Energy Management System

Documentatie

Student: Grumazescu Silviu

Assignment 1

1. Introducere

Tema acestui proiect reprezinta dezvoltarea unui sistem de gestionare a energiei consumate de dispozitivele a mai multor utilizatori, la baza avand doua microservicii si un frontend pentru interactiunea cu utilizatorul.

2. Cerinte functionale

- Sistem de autentificare bazat pe roluri (Client/Admin)
- Administratorul poate face operatii CRUD pe clienti si pe dispozitive
- Administratorul poate asigna un dispozitiv unui client
- Un client isi poate vedea dispozitivele asignate
- Clientului nu ii va fi permis sa acceseze pagini designate altor roluri.

3. Cerinte non-functionale

- Cerintele functionale sunt implementate intr-o arhitectura bazata pe doua microservicii: microserviciu pentru gestionarea clientilor si unul pentru gestionarea dispozitivelor
- Securitate: Clientii sunt restrictionati din a accesa paginile adminului

4. Arhitectura sistemului distribuit

Functionalitatea aplicatiei a fost impartita in doua microservicii, ce au ca backend o aplicatie Java Spring Boot conectata la o baza de date PostgreSQL.

Usermanagement Microservice

Acest microserviciu are ca functionalitate operatiile CRUD pe utilizatori si gestionarea autentificarii clientilor. Autentificarea se realizeaza pe baza unui token. Odata validata parola si username-ul utilizatorul va primi un token ce va fi atasat la toate restul request-urilor facute.

Devicemanagement Microservice

Acest microserviciu are ca functionalitate operatiile CRUD pe dispozitive si maparea utilizatorilor la dispozitive. Deoarece si acest microserviciu are nevoie de informatii despre utilizatori, o copie minimala din baza de date a utilizatorilor va fi mentinuta si in cea a dispozitivelor. Pentru a mentine datele actuale, la fiecare operatie CRUD in microserviciul pentru useri, acesta trimite un request la celalalt microserviciu cu datele modificate.

Frontend

Partea de interfata cu utilizatorul este dezvoltata folosind framework-ul Vue 3. Frontend-ul trimite request-uri HTTP celor doua microservicii, acestea permitand sau respingand cererile in functie de rolul utilizatorului.

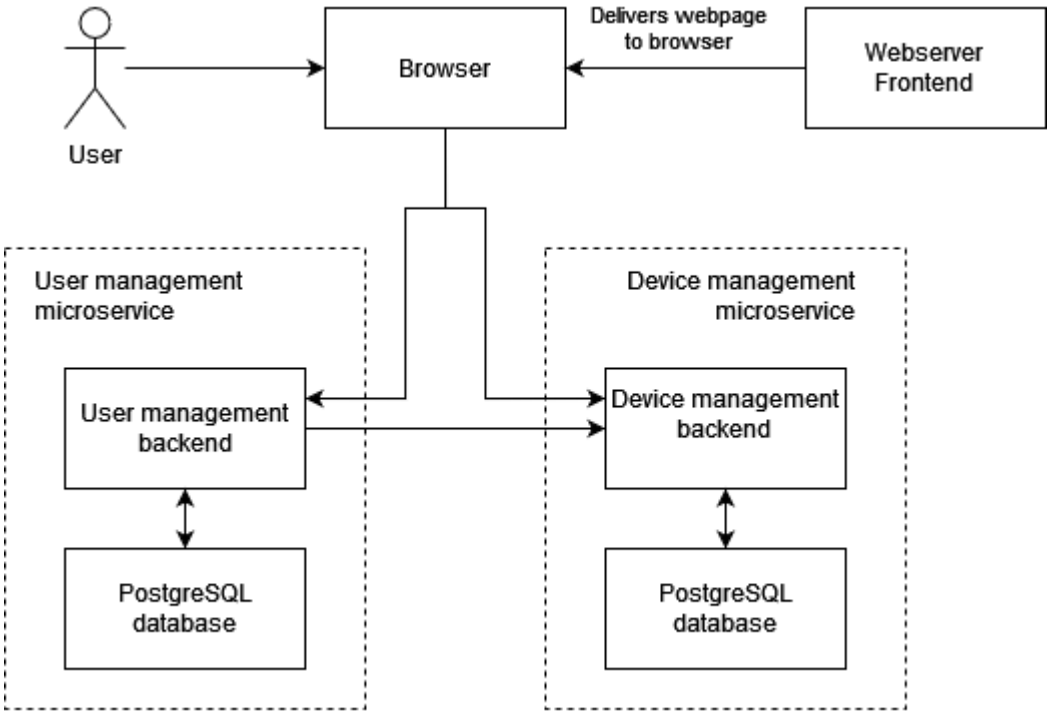
Baze de date

Aplicatia are doua baze de date, cate una pentru fiecare microserviciu. Intr-o baza de date vom stoca dispozitivele si in cealalta utilizatorii.

Pentru a putea face o legatura intre dispozitiv si device, avem nevoie de ID-ul si optional rolul utilizatorilor, problema este ca userii sunt intr-o alta baza de date. Pentru a putea rezolva aceasta problema si pentru a pastra un numar redus de request-uri intre microservicii, vom pastra in baza de date a dispozitivelor o copie a tabelei de useri, ce va contine doar campurile id, username si role.

In consecinta, la fiecare alterare a bazei de date a userilor, microserviciul usermanagement va trimite un request la microserviciul devicemanagement cu modificarile efectuate.

