MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE
UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

DISTRIBUTED SYSTEMS

Assignment 2

Asynchronous Communication and Real-Time Notofication

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

Cuprins

1.Obiectivul Proiectului

Scopul acestui proiect este dezvoltarea unui microserviciu de Monitorizare și Comunicare pentru un sistem de gestionare a energiei, care să permită integrarea cu microserviciile existente pentru gestionarea utilizatorilor și dispozitivelor. Acest microserviciu va utiliza RabbitMQ pentru comunicarea asincronă între microserviciile Device și Monitoring & Communication, iar notificările în timp real vor fi trimise prin WebSocket atunci când consumul de energie al unui dispozitiv depășește un prag definit. Aceasta va îmbunătăți eficiența gestionării consumului de energie, oferind notificări instantanee și actualizări ale statusului dispozitivelor în timp real.

2. Analiza

2.1 Cerințe Funcționale

- Middleware Message-Oriented: Sistemul trebuie să permită simularea datelor și trimiterea lor către microserviciul de Monitorizare, folosind RabbitMQ pentru schimbul de mesaje în format JSON.
- Componentă Consumator: Microserviciul de Monitorizare va consuma mesajele din coadă, va procesa datele și va trimite notificări prin WebSocket când consumul unui dispozitiv depășește limita maximă.
- Persistența Datelor: Consumurile de energie vor fi salvate într-o bază de date PostgreSQL.
- Replicare Microservicii: Microserviciile de User și Device vor fi replicate pentru a asigura disponibilitatea și scalabilitatea.

2.2 Cerințe Non-Funcționale

- **Integrarea Consumatorului**: Componentele asincrone vor fi integrate în microserviciul de Monitorizare, asigurând astfel procesarea datelor în timp real.
- **Load Balancer**: Utilizarea Traefik pentru a distribui traficul între replicile microserviciilor User și Device.
- **Integrarea cu Tema Anterioară**: Sistemul va simula date pentru un dispozitiv existent cu un ID valid, iar un client autentificat va putea vizualiza istoricul consumului de energie pentru o zi selectată, sub formă de grafic.

2.3 Tehnologii Utilizate

Pentru realizarea acestui proiect, au fost utilizate următoarele tehnologii:

1. **Backend**:
   - **Java Spring Boot**: Utilizat pentru crearea microserviciilor RESTful pentru gestionarea utilizatorilor și dispozitivelor.
   - **RabbitMQ**: Folosit pentru comunicarea asincronă între microserviciile Device și Monitoring & Communication.
   - **WebSocket**: Implementat pentru notificările în timp real în frontend.
2. **Frontend**:
   - **ReactJS**: Folosit pentru crearea interfeței de utilizator, care interacționează cu backend-ul pentru a vizualiza datele.
3. **Baze de Date**:
   - **PostgreSQL**: Folosit pentru stocarea datelor despre utilizatori, dispozitive și consumul de energie.
4. **Containerizare**:
   - **Docker**: Pentru a crea containere izolate pentru microservicii și baze de date.
   - **Traefik**: Folosit ca reverse proxy și load balancer pentru microservicii.

2.4 Cazuri de Utilizare și Scenarii

Rolul Administratorului:

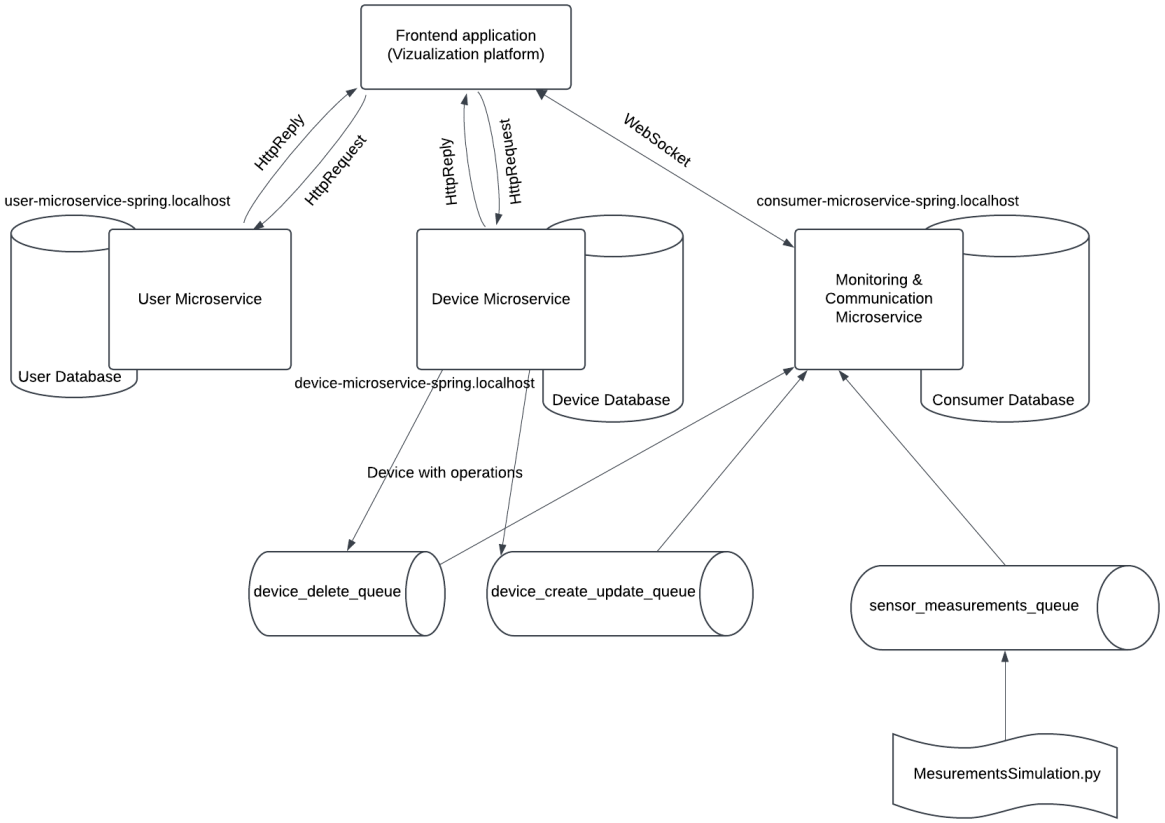
- Administratorul poate adăuga, edita și șterge utilizatori și dispozitive.
- Poate vizualiza istoricul consumului de energie al dispozitivelor.

Rolul Clientului:

- Clientul se poate autentifica și vizualiza consumul de energie al propriilor dispozitive.
- Primește notificări în timp real atunci când consumul de energie depășește limita maximă definită pentru un dispozitiv.

3. Proiectare

3.1 Arhitectura Conceptuală a Sistemului



Sistemul este bazat pe o arhitectură de microservicii, fiecare microserviciu având o funcționalitate specifică și o bază de date proprie. Comunicarea între microservicii și între microservicii și frontend se face prin protocoale standardizate, fie asincrone (pentru RabbitMQ), fie sincrone (pentru interacțiunile directe între microservicii).

Componentele principale:

1. Microserviciul pentru utilizatori (User Microservice): Permite gestionarea utilizatorilor, operațiuni CRUD (Create, Read, Update, Delete) și autentificare. Utilizatorii sunt autentificați folosind sesiuni web și cookie-uri. Datele sunt stocate într-o bază de date PostgreSQL dedicată.
2. Microserviciul pentru dispozitive (Device Microservice): Permite gestionarea dispozitivelor de măsurare a energiei, asocierea acestora cu utilizatorii și actualizarea datelor de consum energetic. Dispozitivele sunt stocate într-o altă bază de date PostgreSQL.
3. Microserviciul pentru monitorizare și comunicare (Monitoring & Communication Microservice): Acesta se ocupă cu monitorizarea consumului de energie al dispozitivelor, salvând și procesând aceste date. Consumurile de energie sunt stocate într-o bază de date separată (PostgreSQL). Comunicarea între acest microserviciu și frontend se face prin WebSocket pentru notificările în timp real.

Cum se face comunicarea:

- Comunicarea sincrona între microservicii și frontend se face folosind HTTP Requests (GET, POST, PUT, DELETE). Microserviciul de utilizatori interacționează cu baza de date pentru a crea, citi, actualiza și șterge utilizatori.
- Comunicarea asincronă între microservicii se face folosind RabbitMQ, unde mesajele de la microserviciul de dispozitive (pentru actualizări sau ștergeri de dispozitive) sunt consumate de microserviciul de monitorizare.

