MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

DISTRIBUTED SYSTEMS

Assignment 1

Request-Reply Communication

Student: Buda Andreea Rodica
Facultatea de Automatică și Calculatoare
Specializarea Calculatoare
Grupa 30244

Cuprins

1.Obiectivul Proiectului

Scopul acestui proiect este dezvoltarea unui sistem de gestionare a energiei care să permită utilizatorilor să monitorizeze și să gestioneze consumul de energie al dispozitivelor dintr-o locuință sau organizație. Aplicația web include roluri și acces diferențiat pentru administratori și clienți: administratorii pot gestiona utilizatorii și dispozitivele, iar clienții pot vizualiza detaliile dispozitivelor proprii și consumul acestora.

Obiectivele includ o arhitectură scalabilă bazată pe microservicii, securitate avansată, o bază de date sigură și o interfață frontend intuitivă. Sistemul nu doar monitorizează și controlează consumul de energie, ci promovează și responsabilitatea energetică în comunitate.

2. Analiza

2.1 Cerințe Funcționale

Cerințele funcționale stabilesc acțiunile și comportamentele de bază ale sistemului. În cadrul proiectului EMS, cerințele funcționale sunt următoarele:

1. **Autentificare și Roluri de Utilizatori**: Sistemul permite autentificarea utilizatorilor și accesul la funcții specifice fiecărui rol (Administrator pentru gestionare, Client pentru vizualizare).
2. **Gestionarea Utilizatorilor (Administrator)**: Creare, citire, actualizare și ștergere de conturi de utilizator.
3. **Gestionarea Dispozitivelor (Administrator)**: CRUD pe dispozitivele de măsurare a energiei.
4. **Asocierea Utilizator-Dispozitiv**: Administratorul poate lega utilizatorii de dispozitive.
5. **Vizualizarea Dispozitivelor (Client):** Clienții pot vedea dispozitivele proprii și consumul acestora.
6. **Restricționarea Accesului**: Implementarea unui sistem de autorizare pentru a împiedica accesul neautorizat.

2.2 Cerințe Non-Funcționale

Cerințele non-funcționale stabilesc caracteristici precum performanța, securitatea și experiența utilizatorilor. În acest proiect, cerințele non-funcționale sunt:

1. **Performanță**: Timp de răspuns sub 2 secunde pentru operațiuni CRUD.
2. **Securitate**: Autentificare și autorizare pentru protecția datelor.
3. **Scalabilitate**: Sistemul trebuie să gestioneze creșterea numărului de utilizatori și dispozitive.
4. **Ușurință în utilizare**: Interfață intuitivă, ușor de navigat.
5. **Mentenabilitate**: Cod documentat și structurat pentru întreținere.
6. **Compatibilitate**: Funcționare uniformă pe toate platformele și browserele majore.

2.3 Tehnologii Utilizate

În dezvoltarea acestui sistem de management energetic, următoarele tehnologii sunt utilizate pentru a satisface cerințele funcționale și non-funcționale:

1. **Backend - Microservicii RESTful:**
   - *Java Spring Boot* pentru crearea microserviciilor REST, utilizate pentru gestionarea utilizatorilor și dispozitivelor de energie.
   - *Docker* pentru containerizarea serviciilor, asigurând astfel izolare și portabilitate între diferite medii de rulare.
2. **Frontend - Aplicație Web:**
   - *ReactJS* pentru dezvoltarea interfeței de utilizator, oferind o experiență intuitivă și interactivă.
3. **Baze de date:**
   - *PostgreSQL* pentru stocarea datelor despre utilizatori și dispozitive, oferind consistență și fiabilitate în gestiunea datelor.
4. **Securitate și Autentificare:**
   - Implementarea securității aplicației utilizând sesiuni și autentificare pentru a asigura accesul autorizat la funcționalitățile aplicației.

2.4 Cazuri de Utilizare și Scenarii

În cadrul Sistemului de Management al Energiei, există două roluri principale: **Administratorul** și **Clientul**. Fiecare rol are acces la funcționalități specifice care îi permit să își îndeplinească scopurile în sistem.