Diagrama arhitecturii este urmatoarea:



5. Deployment

Deployment-ul aplicatiei se va realiza folosind Docker, identificand astfel 5 containere ce necesita deploy: 2 baze de date PostgreSQL, 2 backend-uri Java Spring Boot si frontend-ul.

Este importanta ordinea in care acestea sunt pornite, deoarece backend-ul userului si backend-ul dispozitivului se conecteaza la bazele de date la pornire.

Cele 5 containere ce necesita deploy sunt urmatoarele:

- frontend
- usermanagementspring
- devicemanagementspring
- usermanagementdb
- devicemanagementdb

Aceste containere vor rula in reteaua locala din Docker, iar pentru a putea interactiona cu aplicatia, porturile containerelor trebuie expuse.

Porturile expuse in localhost sunt urmatoarele:

:8080 – Microserviciul usermanagement

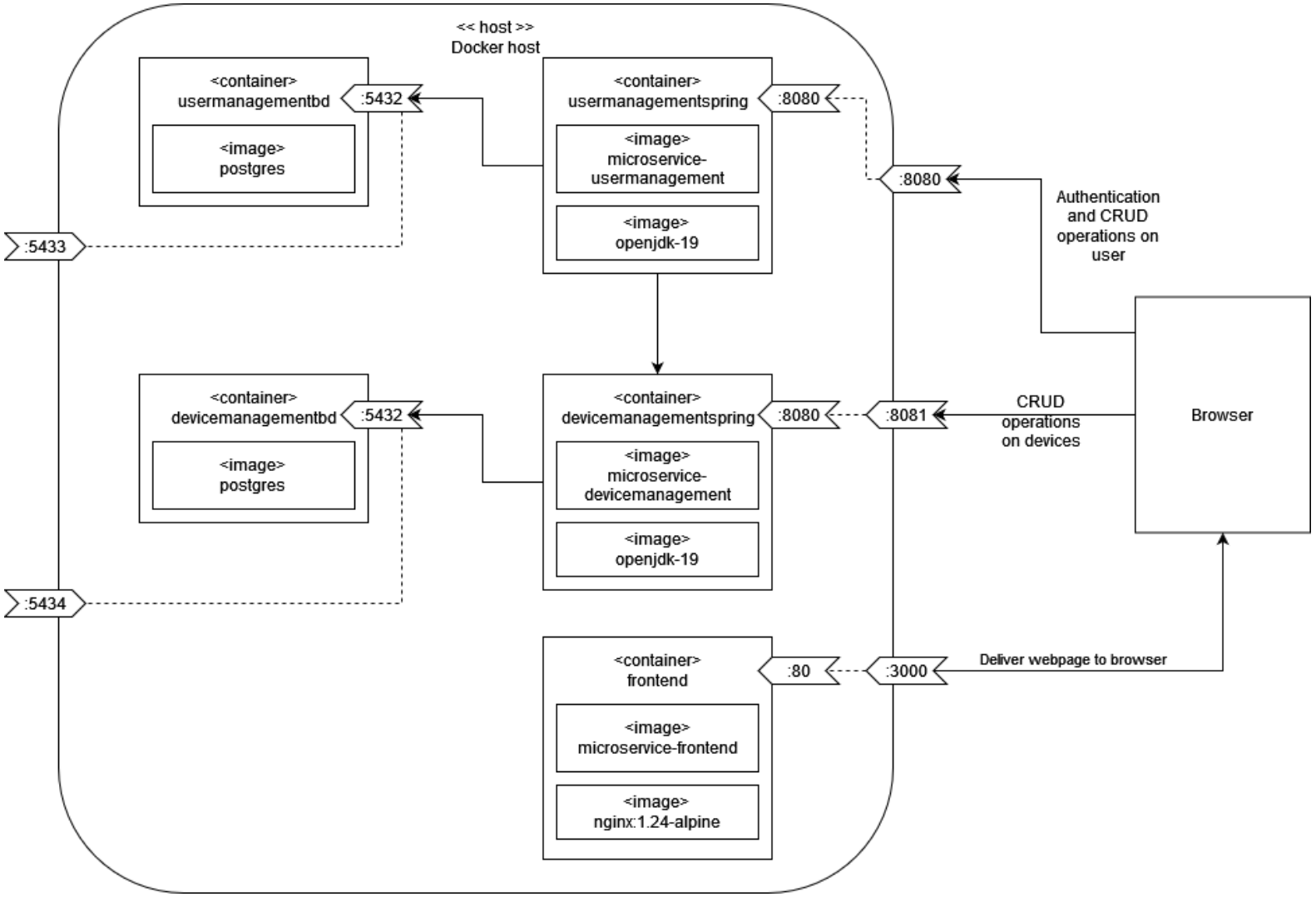
:8081 – Microserviciul devicemanagement

:3000 – Frontend

:5433 – Baza de date useri

:5434 – Baza de date dizpositive

Diagrama urmatoare va specifica care sunt porturile si adresele din reteaua interna Docker, si care sunt maparile cu porturile din masina host.


<< host >>
Docker host
<container>
usermanagementbd
<image>
postgres
:5432
<container>
usermanagementspring
<image>
microservice-
usermanagement
<image>
openjdk-19
:8080
:8080
:5433
Authentication
and CRUD
operations on
user
Browser
<container>
devicemanagementbd
<image>
postgres
:5432
<container>
devicemanagementspring
<image>
microservice-
devicemanagement
<image>
openjdk-19
:8080
:8081
CRUD
operations
on devices
:5434
<container>
frontend
<image>
microservice-frontend
<image>
nginx:1.24-alpine
:80
:3000
Deliver webpage to browser

6. Instructiuni de build

Ierarhia de fisiere este urmatoarea:

Assignment 1-|

|- devices-microservice -|

|-...

|- src

|- Dockerfile

|- usermanagement-microservice -|

|-...

|-src

|- Dockerfile

|-docker-compose.yml

|- management-frontend -|

|- ...

|- src

|- Dockerfile

Dockerfile

Cele doua backend-uri si frontend-ul contin cate un fisier Dockerfile ce genereaza cate o imagine docker, compiland si ruland codul din fisierul src.