Mecanisme de consistență a datelor:

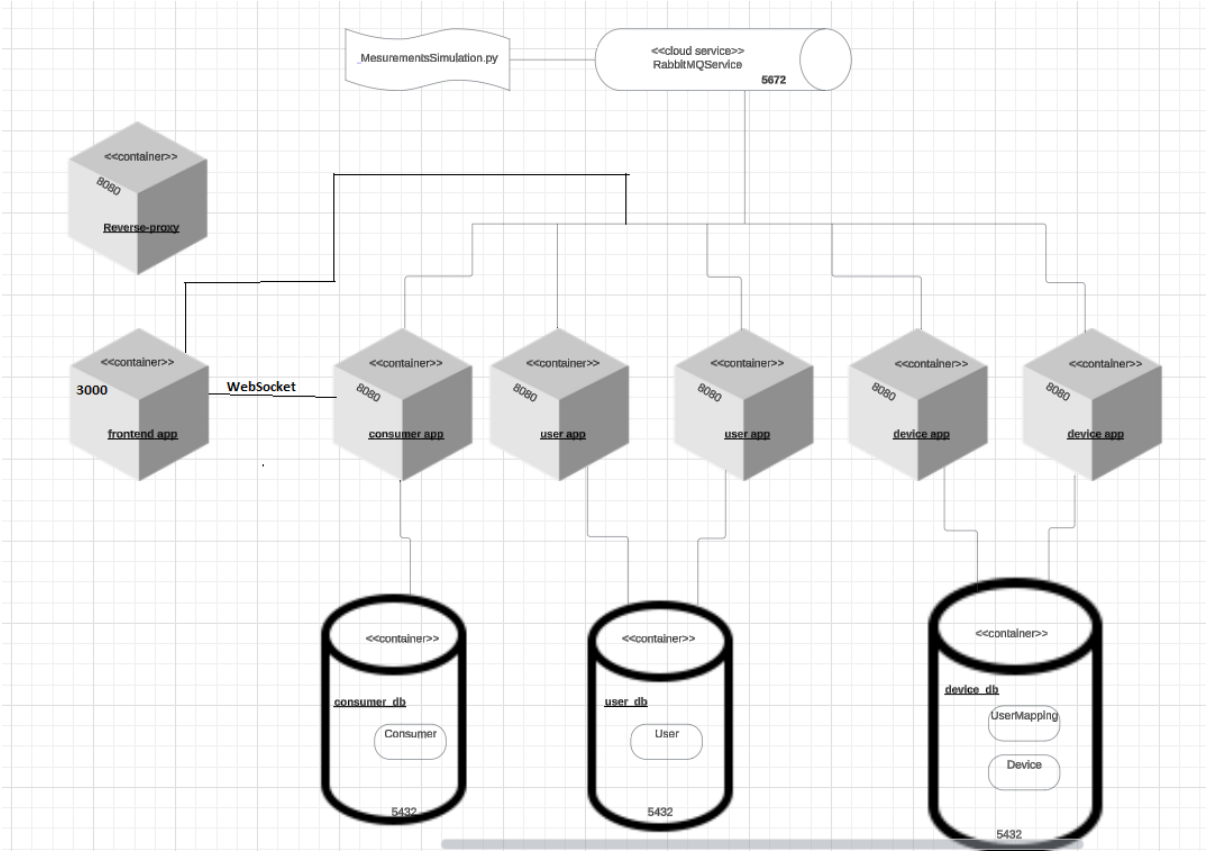
- Transacții: Când un utilizator este adăugat sau șters, microserviciul de utilizatori va trimite un mesaj în RabbitMQ pentru a actualiza dispozitivele asociate în microserviciul de dispozitive.

- Dacă există erori în timpul acestor operațiuni, tranzacțiile sunt anulate și datele sunt sincronizate în mod corect.

Comunicare WebSocket:

- Microserviciul de monitorizare trimite notificări în timp real la frontend atunci când consumul unui dispozitiv depășește un anumit prag (de exemplu, consumul maxim orar). Aceste notificări sunt trimise prin WebSocket, iar frontend-ul le afișează în timp real.

