2 MQTT Client

Clientul MQTT este o aplicatie ce implementeaza MQTT peste TCP/IP pentru a trimite si primi mesaje. Clientul MQTT ce trimite mesajele catre broker este publisher, iar cel ce le primeste de la broker este Subscriber. Fiecare client poate fi subscriber, publisher, dar si ambele in acelasi timp.

Eclipse ofera un client Paho MQTT ce se poate conecta la MQTT broker si implementeaza versiunile 5.0, 3.1.1, si 3.1 ale protocolului MQTT.

## 2.4 HTTPS

HTTPS este versiunea sigura a protocolului HTTP, folosind TLS peste acesta.

Handshake-ul ce are loc intre client si server stabileste un secret si tipul de criptare ce va fi folosita, dar nu cripteaza datele, facand protocoalele TLS si SSL independente de nivelul de aplicatie.

HTTPS foloseste criptarea cu cheie publica/privata, folosind cheia publica pentru criptarea datelor si pe cea privata pentru decriptare.

## 2.5. Spring Boot, Spring Security

## 2.6. Docker, Containerizare si izolare

## 2.7. AI/ML ?

# Descrierea soluției

## 3.1. Arhitectura soluției

Scopul acestei lucrări de disertație este de a proiecta o soluție de tip casa inteligenta modulara si extensibila. Arhitectura soluției consta in mai multe noduri IoT de design propriu, gateway IoT si mai multe micro-servicii lansate intr-o instanta cloud de tip IaaS.

Diagram

Description automatically generated

Figură 3. Arhitectura solutiei

Nodurile IoT folosite sunt placi de dezvoltare ESP8266, conectate la reteaua WiFi si au atasati diversi senzori pe interfata seriala. Datele acestora sunt transmise in clar, insa autentificate prin HMAC si transportate prin intermediul protocolului MQTT catre broker-ul gazduit pe gateway-ul IoT. Gateway-ul folosit este un Raspberry Pi 4 cu 1GB RAM ruland sistemul de operare Raspberry Pi Lite.

La nivel de gateway, a fost configurat broker-ul Eclipse Mosquitto care asculta pe portul TCP 1883 noi cereri de conectare. Broker-ul va distribui mesajele publicate catre abonatii acestora, avand de asemenea posibilitatea pastrarii ultimului mesaj pentru a fi distribuit noilor abonati, cat si ajustarea serviciului calitatii, iar autentificarea clienților se face pe baza de nume de utilizator si parola.

In componenta solutiei se afla de asemenea si o aplicatie NodeJS ce deserveste doua scopuri, client MQTT si client HTTPS. Aceasta se aboneaza la broker pentru diverse topicuri la care sunt publicate datele senzorilor, urmand a fi verificat HMAC-ul si ulterior impinse catre cloud prin intermediul HTTP POST. Pentru a rezuma, gateway-ul IoT va concentra toate datele primite in Intranet de la diversele noduri, urmand a le impinge in cloud pentru stocare si 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.

In cloud, datele sunt trecute printr-un microserviciu web facing, gateway-ul,urmand a fi trimise mai departe catre microservicii private. Aceste microservicii sunt gazduite intr-o instanta Oracle Cloud Instance si rulate in containere Docker pentru a facilita punerea in functiune a aplicatiei, cat si securitatea prin izolarea inherenta oferita de containerizare. De asemenea, modul Swarm este activat, ce determina un grad sporit de rezilienta la diverse probleme ce pot aparea pe durata executiei, cat si un raspuns la cerere foarte rapid prin scalabilitatea oferita de catre acest mod. Pentru dezvoltarea microserviciilor private a fost folosit atat Spring Boot si Java, pentru serviciul OLTP si pentru cel de tip Data Warehouse, cat si Flask si Python pentru dezolvtarea microserviciului responabil cu operatiile de tip Machine Learning. Pentru a facilita utilizarea acestei solutii, o interfacata grafica dezvoltata in Next.js folosind componente din populara librarie de React Material UI a fost adaugata. Framework-ul acesta a fost folosit pentru performanta deosebit de ridicata pe care o ofera, cat si pentru posibilitatile de optimizare pentru motorul de cautare.

Toate cererile HTTP sunt adresate gateway-ului, care pe langa rolul principal de rutare mai are si rol de autentificare si autorizare. Cererile catre interfata grafica sunt permise si utilizator ne-autentificati, iar pentru utilizarea microserviicilor autentificarea este necesare. Autentificarea si autorizarea utilizatorilor este facilitatea de utilizarea furnizorului de identitate Auth0, si este realiza folosind JSON Web Tokens, permitatand decuplarea starii de autentificare de catre server-ul unde a fost efectuata.

