Assignment 3

Energy Management System

Student: Marginean Teodor Ioan

Group: 30644

- Arhitectura conceptuala

Arhitectura sistemului este bazată pe microservicii, cu două microservicii principale:

Microserviciul de gestionare a utilizatorilor este responsabil pentru gestionarea operațiunilor CRUD asupra utilizatorilor, inclusiv autentificarea și înregistrarea.

Microserviciul de gestionare a dispozitivelor este responsabil pentru gestionarea operațiunilor CRUD asupra dispozitivelor, inclusiv alocarea și dezalocarea dispozitivelor către utilizatori.

Schema bazei de date

Microserviciul de gestionare a utilizatorilor folosește baza de date a utilizatorilor.

Microserviciul de gestionare a dispozitivelor folosește o bază de date proprie, care conține două tabele:

deviceType: reprezintă categoria generală a dispozitivelor adăugate în sistem.

deviceInstance: reprezintă instanțele fizice reale ale tipurilor de dispozitive.

Backend-ul

Backend-ul pentru ambele microservicii este structurat într-o arhitectură stratificată:
Stratul de date conține constante, enumerări, entități și repozitorii.
Stratul de servicii implementează logica de afaceri.
Stratul de controlori este responsabil pentru procesarea solicitărilor HTTP.

Stiva de tehnologii

Backend: Spring Boot, Spring Security, OpenAPI, Liquibase
Frontend: Angular, Bootstrap

Observații

Microserviciul de gestionare a dispozitivelor folosește o soluție simplă pentru autorizare, care verifică doar reclamația suplimentară de rol din tokenul JWT. O implementare mai robustă ar folosi o poartă API pentru a gestiona toate autorizațiile necesare. Microserviciul de gestionare a dispozitivelor folosește scripturi Liquibase pentru a crea bazele de date. Acest lucru este recomandat pentru medii de producție, unde fiecare actualizare a structurii bazei de date este realizată prin intermediul schimbărilor.

Concluzie

Arhitectura sistemului este bazată pe microservicii, care oferă o serie de avantaje, inclusiv performanță mai bună, scalabilitate îmbunătățită și o gestionare mai bună a resurselor. De asemenea, dezvoltarea fiecărui microserviciu poate fi realizată independent de către echipe diferite, asigurând o dezvoltare mai rapidă și cod mai bine scris.

Modificări propuse

Implementarea unei soluții mai robuste pentru autorizare în microserviciul de gestionare a dispozitivelor. O soluție posibilă ar fi utilizarea unei porți API pentru a gestiona toate autorizațiile necesare.
Adăugarea de un mecanism de recuperare a parolelor pierdute sau uitate. Acest lucru ar îmbunătăți experiența utilizatorului și ar reduce riscul de conturi compromise.

Implementarea unui sistem de monitorizare și alertă. Acest lucru ar ajuta la identificarea și rezolvarea problemelor înainte ca acestea să afecteze utilizatorii.

Explicații

Implementarea unei soluții mai robuste pentru autorizare ar îmbunătăți securitatea sistemului. O soluție simplă, cum ar fi verificarea reclamației suplimentare de rol din tokenul JWT, nu este suficientă pentru a proteja sistemul împotriva atacurilor. O soluție mai robustă ar folosi o poartă API pentru a gestiona toate autorizațiile necesare. Acest lucru ar asigura că doar utilizatorii autorizați pot efectua acțiuni asupra dispozitivelor.

Adăugarea de un mecanism de recuperare a parolelor pierdute sau uitate ar îmbunătăți experiența utilizatorului. Acest lucru ar permite utilizatorilor să își reseteze parolele în cazul în care le uită sau le pierd.
Implementarea unui sistem de monitorizare și alertă ar ajuta la identificarea și rezolvarea problemelor înainte ca acestea să afecteze utilizatorii. Acest lucru ar include monitorizarea performanței sistemului, detectarea defecțiunilor și alertarea utilizatorilor în cazul unor probleme.

Aceste modificări propuse ar îmbunătăți funcționalitatea și securitatea sistemului.

RabbitMQ

RabbitMQ în cloud oferă o soluție robustă pentru gestionarea comunicărilor asincrone în aplicațiile Spring Boot. În această arhitectură, RabbitMQ acționează ca un broker de mesaje, gestionând transmiterea de informații între diferite părți ale sistemului. Prin crearea cozi de mesaje și schimburi (exchanges), RabbitMQ permite separarea eficientă a procesării și transmiterii datelor, reducând cuplarea dintre diferite componente ale aplicației.