Rolul Administratorului

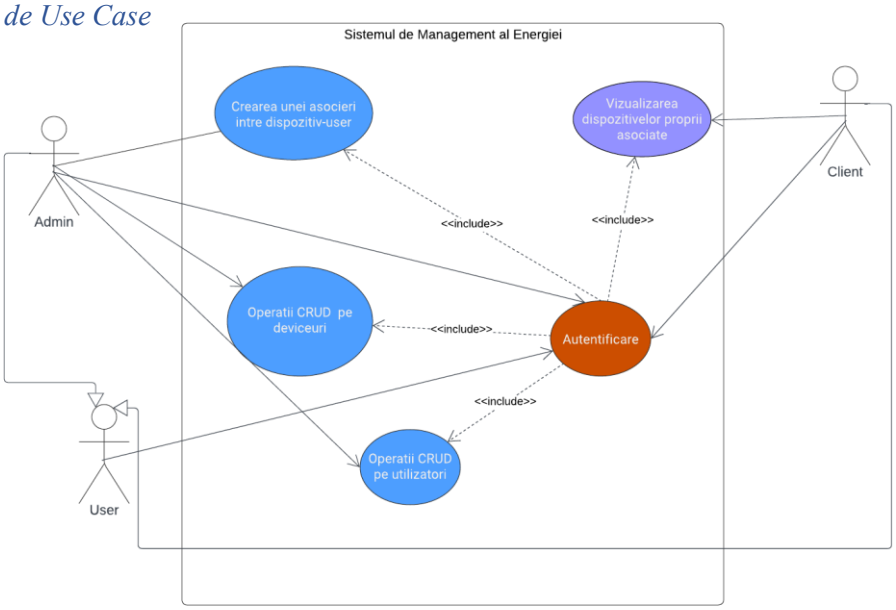
1. **Gestionare utilizatori (CRUD) - Scenariu**: Administratorul poate crea, vizualiza, actualiza și șterge utilizatori în secțiunea de management a utilizatorilor.

2. **Gestionare dispozitive (CRUD) - Scenariu**: Administratorul adaugă, vizualizează, actualizează sau șterge dispozitive, inclusiv configurarea consumului maxim de energie pentru fiecare dispozitiv.
3. **Asociere utilizator-dispozitiv - Scenariu**: Administratorul asociază un dispozitiv unui utilizator, facilitând monitorizarea consumului de către client.

Rolul Clientului

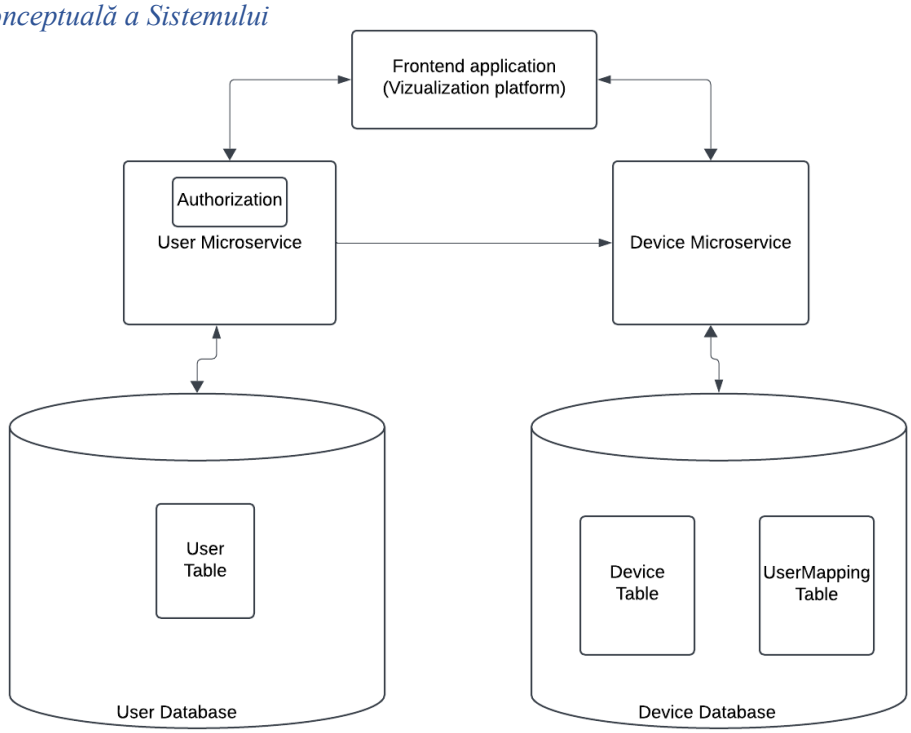
1. **Vizualizare dispozitive associate - Scenariu**: Clientul accesează lista dispozitivelor proprii și vizualizează detalii precum adresa, descrierea și consumul maxim.
2. **Monitorizare consum energie - Scenariu**: Clientul analizează consumul fiecărui dispozitiv asociat pentru a identifica potențialele economii de energie.