Un exemplu de compilare si rulare utilizand Dockerfile este urmatorul:

```
FROM maven:3.9.1-amazoncorretto-19 AS build
COPY src /home/app/src
COPY pom.xml /home/app
WORKDIR /home/app
RUN mvn -f /home/app/pom.xml clean package -Dmaven.test.skip

FROM openjdk:19
WORKDIR /app
COPY --from=build /home/app/target/usernamagement-microservice-0.0.1-SNAPSHOT.jar /usr/local/lib/demo.jar
EXPOSE 8080
CMD ["java", "-jar", "/usr/local/lib/demo.jar"]
```

In prima sectiune ne folosim de imaginea maven:3.9.1 pentru a compila codul sursa. Instructiunile de COPY vor copia fisierele sursa din folder in imaginea maven la locatia /home/app/src

Comanda RUN va rula comanda maven de compilare si va genera un fisier usermanagement-microservice-0.0.1-SNAPSHOT.jar la locatia /home/app/target.

In a doua sectiune folosim imaginea docker openjdk:19 folosita pentru aplicatiile java. Prin instructiunea COPY vom copia fisierul Jar din build si prin CMD from rula aplicatia noastra.

docker-compose.yml

Configuratia containerelor este specificata in fisierul docker-compose.yml ce se foloseste de celelalte Dockerfile-uri pentru a putea genera imaginile si a porni containerele. In acest fisier de configurare se specifica pentru fiecare serviciu: numele containerului, numele imaginii sau fisierul Docker pentru generarea acesteia, maparea dintre portul intern si portul extern.

Un exemplu de definire a unui serviciu in docker-compose.yml

```
devicemanagementback:
  build:
    context: ../devices-microservice
  container_name: devicemanagementspring
  depends_on:
    - devicemanagementdb
  environment:
    - SPRING_DATASOURCE_URL=jdbc:postgresql://devicemanagementdb:5432/postgres
    - SPRING_DATASOURCE_USERNAME=postgres
    - SPRING_DATASOURCE_PASSWORD=root
  ports:
    - "8081:8080"
```

Rulare

Pentru a compila si executa containerele este necesar deschiderea unei console in folderul usermanagement-microservice si executat urmatoarea comanda:

docker-compose up

Odata ce comanda isi finalizeaza executia, imaginile sunt generate, containerele sunt pornite iar frontend-ul aplicatiei poate fi accesat la ip-ul localhost:3000.

Assignment 2

1. Introducere

Scopul acestui assignment este dezvoltarea unui microserviciu de monitorizare a consumului dispozitivelor. Acest microserviciu trebuie sa preia informatiile despre consum al fiecarui dispozitiv, sa il stocheze in baza de date, si sa notifice utilizatorul in momentul in care unul dintre dispozitive depaseste consumul maxim stabilit.

2. Cerinte functionale

- Dezvoltarea unui middle-ware bazat pe cozi ce preia mesaje de la dispozitive si le transmite microserviciului de monitorizare
- Salvarea consumului in baza de date a microserviciului pentru fiecare dispozitiv
- Notificarea asyncrona a utilizatorului in cazul in care un dispozitiv depaseste consumul maxim pe ora
- Sincronizarea datelor stocate despre dispozitiv in baza de date e microserviciului de monitorizare, cu baza de date a microserviciului pentru dispozitive folosind cozi

3. Tehnologii necesare pentru implementare

- RabbitMQ – pentru transmiterea mesajelor dintre dispozitive si microservicii
- Websockets – pentru notificarea utilizatorului asyncron

4. Arhitectura sistemului distribuit

RabbitMQ

RabbitMQ este un middle-ware capabil de a primi diverse mesaje de mai multi utilizatori si de a le trimite pe rand la diferiti ascultatori. Acest serviciu va fi lansat in docker la portul 5672, si va primi mesaje la doua topice:

- „consumptionqueue”: la acest topic se vor primi mesajele de la dispozitive in formatul <timestamp, device_uuid, consumption>
- „devicequeue”: la acest topic, microserviciul pentru dispozitive va trimite mesaje cu actualizari ce s-au efectuat la baza de date a dispozitivelor.

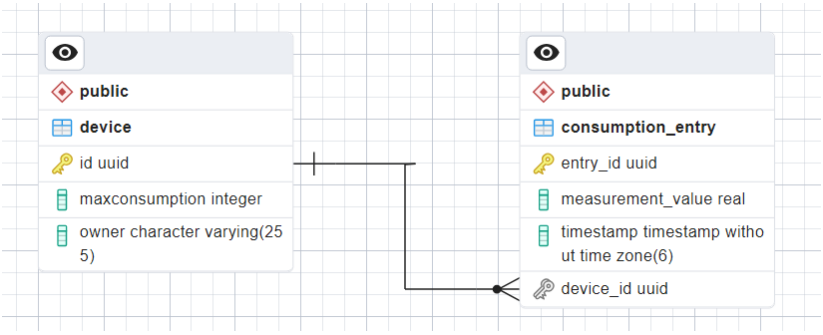
Monitoring microservice

Acest microserviciu are rolul de a asculta coada de mesaje de la topicul „consumptionqueue” din RabbitMQ, si de a le stoca in baza de date. Implementarea este realizata in framework-ul Spring.

