**FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE**

**DEPARTAMENTUL CALCULATOARE**

*Disciplina Sisteme Informatice Distribuite*

*An academic : 2023 – 2024*

**Web sockets and security**

*Holuță Maria Antonia*

*Grupa 30641*

1. A screenshot of a computer

   Description automatically generatedDiagrama Conceptuală a Sistemului Distribuit

Cerinta:

Dezvoltarea unui Microserviciu de Chat și a unui Component de Autorizare pentru Sistemul de Management al Energiei.

Microserviciu de Chat

Utilizatorii pot comunica cu administratorul sistemului prin intermediul unei casete de chat afișate de aplicația front-end.

Mesajele sunt trimise asincron către administrator, care le primește împreună cu identificatorul utilizatorului, putând iniția un chat cu acesta.

În cadrul unei sesiuni de chat, mesajele pot fi schimbate între utilizator și administrator.

Administratorul poate purta conversații simultane cu mai mulți utilizatori.

Utilizatorii primesc notificări atunci când administratorul citește mesajele și invers.

Component de Autorizare

Unul dintre servicii este ales ca server de autorizare (am ales in User, implementat pentru autorizare și autentificare).

Acest serviciu generează token-uri de acces pentru aplicația client, care vor fi utilizate pentru accesarea altor microservicii.

Tehnologii de Implementare

Pentru componenta de chat: tehnologia web sockets.

Pentru componenta de autorizare: autorizare bazată pe JWT (JSON Web Tokens) pentru autentificarea și autorizarea utilizatorilor în toate microserviciile.

Tema este o continuare a temei 1 si 2 ce consta într-un frontend și două microservicii proiectate pentru a gestiona utilizatorii și dispozitivele lor de măsurare a energiei inteligente asociate. Sistemul poate fi accesat de două tipuri de utilizatori după un proces de autentificare: administrator și clienți. Administratorul poate efectua operațiuni CRUD (Creare-Citire-Actualizare-Ștergere) asupra conturilor utilizatorilor (definite prin ID, nume, rol: admin/client), dispozitivelor de măsurare a energiei inteligente (definite prin ID, descriere, adresă, consum maxim de energie pe oră) și asupra asignării utilizatorilor la dispozitive (fiecare utilizator poate deține unul sau mai multe dispozitive inteligente). Tema 2 implica Microserviciul bazat pe un middleware pentru brokerul de mesaje, care aduna date de la dispozitivele de masurare inteligente, proceseaza datele pentru a calcula consumul de energie pe ora si le stocheaza in baza de date a Microserviciului de Monitorizare si Comunicare.

Sincronizarea intre bazele de date ale Microserviciului de Gestionare a Dispozitivelor si noul Microserviciu de Monitorizare si Comunicare se face printr-un sistem bazat pe evenimente care utilizeaza un topic pentru schimbarile dispozitivelor (trimite informatii despre dispozitiv printr-o coada catre Microserviciul de Monitorizare si Comunicare).

O aplicatie Simulator de Dispozitiv de Masurare Inteligent va fi implementata ca Producator de Mesaje. Aceasta va simula un contor inteligent prin citirea datelor de energie dintr-un fisier sensor.csv (adica o valoare la fiecare 10 minute) si trimite date sub forma <timestamp, device\_id, measurement\_value> catre Brokerul de Mesaje (adica o coada). Timestamp-ul este preluat de la ceasul local, iar device\_id este unic pentru fiecare instanta a Simulatorului de Dispozitiv de Masurare Inteligent si corespunde device\_id-ului unui utilizator din baza de date (asa cum este definit in Asignarea 1). Simulatorul de dispozitive trebuie dezvoltat ca o aplicatie independenta (adica o aplicatie desktop). Fisierul sensor.csv poate fi descarcat de la https://dsrl.eu/courses/sd/materials/sensor.csv.

Masuratorile sunt trimise la coada folosind urmatorul format JSON:

{

"timestamp": 1570654800000,

"device\_id": "5c2494a3-1140-4c7a-991a-a1a2561c6bc2",

"measurement\_value": 0.1

}

Microserviciul de Monitorizare si Comunicare va avea un component Consumator de Mesaje care va procesa masuratorile pentru a calcula consumul total de energie pe ora si il va stoca in baza de date. Daca consumul total calculat pe ora depaseste valoarea maxima definita pentru dispozitiv (asa cum este definit in Asignarea 1), notifica asincron utilizatorul pe interfata sa web.