2.5 Diagrama de Use Case



3. Proiectare

3.1 Arhitectura Conceptuală a Sistemului



a. Componentele Sistemului:

1. Frontend React (Client):
    - Permite accesul utilizatorilor la funcționalitățile sistemului, inclusiv autentificarea și redirecționarea pe baza rolului.
    - Utilizează NGINX pentru a trimite cererile către microservicii, asigurând o separare clară între frontend și backend.
2. Users Microservice:
    - Oferă operațiuni CRUD pentru utilizatori, inclusiv autentificare bazată pe sesiuni și cookie-uri, limitând accesul în funcție de rol.
    - Baza de date stochează informații despre utilizatori și asocierile lor cu dispozitivele.
3. Device Microservice:
    - Permite operațiuni CRUD pentru dispozitive și asocieri între utilizatori și dispozitive, stocând detalii despre consumul maxim de energie.
    - Asocierile se realizează prin cereri HTTP către microserviciul Users, cu tranzacții SQL pentru a menține consistența datelor.

b. Comunicația între Microservicii și Aplicație:

- Microserviciile comunică prin API REST folosind metode HTTP standard (GET, POST, PUT, DELETE). Sistemul folosește cereri asincrone pentru a sincroniza datele între microservicii.
- Pentru integritatea datelor, operațiile de sincronizare sunt tranzacționale. De exemplu, la adăugarea unui utilizator nou, Users Microservice trimite o cerere POST la Device Microservice pentru a crea o înregistrare corespunzătoare, garantând astfel fie o tranzacție completă, fie o anulare în caz de eroare.

3.2 Diagrama de Deployment

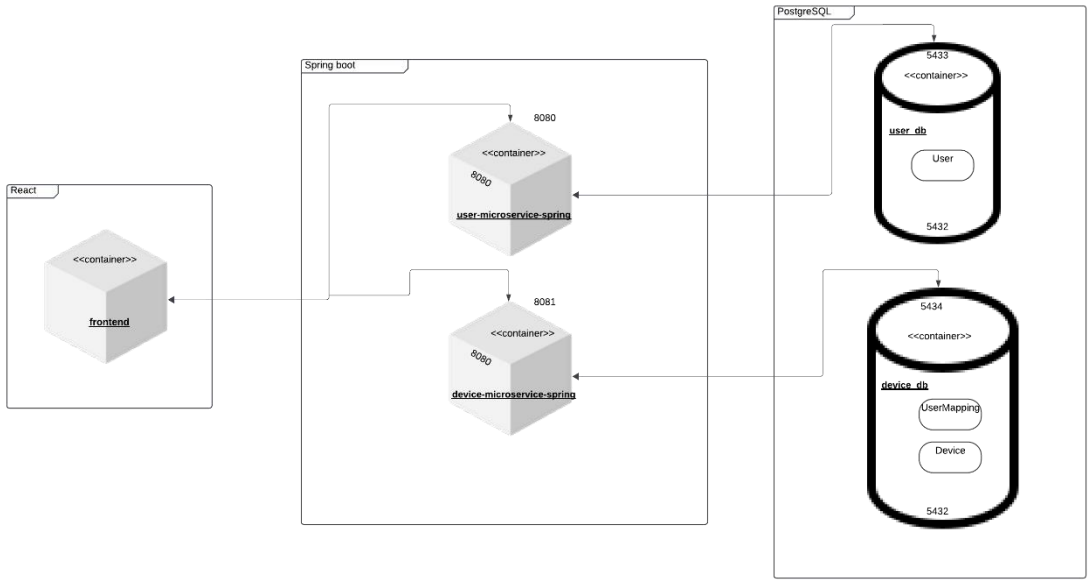


Diagrama de deployment ilustrează sistemul organizat în containere Docker, fiecare componentă configurată să comunice eficient cu celelalte.