Aceasta arhitectura a fost aleasa pentru flexibilitatea si extensibilitatea pe care o ofera, avand de asemenea un grad sporit de securitate. Toate nodurile vor publica in acelasi topic, având necesara zero configurare pentru introducerea de noi senzori in retea. Pentru a putea accesa datele de la noduri, este necesara asocierea gateway-ului unui cont. Acest proces este facilitat de catre interfata grafica dezvoltata in Next.js, iar de indata ce imperecherea a fost încheiata cu succes, datele vor fi impinse in cloud catre gateway ce va delega rezolvarea cerereii catre microserviciul OLTP dedicat stocarii informatiei intr-o baza de date relationala Oracle Express Edition 21c ce efectuaza operatii de tip CRUD. Acest pas va crea un strat de persistenta si faciliteaza accesul la date intr-un format usor de manipulat. De asemenea, acest microserviciu este folosit si in cadrul interfetei grafice pentru a popula date semnificative despre valorile actuale ale senzorilor, precum si date salvate de catre utilizator despre locuinta sa si amplasamentul senzorilor per camera.

Pe langa microserviciul OLTP, mai a fost implementat serviciul ce permite accesarea datelor structurate in modul Data Warehouse ce sunt stocate intr-o baza de date Oracle Express 21c si servite prin intemediul protocolului HTTPS, dar si serviciul responsabil cu efectuarea operatiilor Machine Learning folosind un model pre-antrenat.

De asemenea, aceasta solutie este usor scalabila si fiabila, fiind conceputa cloud-native si fault-tolerant. Modul Docker Swarm permite provizionarea rapida de noi containere in cazul in care procesul initial este ucis de catre o eroare, dar si cresterea sau scaderea numarul de replici pentru a ajusta corespunzator nivelului de incarcare.

## 3.2. Lista de componente

Tabel 1. Lista de componente

|  |  |  |  |
| --- | --- | --- | --- |
| Componenta | Pret(lei) | Cantitate(buc) | Pret Total(lei) |
| Placa de dezvoltare ESP8266 | 36.99 | 2 | 73.98 |
| Senzor de temperatura si 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 capabilitati WiFi produs de catre compania chineza Espressif. Acesta este definit de o arhitectura pe 32 de biti, avand un procesor Tensilica Diamond Standard 106Micro tactat in mod implicit la 80MHz, insa care poate fi setat sa ruleze si la o frecventa de 160Mhz. Ca si memorie volatila, prezinta 32KB rezervati pentru instructiuni, si 80KB disponibili pentru rularea de programe de catre utilizator.

Un factor important in alegerea unei placi de dezvoltare cu acest chip a fost suportul pentru stiva TCP/IP precum si suportul pentru functii criptografice cu o intensitate scazuta, pentru functiile de rezumat si calculul de coduri de autentificare a mesajelor.

Doua placi de acest tip au fost folosite pentru realizarea lucrarii, ele avand rolul de a transmite date de la senzori catre gateway.

1. Senzor de temperatura si umiditate DHT11

DTH11 este un senzor de temperatura si umiditate ce produce un semnal digital pe pin-ul de date. Temperatura este masurata folosind un termistor cu coeficient negativ iar umiditatea relativa este masurata folosind un senzor capacitiv. Senzorul poate citi valori ale humiditatii in plaja 20-90% RH cu o acurateta la masurare de +/- 5% RH. Temperatura poate lua o valoare intre 0 si 60◦C cu o acuratete de +/-2◦C. Senzorul permite un voltaj de intrare in plaja 3.3V si 5V.

1. Raspberry Pi 4 1GB RAM

Raspberry Pi 4 este o placa de dezvoltare ce contine un procesor actualizat fata de generitiile precedente, avand o arhitectura pe 64 de biti si un procesor cu 4 nuclee tactate la 1.5GHz ce are incorporat un disipator de caldura din metal. De asemenea, pe partea de conectivitate acesta are 3 port-uri USB 3.0, placa de retea Wi-Fi ce functioneaza atat in modul 2.4GHz cat si 5GHz, Bluetooth 5.0 Low Energy, Gigabit Ethernet cu capabilitiati de Power over Ethernet, dar si porturi Micro HDMI ce suporta rezolutii pana la 4K.