Această aplicație este construită folosind Spring pentru ambele backend-uri: unul pentru gestionarea dispozitivelor (Devices) și celălalt pentru gestionarea utilizatorilor (Users). Aceste două backend-uri sunt conectate la două baze de date diferite, una pentru stocarea datelor despre utilizatori și alta pentru stocarea datelor despre dispozitive. Frontend-ul este dezvoltat folosind React cu TypeScript (TSX), care aduce un avantaj semnificativ față de JavaScript (JS) prin introducerea verificărilor de tip la nivel de compilare, ceea ce face codul mai sigur și mai ușor de întreținut.

Avantajul TypeScript față de JavaScript constă în:

* Verificarea tipurilor la nivel de compilare: TypeScript detectează erorile de tip în timpul dezvoltării, ceea ce înseamnă că poți identifica și corecta erorile legate de tipuri înainte ca aplicația să ruleze. Aceasta face ca codul să fie mai sigur și mai robust.
* IntelliSense îmbunătățit: TypeScript oferă sugestii inteligente (IntelliSense) în timpul dezvoltării, ceea ce simplifică scrierea codului și reduce riscul de a face erori.
* Documentare îmbunătățită: TypeScript permite adăugarea de tipuri și documentație direct în cod, ceea ce face mai ușor pentru dezvoltatori să înțeleagă și să folosească API-urile.

Arhitectura aplicației folosește o abordare pe layere cu Spring, și include următoarele componente:

* Entities: Acestea reprezintă obiectele de bază ale aplicației, cum ar fi User și Device. Aceste entități sunt mapate direct pe bazele de date corespunzătoare.
* Repository (Repo): Repository-urile Spring Data JPA (JpaRepository) sunt utilizate pentru a interacționa cu bazele de date. Există un repository pentru fiecare entitate pentru a efectua operațiuni CRUD asupra datelor.
* Service: Serviciile conțin logica de afaceri a aplicației. Acestea realizează operațiuni precum validarea datelor, gestionarea tranzacțiilor și coordonarea între repository-uri. Având două backend-uri separate (unul pentru utilizatori și unul pentru dispozitive), este posibil să existe două straturi de servicii separate.
* DTO (Data Transfer Objects): Aceste obiecte sunt utilizate pentru a transfera date între frontend și backend. Ele permit o separare clară între datele de afișare și cele de stocare.
* Controller: Controller-urile gestionează cererile primite de la frontend și dirijează solicitările către servicii corespunzătoare. Controller-urile sunt, de obicei, responsabile de gestionarea autentificării și autorizării.

Builder: Builder-urile pot fi utilizate pentru a transforma date între obiecte DTO și entități. Acestea sunt utile pentru a gestiona conversiile între structurile de date necesare pentru frontend și cele stocate în baza de date.

Rolurile utilizatorilor în această aplicație sunt definite după cum urmează:

* Rolul Administrator/Manager:

Poate efectua operații CRUD asupra utilizatorilor.

Poate efectua operații CRUD asupra dispozitivelor.

Poate crea asignări între utilizatori și dispozitive.

Rolul User/Client:

* Un client poate vizualiza pe pagina sa toate dispozitivele.

Se asigură că utilizatorii cu un anumit rol nu pot accesa paginile corespunzătoare altor roluri, astfel încât funcționalitatea să fie restricționată în funcție de tipul de utilizator autentificat.

Cu ajutorul frontendului, se vor apela api-urile specifice, iar la devices va știi să caute useriii în funcție de id-ul lor, apoi va apela endpoint-ul de la User pentru a afișa numele asociat cu id-ul.

S-a adăugat un al treilea microserviciu de spring care să gestioneze tot într-o altă bază de date informațiile transmise de rabbitmq prin queue folosind serviciul de cloud <https://sparrow.rmq.cloudamqp.com/>.

În proprieties, pe lângă detaliile de conectare la baza de date, s-au folosit datele oferite de acestea:

spring.rabbitmq.ssl.enabled=false

spring.rabbitmq.host=sparrow-01.rmq.cloudamqp.com

spring.rabbitmq.port=5672

spring.rabbitmq.username=pass

spring.rabbitmq.password=pass

spring.rabbitmq.virtual-host=host

În RabbitMQConfig s-au configurat datele necesare acestei conexiuni.