3.2 Diagrama de Deployment



Sistemul este dezvoltat folosind containere Docker, fiecare componentă rulând într-un container separat. Docker Compose este utilizat pentru a orchestra containerele și pentru a asigura comunicarea între microservicii.

Containerele sistemului:

1. User DB (PostgreSQL): Baza de date pentru utilizatori, care stochează informațiile despre utilizatori. Rulează pe portul 5432.
2. Device DB (PostgreSQL): Baza de date pentru dispozitive, care stochează informațiile despre dispozitive și asocierea acestora cu utilizatorii. Rulează pe portul 5432.
3. Consumer DB (PostgreSQL): Baza de date pentru consumul de energie al dispozitivelor. Rulează pe portul 5432.
4. User Microservice (Spring Boot): Microserviciul care gestionează utilizatorii și interacționează cu User DB. Rulează pe portul 8080.
5. Device Microservice (Spring Boot): Microserviciul care gestionează dispozitivele și interacționează cu Device DB. Rulează pe portul 8080.
6. Consumer Microservice (Spring Boot): Microserviciul care gestionează consumul de energie al dispozitivelor și interacționează cu Consumer DB. Rulează pe portul 8080.
7. Frontend (React + Nginx): Aplicația frontend care interacționează cu backend-ul și afișează datele utilizatorilor. Rulează pe portul 3000.
8. RabbitMQ: Sistemul de mesagerie asincronă care gestionează cozi pentru comunicarea între microservicii și pentru comunicarea dintre simulatorul de Python și microserviciul de monitorizare. Rulează pe portul 5672.

9. Traefik: Reverse proxy și load balancer, care distribuie traficul între replicile microserviciilor de utilizator și dispozitiv, asigurând scalabilitate și accesibilitate. Rulează pe portul 8082.

Cum se face interacțiunea între microservicii:

- Microserviciile Spring Boot sunt accesibile din exterior prin intermediul Traefik, care distribuie cererile HTTP între replicile serviciilor.
- RabbitMQ gestionează cozi pentru mesaje între microserviciile de dispozitive și monitorizare, asigurând o comunicare asincronă eficientă.
- Frontend-ul se conectează la microserviciile backend pentru a obține datele utilizatorilor și dispozitivelor folosind request-reply HTTP. Comunicarea în timp real (notificările de depășire a pragurilor de consum energetic) se face prin WebSocket.

4. Implementare

4.1 Instrumente Utilizate

1. Spring Boot pentru crearea microserviciilor

Spring Boot este un framework open-source bazat pe Java, utilizat pentru crearea de aplicații enterprise, scalabile și robuste. În contextul acestui proiect, Spring Boot este folosit pentru a crea microservicii RESTful, care formează backend-ul aplicației de gestionare a energiei.

Principalele caracteristici utilizate:

- Spring Web: pentru a expune microservicii RESTful.
- Spring Data JPA: pentru integrarea cu baza de date PostgreSQL.
- Spring Security: pentru implementarea securității aplicației, inclusiv autentificare și autorizare.
- Spring Boot Actuator: pentru monitorizarea și gestionarea aplicației.

2. RabbitMQ pentru comunicarea asincronă între microservicii

RabbitMQ este un sistem de mesagerie de tip message broker care permite comunicarea asincronă între componentele unui sistem distribuit. În acest proiect, RabbitMQ este folosit pentru a facilita comunicarea între microserviciile de tip producer și consumer.

Cum este utilizat RabbitMQ în acest proiect:

- Microserviciul de Monitorizare primește datele de consum de la microserviciul de Device prin intermediul RabbitMQ.
- În cazul în care consumul unui dispozitiv depășește un prag definit, un mesaj va fi trimis prin RabbitMQ, iar clientul va primi o notificare în timp real.

3. PostgreSQL pentru persistarea datelor

PostgreSQL este o bază de date relațională open-source, care oferă performanță, fiabilitate și scalabilitate pentru stocarea datelor aplicației.

Cum este utilizat PostgreSQL în acest proiect:

- Baza de date pentru utilizatori: Stochează informațiile utilizatorilor și asocierile lor cu dispozitivele.
- Baza de date pentru dispozitive: Conține informații despre dispozitivele care măsoară consumul de energie, inclusiv parametrii de configurare, cum ar fi consumul maxim acceptat.
- Baza de date pentru consum: Înregistrează datele de consum de energie pentru fiecare dispozitiv pe măsură ce sunt colectate.