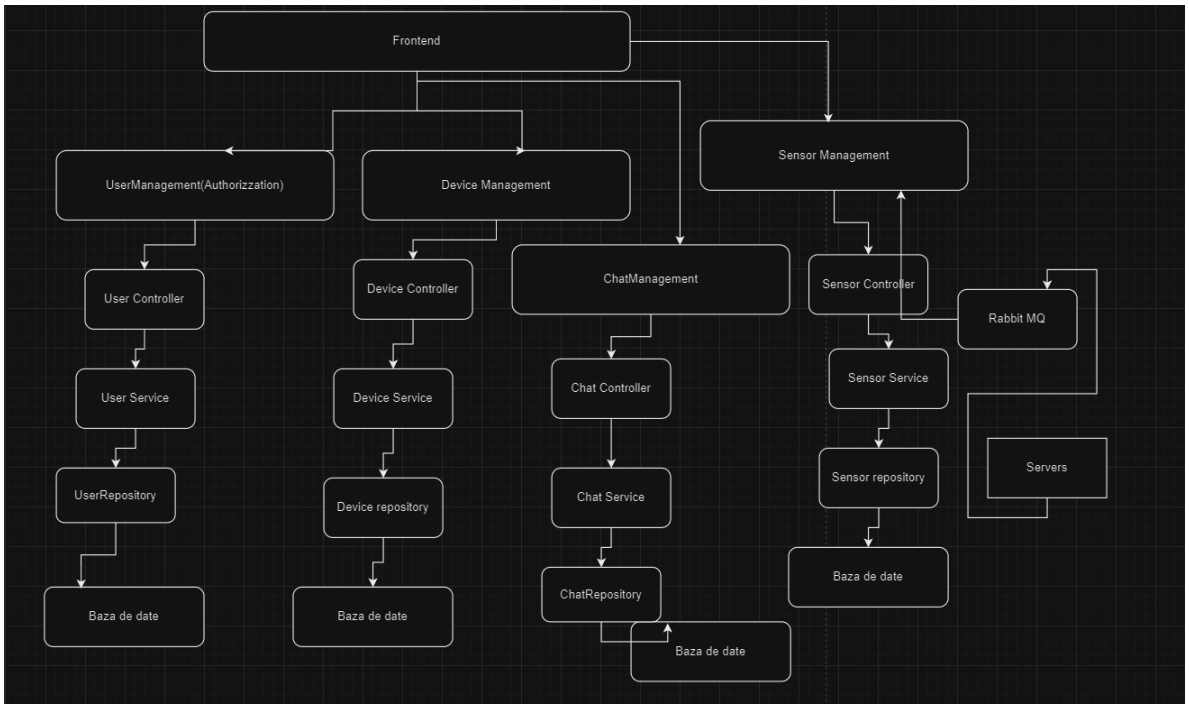
Integrarea cu Spring Boot se realizează prin adăugarea dependențelor necesare și configurarea conexiunii la serverul RabbitMQ din cloud. Aplicația poate apoi defini producători (producers) și consumatori (consumers) de mesaje, facilitând un flux de lucru bazat pe evenimente. Acest model asigură un grad ridicat de scalabilitate și eficiență, permițând aplicației să proceseze cereri într-un mod distribuit și să răspundă rapid la schimbările de stare sau la cererile utilizatorilor.

În plus, utilizarea RabbitMQ în cloud aduce avantajul unei infrastructuri gestionate, cu un grad înalt de disponibilitate și securitate, esențiale pentru aplicațiile enterprise.

Websocket

WebSockets reprezintă un standard avansat de comunicație pe Internet, care permite deschiderea unei conexiuni interactive între un browser al utilizatorului și un server. Această tehnologie facilitează un schimb bidirecțional și în timp real de date, ceea ce este ideal pentru aplicații web care necesită actualizări rapide și constante, cum ar fi jocurile online, chat-urile și aplicațiile de tranzacționare. Spre deosebire de modelul tradițional HTTP, unde clientul inițiază o cerere și serverul răspunde, WebSockets menține o conexiune deschisă, permițând ambelor părți să trimită date în orice moment.

Adăugând la aceasta, WebSockets sunt deosebit de eficiente în gestionarea traficului de date datorită overhead-ului redus. După stabilirea inițială a conexiunii WebSocket, datele pot fi trimise și primite fără a fi necesar să se reia întregul protocol de handshake, cum este cazul în HTTP. Acest lucru reduce semnificativ latența, făcând WebSocket ideal pentru aplicații sensibile la întârziere, precum jocurile multiplayer în timp real sau aplicațiile de tranzacționare în timp real. În plus, suportul pentru mesaje binare și text înseamnă că WebSockets pot fi utilizate pentru o varietate largă de tipuri de date, sporind versatilitatea acestora în diverse scenarii de utilizare. Totuși, este important de reținut că securitatea WebSockets trebuie gestionată cu atenție, deoarece conexiunile persist



- Deployment

Pentru partea de implementare, am folosit Docker. Pentru implementarea microserviciilor din spate, mai întâi am generat fișierele JAR. Apoi, am creat fișierele Docker.