Într-un alt proiect am creaat Producerul care pune in coada datele citite din fisier cu aceasta conexiune prin sendDataToBroker ce trimite o data la 10 secunde valorile.

Apoi în Consumer, am avut nevoie de un RabbitListener care primea datele din coadă și le punea în clasa entitate Monitoring, apoi apela o funcție ce verifica dacă measurements\_value trecea de valoare maximă din Device db. Acest lucru a fost posibil datorită unui WebClient ce apela endpointuirle de la device necesare pentru analiza valorilor.

Apoi s-a creat metoda de a permite notificarile si a le trimite printr-un websocket trimis catre client. În config se va găsi de asemenea configurările pentru WebSocket. În Frontend am folosit

const socket = new SockJS("http://localhost:8082/spring-demo/ws");

const stompClient = Stomp.over(socket);

apoi am definit prin stompClient.subscriber ws la care sa fie atent pentru notificarile date la destinatia stabilita.

In User s-a implementat generarea de tokenuri, validarea si register/login, apoi celelalte microservicii preiau tokenul si extrag informatiile userului din el.

1. A screenshot of a computer

   Description automatically generatedDiagrama UML Deployment

Pentru porturile de la backend s-au păstrat cele inițiale de 8080 și 8081 și pentru React frontend 3000. La baza de date mysql de pe 3306 este setată la 3309, iar la postgres, 5432 si 5433 la 5437 si 5438.

1. Readme build și execuție

Proiectul este structurat în două backend-uri separate (unul pentru dispozitive și unul pentru utilizatori) și un frontend dezvoltat în React cu TypeScript. În plus, folosesc baze de date separate pentru a stoca datele. Iată pașii pentru a porni proiectul:

Pentru rulare cu docker:

* docker build -t t3-react-frontend in folderul Frontend
* docker build -t t3-spring-backend-1 in folderul Backend1 (User)
* docker build -t t3-spring-backend-2 in folderul Backend2 (Device)
* docker build -t t3-spring-backend-3 in folderul Backend3 (Monitoring)
* docker build -t t3-spring-backend-4 in folderul Backend4 (Chat)
* docker-compose up

**Pas 1: Configurare baze de date**

Se instalează PostgreSQL și MySQL pe sistem.

După instalare, se creează două baze de date separate în PostgreSQL și MySQL.

- Pentru PostgreSQL:

CREATE DATABASE postgres

WITH

OWNER = postgres

ENCODING = 'UTF8'

LC\_COLLATE = 'English\_United States.1252'

LC\_CTYPE = 'English\_United States.1252'

TABLESPACE = pg\_default

CONNECTION LIMIT = -1

IS\_TEMPLATE = False;

COMMENT ON DATABASE postgres

IS 'default administrative connection database';

- Pentru MySQL:

CREATE DATABASE IF NOT EXISTS Energy\_Management;

USE Energy\_Management;

**Pas 2: Backend-ul Utilizatorilor**

Se deschide backendul pentru utilizatori (Backend1) într-un alt proiect IntelliJ IDEA.

Se configurează sursa de date a bazei de date MySQL în application.properties cu numele și parola aferente bazei de date.

Se rulează aplicația Spring Boot pentru backendul utilizatorilor. Acesta va crea automat tabelele necesare în baza de date MySQL.

**Pas 3: Backend-ul Dispozitivelor**

Se deschide backendul pentru dispozitive (Backend2) în IntelliJ IDEA.

Se configurează sursa de date a bazei de date PostgreSQL în application.properties cu numele și parola aferente bazei de date.

Se rulează aplicația Spring Boot pentru backendul dispozitivelor. Acesta va crea automat tabelele necesare în baza de date PostgreSQL

**Pas 4: Backend-ul Monitoring**

Se deschide backendul (Backend3) în IntelliJ IDEA.

Se configurează sursa de date a bazei de date PostgreSQL în application.properties cu numele și parola aferente bazei de date.