Odata pornit microserviciul, fisierul de configurare RabbitMQConfig va initializa coada cu parametrii din docker-compose, specificand topicul, username-ul si parola pentru conectare.

In continuarea, pentru a putea interceptiona mesajele de la coada, am implementat clasa ConsumptionUpdateService ce contine doua metode anotate cu @RabbitListener. In momentul in care un mesaj de la coada este receptionat, backend-ul de monitoring va apela aceste functii si va stoca mesajele in baza de date.

Monitoring Database



In aceasta baza de date sunt prezente doua tabele:

- DEVICE: stocheaza UUID-ul dispozitivului, UUID proprietarului si consumul maxim al dispozitivului
- CONSUMPTION_ENTRY: stocheaza toate mesajele de la dispozitiv cu actualizari despre consumul instant la un anumt moment de timp.

Stocarea proprietarului dispozitivului este necesara pentru a sti ce utilizator sa fie notificat in cazul in care consumul unui dispozitiv depaseste limita impusa. Pentru a putea trimite mesaje asincron catre acesta, este necesara sa stabilim o conexiune prin websocket-uri.

Websocket

Configurarea websocket-ului este realizata in clasa WebsocketConfig ce specifica endpoint-ul „/monitor-websocket” necesar pentru a realiza handshake-ul initial dintre front-end si microserviciu. In metoda configureMessageBroker este definit topicul „/topic/notifications”, la care frontend-ul va fi nevoit sa dea subscribe pentru a putea receptiona mesaje.

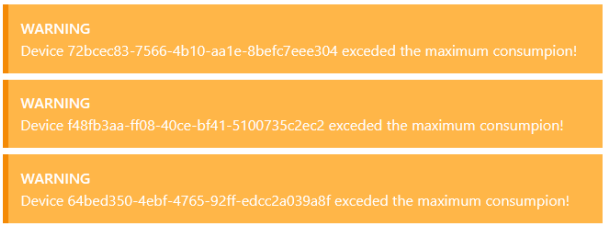
Pentru a putea identifica un user specific caruia sa ii trimitem notificare, la realizarea handshake-ului initial se va genera un UUID random ce va identifica sesiunea acestuia. Acest lucru se realizeaza intr-un handler custom in definit in clasa CustomHandshakeHandler.

Odata definit acest UUID, user-ul va trimite un mesaj asyncron la endpoint-ul „/subscribe”, cu token-ul sau de authentificare si astfel se va face o mapare intre sesiunea sa de conectare si username-ul sau.

Notificarile se vor realiza printr-un CRON job odata la 10 minute. Acesta va calcula consumul total din ultima ora pentru fiecare dispozitiv, iar in momentul in care un dispozitiv a depasit consumul maxim stabilit, se va cauta UUID-ul sesiunii corespunzatoarea proprietarului device-ului. Daca acesta este conectat, se va apela metoda sendNotification din clasa NotificationService cu datele dispozitivului respectiv.

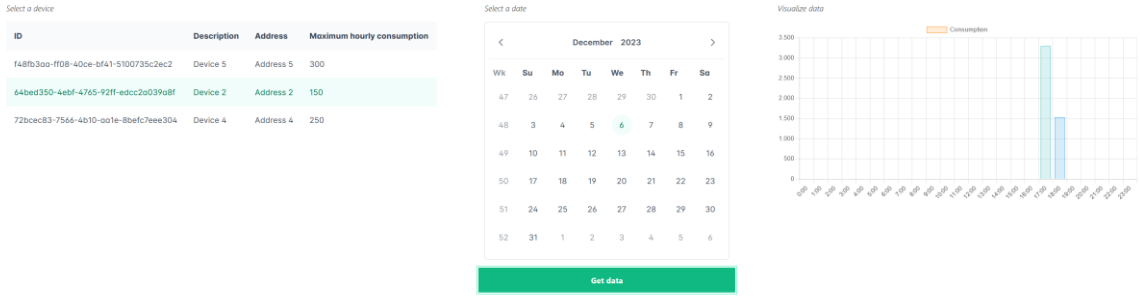
Frontend

Pentru a putea primi notificari asincron in browser, este necesara initializierea unui websocket si la nivelul frontend-ului. Pentru aceasta m-am folosit de stompJS. In momentul in care un user se logheaza, clientul STOMP va initializa handshake-ul la endpointul definit in monitoring microservice (ws://localhost:8082/monitor-websocket), va da subscribe la topicul „/user/topic/notifications” iar apoi va trimite un mesaj la endpoint-ul /app/subscribe pentru a instiinta microserviciul ce user se afla la conexiunea stabilita.



In momentul in care frontend-ul primeste un mesaj asincron, acesta va fi afisat catre utilizator sub forma unei notificari pop-ul.

Utilizatorul poate deasemenea vizualiza consumul pe ore al fiecarui dispozitiv intr-o anumita zi. Aceste date sunt preluate sub forma unui request HTML.