Fișierul Docker începe prin specificarea unei imagini Docker de bază care conține sistemul de operare minim necesar, mediul de rulare necesar aplicației Java și include un kit de dezvoltare Java compatibil (JDK-17 în acest caz). Această imaginație de bază asigură că containerul are tot ce este necesar pentru a rula aplicația Java într-un mediu izolat.

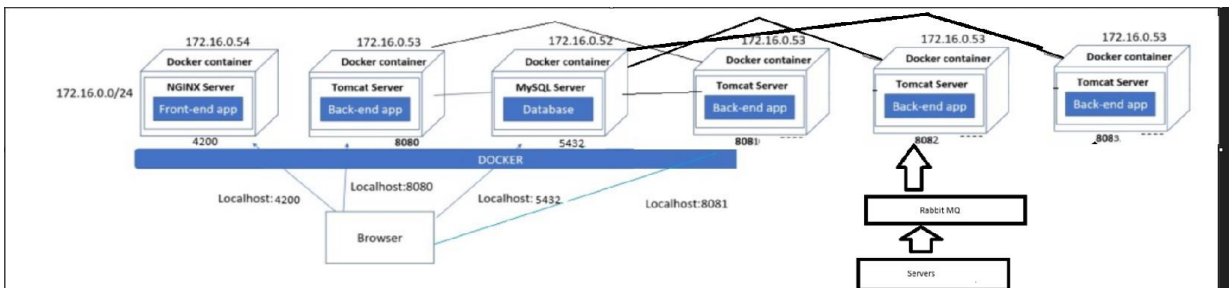
În continuare, am utilizat comenzile COPY și ADD pentru a încorpora în imaginea Docker sursa aplicației, fișierul pom.xml, instrumentul de verificare a codului și fișierul JAR rezultat din compilarea codului sursă. Aceste comenzi permit transferul acestor fișiere în interiorul containerului, astfel încât aplicația noastră să poată fi instalată și executată corect.

De asemenea, am specificat directorul de lucru cu ajutorul comenzii WORKDIR. Acest director este locul în care fișierul JAR va fi plasat în cadrul containerului, și de asemenea, este directorul în care containerul va executa toate comenzile ulterioare. Astfel, am stabilit mediul de lucru corect pentru container.

În ceea ce privește comanda ENTRYPOINT, aceasta specifică comanda care trebuie executată atunci când se creează un container din imaginea noastră. În cazul nostru, comanda ENTRYPOINT rulează fișierul JAR, adică lansarea aplicației Java. Prin acest pas, am definit modul în care containerul nostru va rula aplicația noastră Java.

De asemenea, am configurat variabile de mediu pentru a furniza informații despre conexiunea la baza de date, cum ar fi numele de utilizator, parola și portul. Aceste variabile de mediu sunt folosite de aplicație pentru a ști cum să se conecteze la baza de date, permițând astfel configurarea flexibilă a conexiunii.

După ce am creat fișierul Docker, am utilizat comanda "build" pentru a crea imaginea Docker pe baza acestuia. Această imagine conține toate elementele necesare pentru a rula aplicația noastră Java într-un container Docker. Ulterior, am integrat această imagine în fișierul docker-compose și am inclus-o într-un container dedicat pentru fiecare microserviciu. Astfel, am creat două aplicații cu containere multiple: una pentru gestionarea utilizatorilor și alta pentru gestionarea dispozitivelor. Fiecare aplicație conține un container pentru aplicația Spring, care utilizează imaginea creată, și un container pentru baza de date PostgreSQL, care utilizează o imagine de bază PostgreSQL. Această arhitectură ne permite să izolăm și să scalăm fiecare componentă a sistemului nostru în mod eficient.



README

Introducere

Acest proiect este o simulare a unei pagini web pentru un sistem de gestionare a energiei. Acesta utilizează tehnologiile Java Spring, Angular și PostgreSQL.

Tipuri de utilizatori

Există două tipuri de utilizatori: client și administrator. Administratorul poate efectua operațiuni CRUD pe baze de date și poate asocia dispozitive clienților. Clientul poate vizualiza dispozitivele sale.

Construire și rulare în IDE

Pentru a rula proiectul într-un IDE, trebuie să creați două baze de date PostgreSQL și să configurați fișierele application.properties. Apoi, puteți rula proiectul făcând clic pe triunghiul verde din dreapta sus.

Construire și rulare în Docker

Pentru a rula proiectul în Docker, trebuie să descărcați, instalați și creați un cont pe site-ul Docker. Apoi, puteți rula proiectul făcând clic pe butonul Run din Docker Desktop.

Navigare pe pagina web

Pagina web poate fi accesată la adresa http://localhost:4200/login.