Structura Containerelor și Rolul Fiecărei Componente:

1. **User_db (PostgreSQL)**:
    - Gestionează datele utilizatorilor și este accesibil doar microserviciului de utilizatori.
    - Expus extern pe portul 5433, cu comunicare internă pe portul 5432.
2. **Device_db (PostgreSQL)**:
    - Stochează datele dispozitivelor, fiind accesibil doar microserviciului pentru dispozitive.
    - Expus extern pe portul 5434, comunicare internă pe portul 5432.
3. **User-microservice-spring și Device-microservice-spring (Spring Boot)**:
    - Fiecare microserviciu rulează într-un container separat, definit prin Dockerfile.
    - User Microservice funcționează pe portul 8080, iar Device Microservice pe portul 8081.
    - Conexiunea la baze de date se face prin variabile de mediu definite în docker-compose.yml, pentru configurare și acces flexibil.

4. **Frontend (React + NGINX)**:
    - Include aplicația React și un server NGINX configurat ca reverse proxy.
    - Rulează pe portul 3000, direcționând cererile frontend-ului către microservicii prin rutele /userserver și /deviceserver, cu variabile de mediu pentru IP și port.

Această structură pe containere asigură izolarea, configurarea dinamică și scalabilitatea componentelor, contribuind la mentenabilitate și performanță optimizate.

4. Implementare

4.1 Instrumente Utilizate

În cadrul proiectului de gestionare a energiei, au fost utilizate următoarele instrumente pentru dezvoltarea și containerizarea aplicației:

1. **Java Spring Boot**:
    - Utilizat pentru crearea microserviciilor de utilizatori și dispozitive.
    - Spring Boot a fost ales pentru că oferă o arhitectură modulară și ușor de scalat, permițând implementarea rapidă a serviciilor RESTful.
2. **React**:
    - Framework-ul JavaScript folosit pentru dezvoltarea frontend-ului.
    - React a fost ales pentru flexibilitatea sa și pentru capacitatea de a crea o interfață de utilizator dinamică și interactivă.
3. **PostgreSQL**:
    - Baza de date relațională utilizată pentru stocarea datelor utilizatorilor și dispozitivelor.
    - Fiecare microserviciu are o bază de date PostgreSQL dedicată, asigurând astfel izolare și performanță sporită.
4. **Docker și Docker Compose**:
    - Docker a fost utilizat pentru a containeriza fiecare componentă a aplicației, de la microservicii până la frontend și baze de date.
    - Docker Compose a permis orchestrarea containerelor și configurarea variabilelor de mediu pentru a asigura comunicația între componentele aplicației.
5. **NGINX**:
    - Configurat ca reverse proxy în containerul de frontend, NGINX direcționează cererile către microserviciile backend în funcție de rutele definite, protejând detaliile URL-urilor interne.
6. **Postman** (pentru testare):
    - Utilizat pentru testarea și verificarea API-urilor RESTful oferite de microservicii, Postman a permis simularea operațiilor CRUD și verificarea răspunsurilor microserviciilor înainte de integrarea cu frontend-ul.

4.2 Fișier README

Aplicația finală poate fi rulată în Docker urmând pașii de mai jos:

1. Clonează repository-urile backend și frontend de pe acest repository principal: https://gitlab.com/budaandreea02/ds2024_30244_buda_andreea_assignment_1
2. Mută fișierul docker-compose.yml în directorul de lucru, același director unde au fost clonate anterior ambele repository-uri.
3. Deschide terminalul în acest director și pornește Docker Desktop pentru a activa daemon-ul Docker.
4. Construiește și rulează containerele utilizând comenzile: **docker-compose build, docker-compose up.**
5. Accesează aplicația în browser după ce containerele sunt pornite:
    - Aplicația va fi disponibilă la adresa http://localhost:3000/.

Pentru a rula componentele separat (fără Docker), backend-urile pot fi pornite dintr-un IDE de dezvoltare (ex. IntelliJ) la adresele http://localhost:8080/, respectiv, http://localhost:8081/, iar frontend-ul poate fi pornit cu comanda npm start la adresa http://localhost:3000/.