5. Deployment

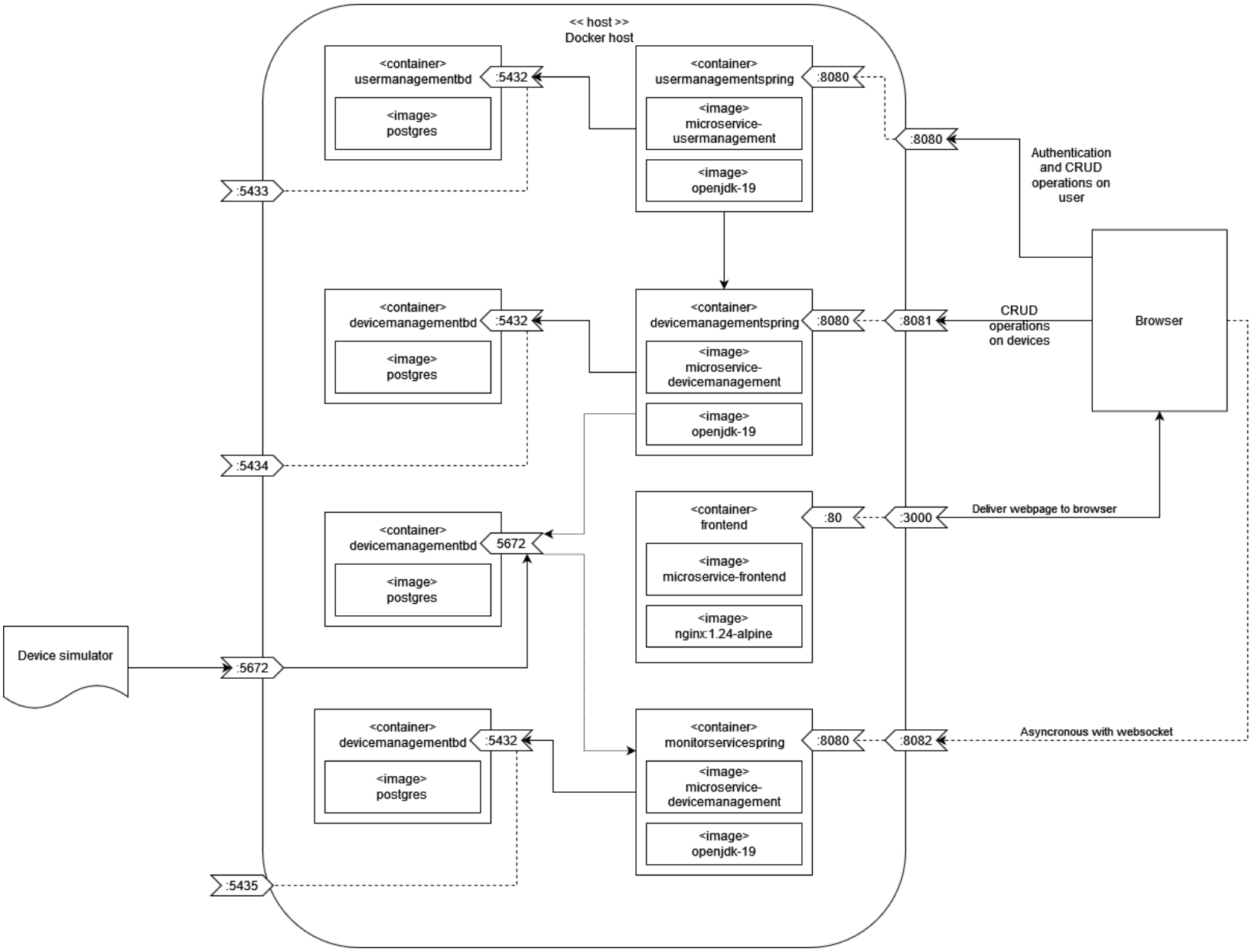
Din punct de vedere al deploymentului, se vor initializa inca 3 containere:

- monitoringspring: Microserviciul de monitorizare
- monitoringdb: Baza de date postgresql
- rabbitmq: serviciul RabbitMQ pentru coada de mesaje

In plus, se va initializa un script in python de ca simula dispozitivele, citind date dintr-un fisier csv si trimitandu-le la server-ul de RabbitMQ la un anumit interval. Scriptul va fi rulat inafara retelei docker, de aceea se va conecta la portul expus de rabbitMQ, 5672.

Atat microserviciul de monitoring cat si cel de device se vor conecta in cadrul retelei docker la coada RabbitMQ care va reprezenta intermediarul comunicarii dintre acestia, devicemonitoring transmitand update-uri cu privire la modificarile facute asupra bazei de date a dispozitivelor.

Diagrama actualizata de deployment este urmatoarea:


<< host >>
Docker host
<container>
usermanagementbd
<image>
postgres
:5432
:5433
<container>
usermanagementspring
<image>
microservice-
usermanagement
<image>
openjdk-19
:8080
:8080
Authentication
and CRUD
operations on
user
Browser
<container>
devicemanagementbd
<image>
postgres
:5432
:5434
<container>
devicemanagementspring
<image>
microservice-
devicemanagement
<image>
openjdk-19
:8080
:8081
CRUD
operations
on devices
<container>
devicemanagementbd
<image>
postgres
5672
<container>
frontend
<image>
microservice-frontend
<image>
nginx:1.24-alpine
:80
:3000
Deliver webpage to browser
Device simulator
:5672
<container>
devicemanagementbd
<image>
postgres
:5432
:5435
<container>
monitorservicespring
<image>
microservice-
devicemanagement
<image>
openjdk-19
:8080
:8082
Asyncronous with websocket

6. Instructiuni de build

Ierarhia de fisiere este urmatoarea:

Assignment 1-|

|- devices-microservice -|

|-...

|- src

|- Dockerfile

|- usermanagement-microservice -|

|-...

|-src

|- Dockerfile

|-docker-compose.yml

|- management-frontend -|

|- ...

|- src

|- Dockerfile

|- monitoring-microservice -|

|-...

|- src

|- Dockerfile

|- device-producer -|

|-producer.py

|- sensor.csv

Pasi de rulare:

Se va intra in folderul usermanagement-microservice cu un command line si se va rula:

docker-compose up

In folderul management-frontend se va rula comanda: npm run serve

Se va accesa adresa localhost:8083 in browser si in pagina de login se vor introduce urmatoarele credentiale

username: admin

password: admin

Odata logati, se vor crea utilizatorii si dispozitivele, iar apoi in command line se va rula scriptul producer.py ce va prelua automat UUID-urile dispozitivelor si va trimite date din sensor.csv la microserviciu.