Se rulează aplicația Spring Boot pentru backendul dispozitivelor. Acesta va crea automat tabelele necesare în baza de date PostgreSQL

**Pas 5: Producer**

Se deschide aplicatia Producer si se introduc datele configurarii rabbitmq, apoi se ruleaza cu exemplul din sensor.csv sau se citeste din alt fisier.

**Pas 6: Frontend**

Deschide frontend-ul într-o fereastră terminală separată sau într-un IDE web, cum ar fi Visual Studio Code.

(Navighează în directorul frontend folosind comanda cd frontend).

Instalează dependințele frontend-ului folosind comanda npm install.

După instalare, rulează frontend-ul cu comanda npm start.

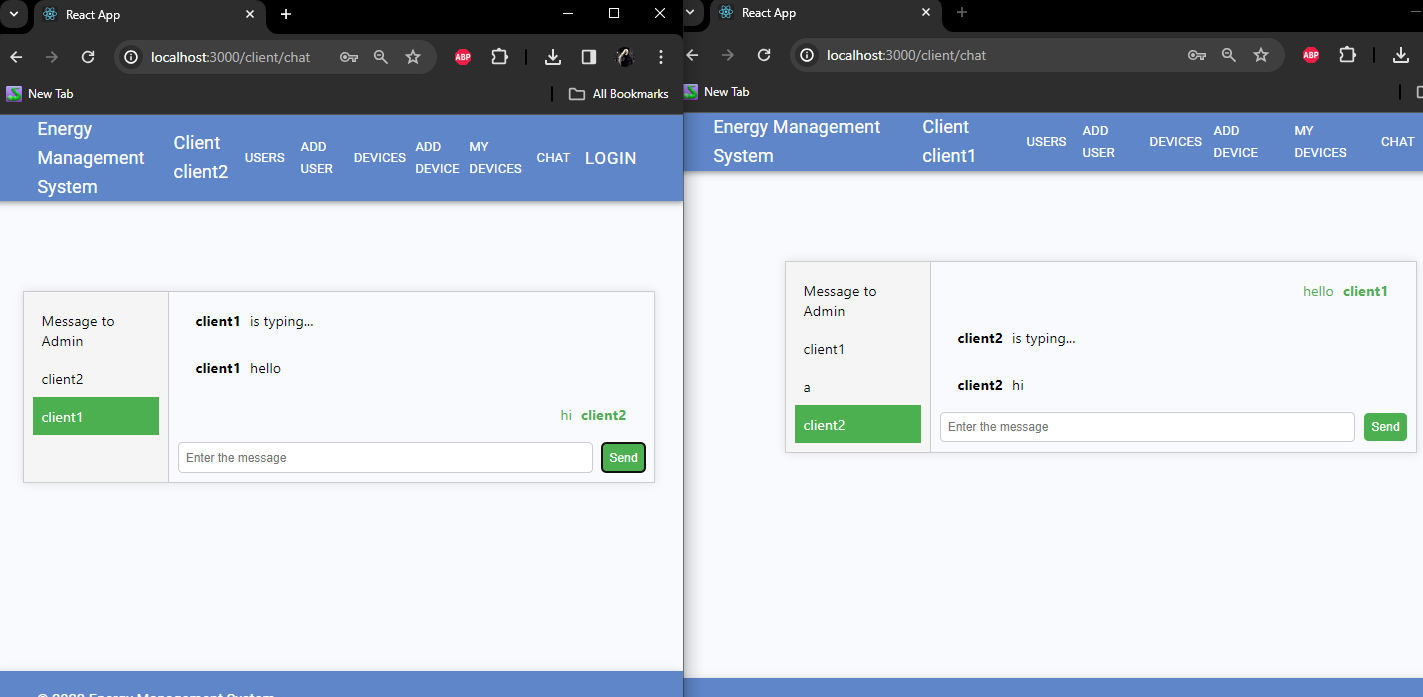
**Pas 5: Accesarea aplicației**

Din frontend, în meniul de navigare se alege login pentru autentificare.

Administratorii vor putea accesa din navigare Users, Add User, Device sau Add Device, iar Clientii doar My Devices.

Numele userului asignat deviceurilor se poate vedea din Devices, User Assigned

Pentru update se alege icon-ul creion din dreapta, pentru stergere cosul de gunoi, iar pentru a adauga/sterge device-uri la useri se alege Users -> Manage Devices –> Add/Remove



A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated