# Aplicație pentru gestionarea unei rețele energetice

## Sisteme Distribuite

## Andrei-Denis Alexandru

## Grupa 30644

## Profesor îndrumător: Marcel Antal

## An universitar 2023-2024

1. Introducere

Sistemul de Management al Energiei dezvoltat este compus dintr-un frontend și trei microservicii create pentru administrarea utilizatorilor și a dispozitivelor lor inteligente de măsurare a energiei. Acest sistem poate fi accesat de două tipuri de utilizatori după un proces de autentificare: administrator și clienți. Administratorul poate efectua operații CRUD asupra conturilor, dispozitivelor de măsurare a energiei și asupra asignării utilizatorilor la dispozitive (fiecare utilizator poate deține unul sau mai multe dispozitive). Clientul isi poate vedea consumul energetic zilnic al dispozitivelor asociate.

1. Cazuri de utilizare

Cerintele tehnice ale proiectului identifica cei 2 actori care utilizeaza platforma: administrator si client.

Administratorul poate face log-in in aplicatie si are responsabilitatea de a crea si gestiona conturile clientilor si dispozitivele electronice, fiind responsabil si de asocierea dintre acestea. Un utilizator de tip administrator poate fi inserat in baza de date (hard-codat) sau poate fi creat de catre un alt administrator. Un dispozitiv nu poate fi creat fara sa apartina unui utilizator deja existent.

Clientul are dreptul de a se loga in platforma si de a-si vedea toate dispozitivele care ii sunt asociate. Acesta nu isi poate crea un cont nou, responsabilitatea ramanand administratorului. Clientul va fi atentionat in momentul in care un dispozitiv asociat acestuia depaseste limita de energie consumata pe ora.

1. O imagine care conține text, captură de ecran, diagramă, schiță

   Descriere generată automatStructura si tehnologiile folosite

Fig.1. Diagrama arhitecturala stratificata

Pentru implementarea sistemului, au fost utilizate următoarele tehnologii:

* Microservicii:
  + Microserviciu de Management al Utilizatorilor: Java Spring
  + Microserviciu de Management al Dispozitivelor: Java Spring
  + Microserviciu de Management al Urmaririi Dispozitivelor: Java Spring
  + Tehnologii de comunicare intre microservicii: REST API, RabbitMQ, WebSockets
* Framework pentru aplicatia client: Javascript (React.js)
* Platforma de virtualizare in containere: Docker

Comunicarea eficientă între microservicii se realizează utilizând arhitectura REST (Representational State Transfer) pe baza protocolului HTTP. Acest cadru permite microserviciilor să transmită date între ele prin intermediul cererilor HTTP, oferind o modalitate simplă și standardizată de comunicare. REST API asigură interoperabilitatea între microservicii și le permite să funcționeze independent, fără a crea dependențe strânse între ele.

O alta modalitate eficienta pentru a comunica intre servicii este folosirea cozilor de mesaje (serviciul RabbitMQ) bazata pe un mecanism de tip Publisher – Consumer. Aceasta tehnica a fost folosita atat pentru sincronizarea masuratorilor capturate de catre simulator cat si a modificarilor dispozitivelor si serviciul de monitorizare a dispozitivelor.

Pentru a asigura sincronizarea între cele două microservicii, am implementat o abordare eficientă utilizând Spring WebClient. În acest context, microserviciul de clienți utilizează WebClient pentru a efectua cereri către microserviciul de dispozitive.

Notificarea utilizatorului despre excederea limitei de consum al unui device se face cu ajutorul tehnologiei WebSocket, aplicata intre Microserviciul de monitorizare al dispozitivelor si frontend.

Baza de date din microserviciul de dispozitive conține un tabel simplificat care păstrează o listă a utilizatorilor din microserviciul de clienți, conținând în general doar ID-urile și numele de utilizator. Această bază de date din microserviciul de dispozitive este actualizată în timp real atunci când apar operații asupra tabelului utilizatorilor din microserviciul de clienți.

Procesul de actualizare și sincronizare se desfășoară astfel:

* O operație de modificare sau adăugare este efectuată în tabelul de utilizatori din microserviciul de clienți.
* Microserviciul de clienti utilizează WebClient pentru a solicita actualizarea listei de utilizatori din microserviciul de clienți.
* Microserviciul de dispozitive răspunde cu datele actualizate despre utilizatori sau cu un raspuns de tip eroare in cazul in care operatia nu a avut succes.
* Microserviciul de clienti trimite catre locul din care a fost apelat un raspuns bazat pe raspunsul primit anterior de la microserviciul de dispozitive.

Această abordare asigură o sincronizare continuă și eficientă între cele două microservicii, permițând microserviciului de dispozitive să aibă întotdeauna o imagine actualizată a utilizatorilor din microserviciul de clienți. Prin intermediul acestui mecanism de comunicare, actualizările sunt propagate în timp real, asigurând o funcționarea întregului sistem de Management al Energiei.

In cadrul structurii containerelor definite cu ajutorul Docker, gasim o abordare bine organizată pentru gestionarea componentelor. Fiecare serviciu reprezintă o componentă distinctă a aplicației și este izolat în propriul container. Imaginile Docker sunt utilizate pentru a crea aceste containere.  
Pentru bazele de date, avem serviciile "userdb" și "devicedb", care sunt definite folosind imaginile MySQL. Aceste servicii sunt configurate cu variabile de mediu pentru a specifica numele bazelor de date, parolele de acces și altele. Ele rulează pe porturi diferite și utilizează volume Docker pentru a stoca datele persistente.

Microserviciile "spring-user" și "spring-device" sunt construite din imagini Docker personalizate, definite în fișierele Dockerfile corespunzătoare. Aceste servicii au propriile variabile de mediu pentru a se conecta la bazele de date și pentru a configura interacțiunile lor.

Serviciul "frontend" reprezintă aplicația client și este construit dintr-o imagine Docker personalizată. Acest serviciu interacționează cu microserviciile din spate folosind adresele IP și porturile corespunzătoare. Toate aceste servicii sunt conectate într-o rețea Docker externă ("demo\_net") pentru a permite comunicarea între ele.

O imagine care conține captură de ecran, text, proiectare

Descriere generată automat

Fig. 5. Diagrama de deployment