Assignment 2

1. Introducere

Scopul acestei teme este realizarea unui sistem de chat bazat pe mesaje trimise asincron prin intermediul websocket-urilor. Utilizatorii trebuie sa fie capabili de a incepe o conversatie cu un admin iar acesta sa raspunda tot asincron.

2. Cerinte Functionale

- Dezvoltarea unui sistem de chat bazat pe websocket-uri
- Notificarea asincrona a utilizatorului in cazul in care mesajul sau este vazut de destinatar
- Notificarea asincrona a utilizatorului in momentul in care destinatarul scrie un mesaj catre el.
- Securizarea microserviciilor folosind JWT.

3. Tehnologii necesare pentru implementare

- Websocket-uri
- Spring security

4. Arhitectura sistemului distribuit

Microserviciul de chat

Pentru realizarea sistemului de chat s-a dezvoltat un microserviciu nou numit ChatMicroservice.

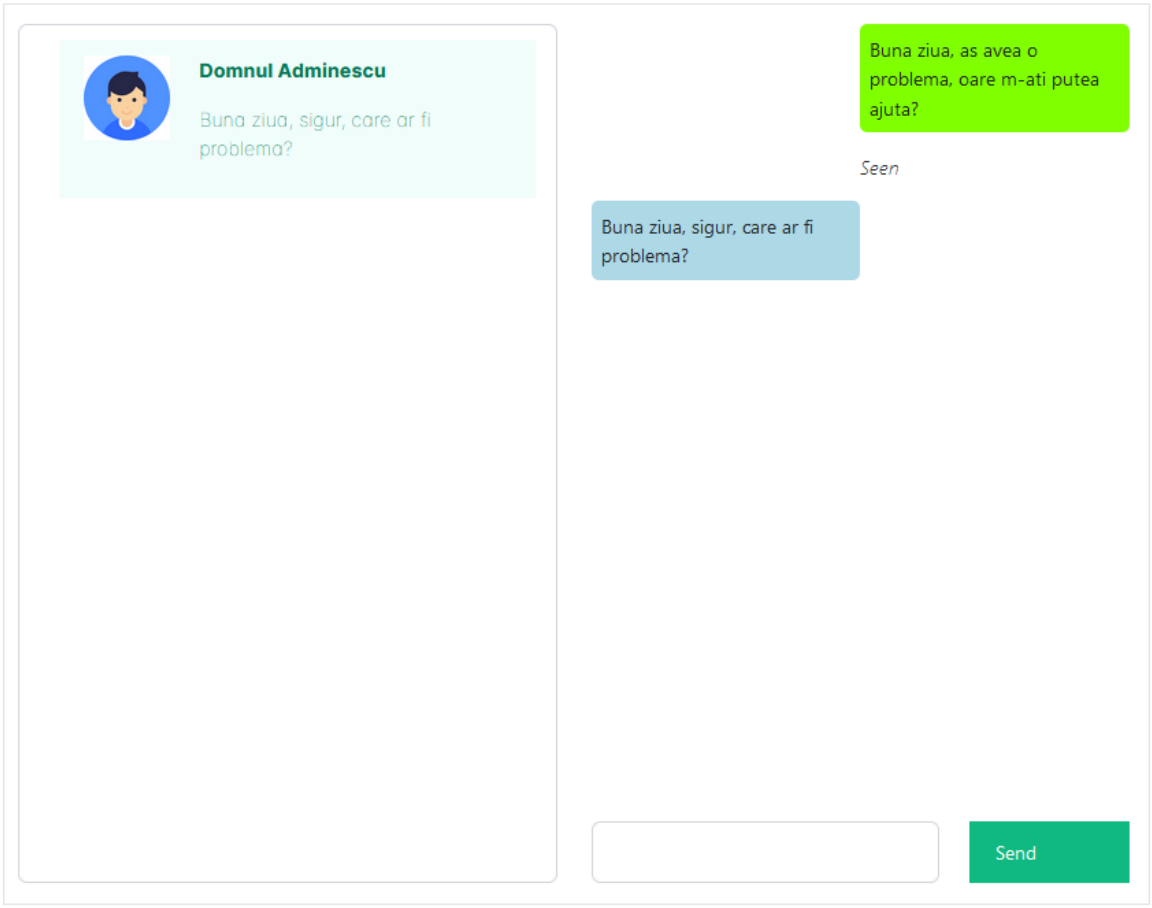
Acesta configureaza un endpoint (/chat-websocket) la care un client STOMP din frontend va initia un handshake pentru a stabili conexiunea asincrona dintre cele 2 componente. Endpoint-ul este configurat in clasa WebsocketConfig.

Odata ce conexiunea este stabilita, componenta frontend va trimite un mesaj la endpoint-ul /subscribe cu token-ul sau JWT, astfel microserviciul stabilind o legatura intre identificatorul sau de conexiune si username-ul utilizatorului, stiind in continuare in cazul in care este necesar sa trimita o notificare unui utilizator specific, care este conexiunea acestuia.