Versiunea folosita pentru realizarea proiectului este dotata cu 1GB de memorie volatila si ruleaza sistemul de operare Raspberry OS Lite. Aceasta placa de dezvoltare este folosita pe post de gateway, sustinand infrastructura necesara conectarii senzorilor si transmiterea de date atat de la senzor la gateway cat si de la gateway in cloud. Concret, gateway-ul gazduieste un broker Mosquitto MQTT cat si aplicatie ce colecteaza date publicate si le transmite securizat folosind HTTPS catre Cloud.

1. Card MicroSD 32GB

Card-ul MicroSD este folosit pentru a dota Raspberry Pi-ul cu memorie non-volatila, alengadu-se capacitatea de 32GB pentru a permite instalarea sistemului de operare cat si a tuturor aplicatiilor necesare.

Conectarea fizica a senzorilor cu placa de dezvoltare a fost efectuata conform tabelului de mai jos.

Tabel 2. MAPPAREA!?!?!

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

Diagram, schematic

Description automatically generated

Figură 5. Schema electrica

Pentru stratul de perceptie al nodului, senzorul de umiditate si temperatura DHT11 a fost conectat la placa de dezvoltare ESP8266. Aceasta placa de dezvoltare este dotata 17 pini GPIO, avand suport pentru interfetele SPI, I2C, UART dar si Analog-to-Digital pe 10 biti. Alimentarea senzorului a fost efectuata folosind pinul de 3V prezent pe placa, voltaj permis conform fisei tehnice a acestuia. Pinul de date a fost ulterior conectat direct catre pinul GPIO5, marcat pe placa ca si D1, iar impamantarea a fost legata de pinul G.

## 3.3. Descrierea nodului IoT

Nodurile IoT sunt consistuite din placi de dezvoltare ESP8266 avand conectati pe interfata seriala diversi senzori. Un exemplu de acest fel ar fi DHT11, senzor de umiditate(%) si temperatura(◦C). Acestea comunica cu gateway-ul prin intermediul protocolului la nivel de applicatie MQTT, iar la nivel de transport TCP.

Acestea sunt conectate la rețeaua Wi-Fi de tip intranet, ele nefiind menite sa se conecteze in internet din considerente de securitate. Datele receptionate de la senzori sunt trimise prin intermediul protocolului MQTT. Concret, nodurile publica pe acelasi topic, fiind clienti MQTT. Plasarea nodurilor in intranet a fost aleasa strategic, permitand economisarea bateriei si limitarea consumului de resurse prin renuntarea implementarii securitatii la nivel de transport prin TLS, bazandu-se strict pe confidentialitatea oferita de catre WPA3-PSK la nivelul legaturii de date. Autentificarea nodurilor la broker se face pe baza unor credentiale arse in EEPROM, iar autentificarea mesajelor se face pe baza HMAC. Cheia simetrica este similar stocata in EEPROM. De indata ce nodurile sunt conectate la WiFi, acestea vor incepe sa trimita date catre broker, nefiind necesara configurare aditionala.

Dezvoltarea componentei software aferente nodului IoT a fost facuta in framework-ul Arduino, ce permite folosirea limbajului C++ si o paradigma orientata obiect. De asemenea, conform sugestiei din documentatie [2], abstractizarea sistemului de asamblare a binarului si de gestionare a dependintelor a fost realizata cu ajutorului extensiei de Visual Studio Code numita PlatformIO. Aceasta extensie permite de asemenea si un sistem avansat de completare si sugestie a codului, precum si suport pentru depanare si o portabilitate a codului ridicata. Compilatorul folosit este o versiune modificata a GCC (GNU Compiler Collection), ce suport standardul C++ 11 si un subset al librariei *std* pe langa librariile specifie framework-ului Arduino si dar si librarii destinate accesarii functiilor prezente pe ESP8266, precum Wi-Fi sau posibilitatea de a restarta nodul programatic.

## 3.5. Descrierea Gateway-ului

Acesta determina strategia de trimitere a mesajelor, putand lua una dintre urmatoarele valori:

* 0, modul „fire and forget”, ce garanteaza trimiterea mesajului o singura data, fara a mai fi necesara confirmarea primirii
* 1, modul „at least once”, ce garanteaza trimiterea mesajului cel putin o data, fiind necesara confirmarea
* 2, modul „exactly once”, ce garanteaza trimiterea mesajului exact o singura data, fiind necesara confirmarea si lipsa duplicatelor

Pastrarea ultimului mesaj cu scopul distribuirii catre noii abonati este de asemenea importanta pentru topicurile actualizate rar.

## 3.6. Microservicii dezvoltate

Parte integranta a solutiei este constituita de microserviciilor dezvoltate folosind fie framework-ul Spring Boot in Java, fie Flask in Python, ce sunt gazuite pe o instanta de cloud Oracle. Architectura aceasta a fost folosita in detrimentul monolitului din primsa beneficiilor multiple cu sunt oferite pe parte de distribuire si mententata, in ciuda efortului de dezvoltare asociat.

Functional, acestea realizeaza fie rol de aplicatie OLTP, asociata unei baze de date relationala Oracle Express Edition 21c, fie rol de aplicatie Data Warehouse imperecheata cu o aceeasi baza de date mentionata anterior, insa folosind alta schema, dar si o aplicatie specializata pentru machine learning. Toate cererile sunt securizate prin intermediul protocolului HTTPS, si autenficate folosind JSON Web Tokens emise si verificate de catre autoritatea emitenta Auth0.

Diagram

Description automatically generated

Figură 6. Diagrama microserviciilor

Tehnic, toate aceste cereri sunt trecute printr-un gateway care prima data autentifica in functie de calea folosita si apoi ruteaza catre microserviciul responsabil. Partea de autentificare este facilitata de folosirea starter-ului Spring Cloud Security dar si al starter-ului ce verifica transparent JSON Web Token-urile primite in campul Authorization al cererii, numit OAuth 2.0 Resource Server. Transparenta aceasta este obtinuta prin folosirea datelor din fisierul de configurare pentru a injecta un Bean ce foloseste URL-ul autoritatii emitente pentru a verifica validitatea token-ului. Pentru a modifica setarile implicite care obliga autentificarea fiecarui request, a fost creat un *WebFilter* ce permite acces neautorizat catre interfata grafica pentru a permite utizilatorilor sa-si creeze cont. Creearea de useri, emiterea si verificarea de token-uri sunt delegate catre Auth0, permitand simplificarea architecturii de securitate.

Text

Description automatically generated

Figură 7. Setări de securitate aferente Gateway-ului

Partea de rutare este implementata folosind Spring Cloud Gateway, insa si alte microservicii destinate infrastructurii au trebuit sa fie adaugate. Printre acestea se innumara un microserviciu de Service Discovery, dar si un microserviciu destinat servirii de configuratii dintr-un repository de GitHub.

Microserviciul destinat configuratiilor este dezvoltat tot in Spring Boot, folosindu-se de starter-ul Spring Cloud Config Server. Acesta foloseste fisierul de configurare pentru a-si extrage informatiile necesare crearii unui server ce incarca in timp real din GitHub fisiere YAML ce urmeaza a fi servite catre alte microservicii. Pentru rezolvarea fisierului corect, formatul *nume-profil.yml* trebuie sa fie folosit, exemplu fiind *gateway-prod.yml*, unde *prod* este numele profilului. Pentru a evita eventualele probleme asociate pornirii multiplelor servicii, portul 8800 a fost folosit. Concret, fisierele de configurare pentru fiecare microserviciu au fost incarcare intr-un repository privat, iar credentialele de autentificare au fost plasate in fisierul *application-prod.yml* ca si nume de utilizator si token de access generat prin intermediul setarilor destinate dezvoltatorilor din cadrul interfetei web.

Rezolvarea serviciilor la momentul rularii este realizata prin intermediul microserviciului de Service Discovery, dezvoltat de asemenea in Spring Boot si folosind starter-ul Netflix Eureka Server. Avand in vedere existenta server-ului de configurari, aceasta este incarcata la momentul intializarii, declarand in fisierul impachet cu microserviciul doar numele aplicatiei si URL-ul serverului. In cadrul fisierului servit din repository-ul de GitHub, portul a fost setat sa fie 8761, iar clientul de Eureka configurat pentru a evita inregistrarea serverului.

Text

Description automatically generated

Figură 8. Configurarea microserviciului de Service Discovery

Un alt aspect notabil apartinand infrastructurii o constituie folosirea de Zipkin, un sistem distribuit de tracing ce permite o mult mai usoare vizualizare a cererilor, precum si depanarea problemelor de latenta intalnite.

Revenind la rutarea cererilor de catre gateway spre celelalte microservicii, aceasta este facuta declarativ prin intermediul fisierului de configurare.

Text

Description automatically generatedText

Description automatically generated

Figură 9. Configurarea Gateway-ului

Libraria Spring Cloud Gateway a fost aleasa in detrimentul altor solutii similare datorite integrarii puternice pe care o ofera cu server-ul de Service Discovery Eureka, permitand rezolvarea serviciului cu cea mai mica incarcatura pentru a servi cererea, dar si integrearea cu librarii ce aplica design pattern-ul Circuit Breaker. Acest pattern vine in completarea celui de Retry si este util in cazurile in care erorile apar din cauza unor evenimente ce nu pot fi anticipate, permitand folosirea unei metode de fallback in caz de circuit „deschis”. Libraria Resilience4j a fost aleasa pentru a indeplini aceasta sarcina, specificand un circuit breaker pentru fiecare cale disponibila. Ca si client de Service Discovery, starter-ul Netflix Eureka Client a fost folosit, configurarea acestuia facandu-se prin intermediul fisierului YAML, specificand URL-ul serverului, in speta microserviciul de Service Discovery, dar si faptul ca acest client n-ar trebui sa se inregistreze server-ului. Pentru o mai rapida rutare a cererilor catre serviciul destinatie, URL-urile rezolvate de Eureka sunt pastrate intr-un cache manageriat prin abstractia specifica Spring oferind implementarea concreta Caffeine injectata in context prin intermediul unui Bean.

Text

Description automatically generated

Figură 10. Configurarea cache-ului din Gateway

Microserviciul OLTP este adresabil folosind prefixul /*iot*, iar scopul acestuia este de a persista datele aferente senzorilor dar si despre locuinta utilizatorului si incaperile acesteia. Serviciul este dezvoltat folosind starter-ul Web, ce permite conventie asupra configurarii prin folosirea de annotari precum *@RestController* sau *@Service*. Acesta suporta de asemenea si tracing prin configurarea Zipkin, dar si rezolvarea fisierelor de configurare la momentul initializarii prin Spring Cloud Config. Acesta se inregistreaza la registrul de aplicatii prin clientul Eureka pentru a putea fi identificat de catre Gateway. De asemenea, toate cererile sunt autentificate si autorizate prin folosirea de JSON Web Tokens, folosind acelasi starter OAuth 2.0 Resource Server, autoritatea emitenta fiind tot Auth0. Pentru imbunatariea performantei, startul de abstractizare a cache-ul oferit de catre Spring Boot a fost folosit in conjunctie cu implementarea Caffeine, iar pentru imbunatariea rezilientei la erori a solutiei a fost activat management-ul tranzactiilor.

! INSERARE POZA CONFIGURARE

Acesta se foloseste de o baza de date Oracle Express Edition 21c pentru a persista datele, folosind driver-ul de JDBC thin. Schema bazei de date a fost generata folosind Hibernate, setand proprietatile aferente in fisierul de configurare a profilului *dev*, urmand a fi creat un fisier SQL la pornirea serviciului.

Text

Description automatically generated with low confidenceINSERARE POZA CU SCHEMA IN DR

Spring Data JPA a fost folosit pentru a creea stratul raspuzator accesului de date, reducand semnificativ codul boiler-plate si imbunatadind viteza de dezvoltare. Entitatile sunt reprezentate de clase annotate cu *@Entity* iar membrii acestora pot reprezenta atat campuri, dar si chei primare sau de legatura.Accesul la date este facilitat prin folosirea de repository-uri reprezentate de interfete ce extind tipuri definite de Spring precum *CrudRepository* sau *PagingAndSortingRepository*, dar si prin declararea de metode folosind limbajul specific domeniului ce vor fi generate la momentul compilarii 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 gasite si injectate de catre Spring Boot in context, ulterior fiind expuse printr-un container Tomcat. Au fost create controllere pentru toate entitatile, aplicandu-se conceptul *Separation of Concerns* structurand proiectul in in controllere, service-uri si repository-uri, imbunatatind astfel lizibilitatea proiectul si usurand viitoarea mentenanta. De asemenea, serviciul n-are nicio dependinta exceptand cele de infrastructura, fiind astfel usor de distribuit si scalat in functie de incarcatura.

Diagram

Description automatically generated

Fluxul de folosire a acestui serviciu implica creerea de tari si orase de catre administratorul aplicatiei, urmand ca utilizatorul sa-si creeze o casa inainte de orice actiune de imperechere cu un gateway. Identificarea acestuia se face prin token-ul purtator, urmand sa aiba valabila posibilitate de salvare a unei case, regasire a detaliilor acesteia dar si editare a numelui. Pentru a asocia o casa unui user se foloseste campul *sub* din JWT, iar la salvarea casei se creeaza si o camera implicita unde urmeaza a fi salvati senzorii la momentul imperecherii.

Controller-ul IotGateway are o metoda *signal* destinata a fi apelata de catre dispozitivul IoT pentru a semnala disponibilitatea acestuia, trimitand in corpul cererii detalii precum codul de imperechere, tipul acestuia dar si senzorii instalati. Nodul este identificat prin intermediul campului *sub* al token-ului trimis, iar daca nodul este deja imperechat, este intors ID-ul utilizatorului; altfel, detaliile acestea sunt salvate intr-o mapa concurenta, marcand astfel disponibilitatea dispozitivului de a fi imperechat, urmand sa cheme aceasta metoda in bucla pana va fi primii ID-ul unui utilizator. Endpoint-ul *pair* este destinat de a fi apelat de catre utilizator, trimitand codul masinii si codul de imperechere pentru a-si insusi dispozitivul, fiind astfel scos din pool-ul dispozitivelor pregatite pentru imperechere. In urma acestor operatiuni, utilizatorul poate edita numele gateway-ului, sterge un dispozitiv imperecheat, dar si intoarcerea tuturor gateway-urilor imperecheate. Un gateway are unul sau mai multe senzori, editabili folosind endpoint-ul *sensor,* acestia putand fi mutati in alte camere definite de catre utilizator prin intermediul endpoint-ul *room*. In privinta endpoint-ului *record*, acesta permite salvarea de noi intrari de catre dispozitivul IoT, acesta fiind identificat prin intermediul token-ului purtator, dar si intoarcerea de inregistrari catre utilizator in regim paginat, identificarea fiind analog facuta ca si la endpoint-ul precedent.

Microserviciul de Data Warehouse este introdus in proiect pentru a intruni nevoia de stocare a volumului mare de date istorice asupra carora se pot dezvolta rapoarte si analize. Acesta foloseste o instanta de baza de date Oracle Express Edition 21c, unde s-au agregat datele relevante din baza de date OLTP.

Diagram

Description automatically generated

Pentru modelarea depozitului de date s-a folosit proiectarea schemei sub forma de stea, avand in vedere performanta pe care o aduce stocarea intregii informatii intr-o singura inregistrare.

Centrul aesteia consta intr-o tabela de fapte, *Record,* aceasta fiind subiectul principal al aplicatiei si sursa de generare a diferitelor analize. Tabelele dimensiune ce stocheaza informatia aditionala tabelului de fapte sunt: *Sensor,* *Address*, *Home*, *Time*.

DE INSERAT SCHEMA!!!

Toate tabelele dimensiune au fost rezultate din operatii de JOIN, denormalizand schema OLTP astfel:

Tabela *Sensor* stocheaza datele cu privire la senzor *id, name, description*, dar regaseste si informatiile Gateway-ului cu care este asociat. Asadar acesta are 3 atribute suplimentare *machine\_id, gateway\_friendly\_name, type,* aduse prin relatia dintre *Sensor* si *Gateway* din OLTP.

Tabela *Address* mentine informatii cu privire la oras si tara, avand atributele *id, city, country*, fiind o asociere intre tabelele *City* si *Country* din OLTP.

Tabela *Home* stocheaza date cu privire la locuinta, incluzand aici atat home\_name adus din tabela Home, cat si room\_name adus din tabela Room din OLTP.

Tabela dimensiune *Time* este adusa in aceasta schema pentru a descrie temporal informatia, mentinand ca atribute *reading\_id, full\_date, year, month, day, quarter*.

Aplicatia Data Warehouse este creata folosind framework-ul Spring Boot, utilizand pentru partea de acces la baza de date JDBC Template pentru a avea mai multa flexibilitate in vederea crearii cererilor de informatii ce stau la baza generarii rapoartelor.

Astfel, aplicatia contine cereri pentru 2 rapoarte: cel ce aduce temperatura maxima din ziua curenta si cel ce aduce umiditatea medie din ziua curenta, ambele cereri permitand selectarea informatiei dintr-o locuinta aleasa de utilizator, primita ca parametru in path-ul requestului.

Text

Description automatically generated

Datele din Data Warehouse sunt colectate din OLTP folosind proceduri stocate. Procedura migrate\_*data()* insereaza valorile aferente pentru a popula tabelele *Address*, *Home* si *Sensor*. Procedura insert\_*time*(*)* insereaza valorile coloanelor din tabela Time. Procedura insert\_*records()* insereaza datele din OLTP pentru tabela *Record*, aplicand multiple operatii de JOIN pentru a colecta toate atributele necesare.

Procedurile stocate sunt chemate in metodele din repository-ul *RecordRepository*, care sunt la randul lor apelate in metodele din *RecordService*, iar mai apoi in cele din *RecordController*, mapand fiecare apel de raport intr-un serviciu de tip *GET* si fiecare apel de inserare intr-un serviciu de tip *PUT*.

Text

Description automatically generated

Pe langa serviciile de OLTP si Data Warehouse, mai sunt prezente si cele de ML si inferenta care sunt intr-o stransa legatura.

Serviciul ML este dezvoltat folosind Spring Boot si este disponibil la calea *nn*, fiind de asemenea si client de Eureka ce ii permite Cloud Gateway-ului sa 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 autorizarii si autentificarii cererilor, rezolvarea proprietatilor la run-time prin Spring Cloud Config, folosirea stratului de abstractie a cache-ului impreuna cu implementarea Caffeine pentru a imbunatati performanta, dar si includerea Spring Data pentru a facilita accesul la baza de date. Persistenta este oferita de o instanta de Oracle Express Edition 21c, partajata cu seviciul Data Warehouse pentru a a accesa datele formatate conform diagramei stea. Serviciul expune un singur endpoint *predict*, primind ca variabila a caii ID-ul senzorului pentru care sa fie facuta inferenta. La nivelul serviciului, acesta verifica existenta unui senzor cu acel ID, apoi selecteaza din baza de date inregistrari aferente senzorului pe ultimele 24 de ore. Acestea sunt apoi formatate ca matrice cu doua coloane, ora si valoarea, si trimise catre serviciul de Inferenta, prin intermediul unui client Feign. Acest client permite un stil declarativ in folosirea de client REST, abstractizand detaliile ce tin de implementare.

Text

Description automatically generated

Serviciul de Inferenta este dezvoltat in Python folosind framework-ul Flask pentru a creea un server HTTPS, iar partea de machine learning este construinta folosind Tensorflow, Keras si Sci-Kit Learn. Aceste serviciu nu necesita autentificare, fiind disponibil doar in Intranet si apelat de catre serviciul ML, ne-existand ruta a Gateway-ului care sa aiba legatura directa cu acest microserviciu. Exista un singur endpoint *predict*, apelat folosind metoda HTTP POST, avand ca incarcatura 24 inregistrari reprezentand valori din trecut, iar ca raspuns va trimite rezultatul inferentei folosind un model si un scaler pre-antrenat, la care sunt folosite date prelucrate. Din matricea originala ce este caracterizata de doua coloane, *ora* si *valoare*, mai sunt introduse coloanele *ora\_cos*, cosinusului orei ajustata, si *ora\_sin*, rezultatul functiei sinus aplicata orei ajustate, urmand a fi indepartata coloana *ora*.

Modelul si scaler-ul au fost pre-antrenate si exportate de catre alta aplicatie Python, fiind definit un model secvential Keras ce reprezinta o retea neuronala recurenta de tipul Long Short Term Memory (LSTM) cu 40 de unitati si optimizata folosind Adam bazata pe metrica de cost Mean Squared Error (MSE). Reteaua neuronala recurenta este indicata pentru regresii efectuate pe serii de timp, permitand informitatilor sa fie persistate pe parcursul buclei de dimensiune *unitati*. Aditional, folosirea de LSTM evita aparita problemei de invatare in cazul dependintelor pe termen lung. Dupa compilare, acesta a fost exportat folosind greutati intializate aleatoriu folosind o distributie normala.

Text

Description automatically generatedDiagram

Description automatically generated with medium confidence

Modelul a fost antrenat pe un dataset constituit din 500 de inregistrari istorice provenite de la senzor, fiind incarcate ca *Dataframe Pandas* dintr-un fisier CSV, prelucrate analog cu procesul efectuat de catre serviciul de Inferenta, adaugand coloanele *ora\_cos* si *ora\_sin* si eliminand coloana *ora*, urmand a fi ulterior separate in dataset destinat antrenarii si dataset destinat testarii pentru a prevenii oferfitting-ul modelului.

Chart, scatter chart

Description automatically generated

Ulterior, scaler-ul din Sci-Kit Learn *MinMaxScaler* este folosit pentru a plasa valorile pe o scara de la 0 la 1, acesta fiind antrenat pe datele de antrenament si aplicat atat pe set-ul de antrenament cat si cel test. Dupa prelucrarea datelor, modelul exportat anterior a fost instantiat folosit *load\_model* din Keras iar datele redimensionate corespunzator. Antrenamentul a fost accelerat prin intermediul unitatii grafice folosind plugin-ul Metal. Dupa mai multe incercari, 35 de epoci cu un split de 10% din date folosite pentru validare a rezultat in cele mai mici valori ale functiei de cost. In urma efectuarii antrenamentului, modelul a fost evaluat pe datele de test cu rezultate bune ce sugereaza un fit robust. Avand greutatile setate si scaler-ul antrenat, acestea au fost exportate pentru uz ulterior.

Text

Description automatically generatedText

Description automatically generated

Serviciul de Inferenta foloseste atat model-ul cat si scaler-ul, pe care le incarca la pornirea server-ului Flask. Inainte de a efectua predictia, date sunt prelucrate conform mentiunilor anterioare, transformate de catre scaler si redimensionate corespunzatoar. Inferenta este api efectuata, iar rezultatul acestuia este readus la scara normala prin metoda *inverse\_transform* a scaler-ului. Acest rezultat este convertit in format JSON si intors serviciului ML, care il intoarce apoi catre utilizator.

Pentru a facilita utilizarea acestei solutii, a fost dezvoltata o interfata grafica folosind framework-ul Next.js. Componentele folosite in crearea view-urilor sunt extrase din libraria React Material UI. Next.js a fost folosit pentru gradul inalt de calitate a experientei utilizatorului, pentru performanta deosebit de ridicata pe care o ofera, cat si pentru posibilitatile de optimizare pentru motorul de cautare.

Graphical user interface, application, table

Description automatically generated

Pagina de *dashboard* este configurata sa suporte atat un utilizator ne-autentificat, cat si un utilizator autentificat. Un utilizator ne-autentificat se va putea conecta in aplicatie selectand functia de *Login,* fiind astfel redirectionat catre platforma Auth0 unde va putea fie crea un cont in cazul in care utilizatorul este nou, fie se va putea loga in cazul unui utilizator existent.

Graphical user interface, application

Description automatically generated

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

Graphical user interface, text, application, chat or text message

Description automatically generated

Pagina de *settings* permite utilizatorului autentificat sa isi modifice setarile cu privire la casa si camere. Apasand grid-ul *Home* se va deschide grid-ul ce ii va permite utilizatorului sa creeze o casa, iar in cazul in care casa este deja creata, aceasta se poate redenumi.

Graphical user interface, application

Description automatically generated

In acelasi fel, apasand grid-ul *Rooms*, grid-ul ce permite modificarea locuintei se va inchide si se va deschide grid-ul ce permite mentinerea camerelor. Acesta vine preincarcat cu camera *Default*, aceasta fiind necesara asocierii unui dispozitiv inainte de a-i alege camera. In continuare insa se pot adauga camere noisi se pot sterge respectiv modifica(redenumi) camere existente.

Graphical user interface

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Odata introdusa o camera aceasta va putea fi vizualizata atat in pagina de setari cat si in dashboard, specificand de asemenea si numele locuintei din care aceasta face parte.

Graphical user interface, website

Description automatically generated

Pagina de *devices* permite utilizatorului sa isi adauge un device IoT folosind un machine ID si codul de imperechere. Tot aici se pot redenumi device-urile si se vor afisa informatiile utile precum temperatura si umiditatea actuala, cea medie in urma ultimelor 24 ore, cea maxima si cea minima, informatii ce sunt extrase din baza de date depozit.

Graphical user interface, text, application

Description automatically generated

!!! DE FACUT SS CU RECORDURI LA DEVICES

In pagina principala *dashboard*, utilizatorul isi poate adauga dispozitivele imperecheate in lista de dispozitive preferate. Aceasta este o lista persistata local in cache, iar la fiecare golire a acestuia, ea va fi stearsa. Utilizatorul are optiunea de a adauga maximum 6 device-uri la favorite. Apasand butonul de *Add* se va deschide dialogul in care se poate selecta dispozitivul dorit. In urma procesului de adaugare, acesta va aparea in grid-ul ales, specificand denumirea acestuia.

Graphical user interface, application

Description automatically generated

Ramas de adaugat: ss cu recorduri la devices (toata pagina), rapoarte + ss, help page ss,favorite devices ss incarcate corect

# Concluzii

# Bibliografie