4. ReactJS pentru dezvoltarea frontend-ului

ReactJS este o bibliotecă JavaScript open-source pentru construirea de interfețe de utilizator (UI). Este folosită pentru a crea aplicații web dinamice și interactive, axate pe performanță și reutilizarea componentelor.

Cum este utilizat ReactJS în acest proiect:

- Interfața utilizatorului (UI): ReactJS permite utilizatorilor să vizualizeze în timp real consumul de energie al dispozitivelor, să acceseze istoricul acestora și să primească notificări atunci când se depășesc pragurile de consum.
- Componente interactive: Dashboard-ul interactiv va permite utilizatorilor să filtreze datele și să vizualizeze grafice despre consumul de energie al dispozitivelor lor.

5. Docker pentru containerele aplicației și orchestrarea acestora cu Docker Compose
Docker este o platformă pentru crearea, testarea și rularea aplicațiilor în containere, care sunt medii izolate ce conțin toate componentele necesare pentru a rula aplicația (de exemplu, biblioteci, dependențe, configurări).
Cum este utilizat Docker în acest proiect:

- Containere pentru microservicii: Fiecare microserviciu (User, Device, Monitoring & Communication) este izolat într-un container Docker, asigurându-se astfel că toate componentele pot fi gestionate și rulate independent.
- Orchestrarea cu Docker Compose: Docker Compose este utilizat pentru a coordona containerele, permitând rularea și gestionarea mai multor microservicii, baze de date și alte componente necesare pentru aplicație.

6. Traefik pentru load balancing și reverse proxy
Traefik este un proxy de aplicație modern, care se integrează ușor cu Docker și permite distribuirea traficului între mai multe instanțe ale microserviciilor (load balancing).
Cum este utilizat Traefik în acest proiect:

- Load balancing: Traefik va distribui traficul de rețea între microserviciile de User și Device, asigurându-se că cererile sunt distribuite eficient și aplicația rămâne disponibilă chiar și în condiții de trafic intens.
- Reverse proxy: Traefik va redirecționa cererile către diferitele

4.2 Fișier README

Aplicația finală poate fi rulată în Docker urmând pașii de mai jos:
1. Clonează repository-ul proiectului:
   git clone https://gitlab.com/budaandreea02/ds2024_30244_buda_andreea_assignment_2

2. Accesează directorul proiectului:
   Deschide un terminal și navighează în directorul unde ai clonat proiectul

3. Asigură-te că Docker Desktop este pornit:
   Verifică dacă Docker Desktop rulează pe mașina ta.

4. Rulează comenzile pentru a construi și a porni containerele Docker:
   În terminal, rulează următoarea comandă pentru a construi și a porni containerele folosind Docker Compose:
   docker-compose up --d --build

5. Inițializează baza de date:
   Execută comenzile pentru a popula bazele de date cu datele inițiale. Aceste scripturi vor insera utilizatori, dispozitive și informații despre consum în baza de date PostgreSQL:
   cat .\Scripts\insertUserDB.sql | docker exec -i tema2sd-user_db-1 psql -U postgres -d user_db
   cat .\Scripts\insertDeviceDB.sql | docker exec -i tema2sd-device_db-1 psql -U postgres -d device_db
   cat .\Scripts\selectConsumerDB.sql | docker exec -i tema2sd-consumer_db-1 psql -U postgres -d consumer_db

6. Instalează pachetul Python pika:
   Asigură-te că ai instalat pachetul necesar pentru a interacționa cu RabbitMQ. În terminal, rulează:
   python -m pip install pika --upgrade

7. Accesează aplicația:
   După ce containerele sunt pornite, accesează aplicația în browser la următoarea adresă:
   http://reactapp.localhost

8. Vizualizează funcționalitatea load balancer-ului:
   Accesează Traefik, care este configurat pentru a distribui traficul între microserviciile de backend, la adresa:
   http://localhost:8082

9. Rulează simulatorul de date pentru consumul energetic:
   Pentru a simula și trimite datele către RabbitMQ, folosește următoarea comandă Python:
   python .\MesurementsSimulation.py