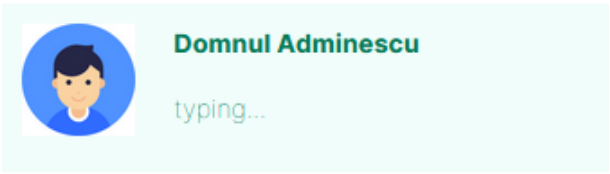
Pentru a putea facilita conversatia a doua persoane, microserviciul ChatMicroservice se va comporta ca un intermediar.

- Frontend-ul transmitaroului trimite un mesaj asincron catre endpoint-ul „/sendmessage”
- Microserviciul identifica dupa id-ul conexiunii username-ul transmitatorului si apoi id-ul de conexiune al destinatarului.
- Microserviciul transmite asincron mesajul catre destinatar.
- Destinatarul asculta topicul „/topic/chat” si primeste asincron mesajul.

Aceeasi procedura se realizeaza si pentru trimiterea de notificari, componenta frontend trimitand un mesaj asincron catre destinatar specificand ca utilizatorul scrie un mesaj sau a vizualizat un mesaj.



Odata ce notificarea este primita de destinatar, componenta frontend a acestuia marcheaza mesajul corespunzator cu „Seen” in cazul in care se specifica ca un mesaj a fost vizualizat, sau scrie in dreptul utilizatorului mesajul „Typing” in cazul in care notificarea marcheaza ca utilizatorul scrie un mesaj.

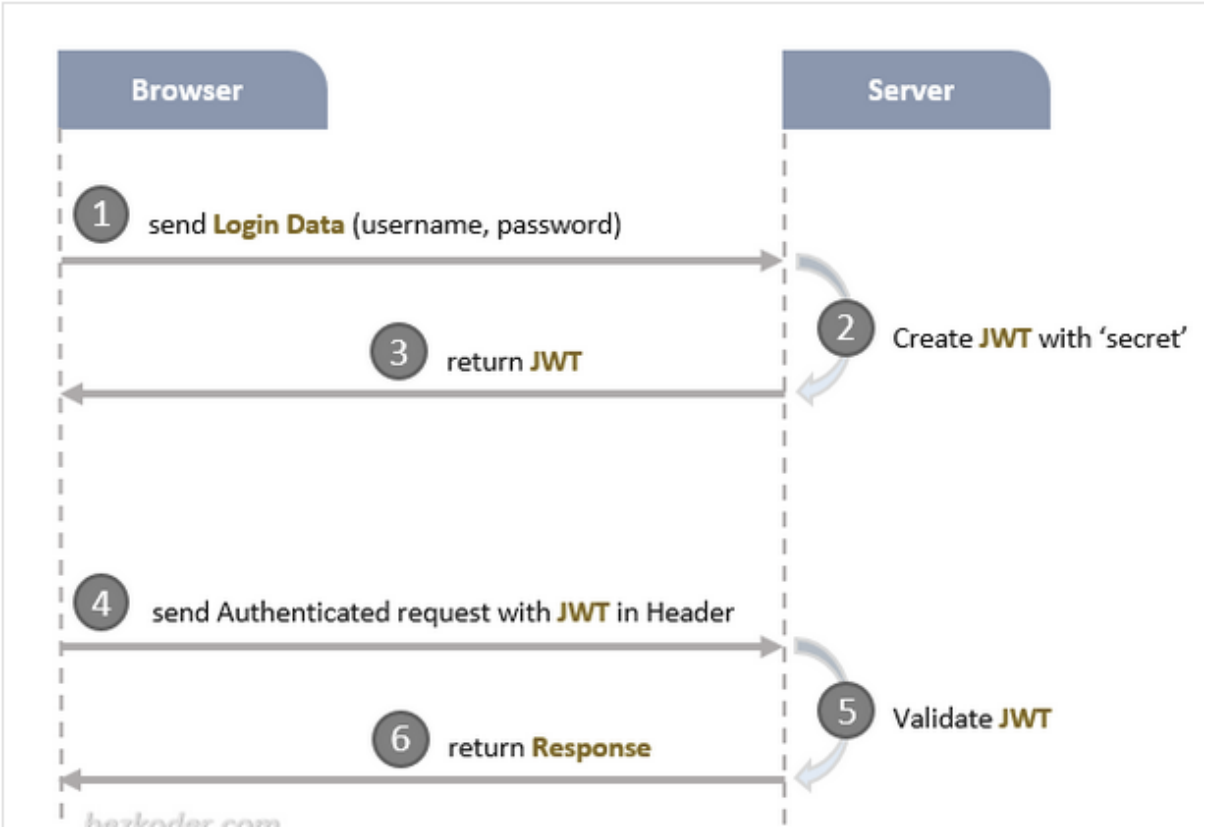


Securitatea

Pentru a asigura securitatea microserviciilor, vom adauga la microserviciul UserManagement o componenta de autorizare cu JWT.

In momentul in care un user se logheaza cu succes, microserviciul UserManagement va genera un sir de caractere in care va fi codificat username-ul acestuia si care utilizatorul il va folosi pentru asi autoriza toate request-urile HTTP viitoare.

Autorizarea token-ului se va face folosind un string secret pe care doar backend-ul il va cunoaste, astfel asigurandu-se ca token-ul respectiv nu va putea fi manipulat.



Pentru a implementa acest lucru, ne vom folosi de Spring Security.

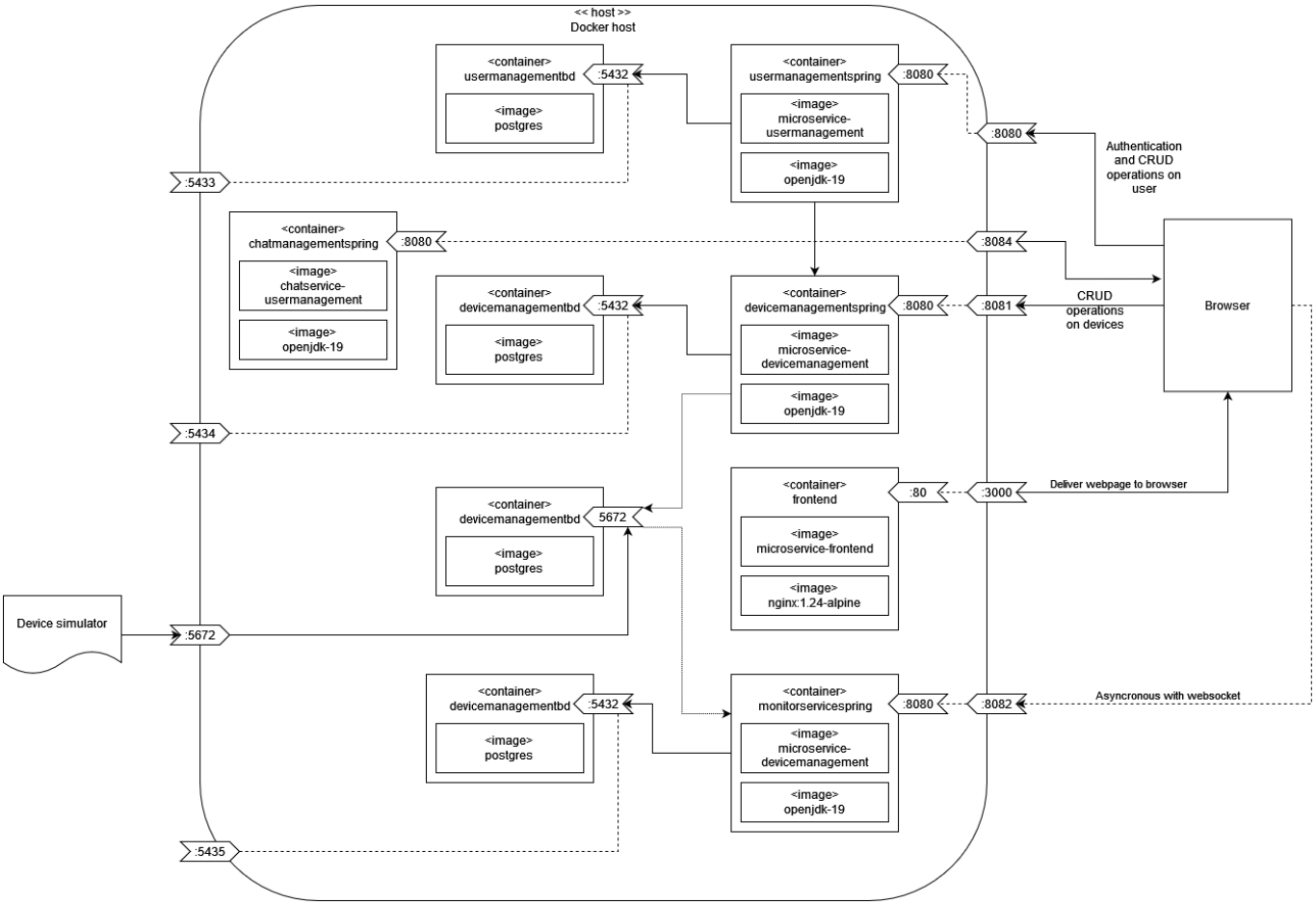
Vom defini un filtru custom pentru request-uri HTTP, astfel inaintea procesarii fiecarui request, vom cauta header-ul „Authorization", vom valida si decodifica string-ul JWT si vom seta rolul utilizatorului.

Odata ce permisiunile au fost setate, in componenta SecurityFilterChain vom defini care endpoint-uri sunt accesibile pentru fiecare rol, astfel blocand accesul utilizatorilor neautorizati.

Aceste componente vor exista in fiecare microserviciu pe care dorim sa il securizam.

5. Deployment

Din punct de vedere al deploymentului, se va da deploy la un microserviciu nou, ChatMicroservice, care va fi conectat la portul 8084.



6. Instructiuni de utilizare

Ierarhia de fisiere este urmatoarea:

Assignment 1-|

|- devices-microservice -|

|-...

|- src

|- Dockerfile

|- usermanagement-microservice -|

|-...

|-src

|- Dockerfile

|-docker-compose.yml

|- management-frontend -|

|- ...

|- src

|- Dockerfile

|- monitoring-microservice -|

|-...

|- src

|- Dockerfile

|- chat-microservice -|

|-...

|- src

|- Dockerfile

|- device-producer -|

|-producer.py

|- sensor.csv

Pasi de rulare:

Se va intra in folderul usermanagement-microservice cu un command line si se va rula:

docker-compose up

In folderul management-frontend se va rula comanda: npm run serve

Se va accesa adresa localhost:8084 in browser si in pagina de login se vor introduce urmatoarele credentiale

username: admin

password: admin

Se va intra de pe alt browser la localhost:8084 si se va inregistra un utilizator nou. Se va testa aplicatia trimitand in sectiunea de chat un mesaj catre admin.