MINISTRY OF EDUCATION
TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA
FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

ENERGY MANAGEMENT SYSTEM

SISTEME DISTRIBUITE

Numele studentului: Alina Aurică

An 4, TI, Gr. 30643

Profesor îndrumător: David Chiș

Contents

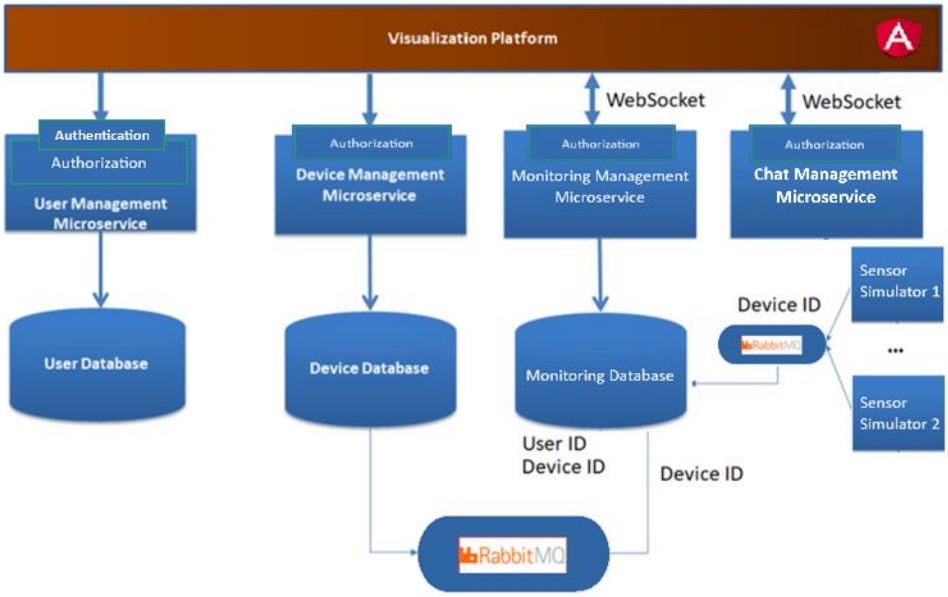
1. INTRODUCERE

Proiectul de față reprezintă o simulare a unei pagini web. Pagina web descrie un sistem asemănător celor de gestiune a energiei într-o locuință, sistem care poartă denumirea de Energy Management System. Am utilizat ca tehnologii: Java Spring pe partea de backend, Angular pe partea de frontend și PostgreSQL ca server de baze de date. Pentru comunicarea și transferul de date între microservicii, precum și între Sensor Simulator și Monitoring Service, am utilizat tehnologia RabbitMQ. Sistemul de notificari pentru depasirea consumului, precum și componenta de chat sunt realizate prin intermediul WebSocket-urilor.

Există 2 tipuri de useri: client și admin. Adminul poate realiza operații pe CRUD pe cele 2 baze de date, să asocieze unui device un client și să comunice cu clienții prin intermediul unui chat. Clientul poate să-și vizualizeze device-urile, să primească notificări când consumul unui divece asociat i-a fost depășit, să poată vedea graficele cu consumul pe oră (al fiecărui device) și să comunice cu admin-ul.

Pentru că securitatea este o parte importantă a acestui proiect, ea s-a realizat utilizând Spring Security și JWT Token, pentru a se crea un sistem de autentificare-autorizare pentru fiecare microserviciu în parte.

2. CONCEPTUAL ARCHITECTURE PENTRU SISTEMUL DISTRIBUIT:



Proiectul este format din 4 Microservicii: User Management MicroService, Device Management MicroService, Monitoring Management Microservice și Chat Management Microservice, ambele legate la 4 baze de date specifice (PostgreSQL database). Pe partea de backend am utilizat limbajul de programare Java, cu framework-ul REST Spring, iar pe partea de frontend am lucrat cu Angular (TypeScript, Html, CSS). Legătura dintre Monitoring Management Microservice și Device Management Microservice, precum și cea dintre Monitoring Managament Microservice și Sensor Simulator este realizată prin intermediul unor cozi de RabbitMQ. Sistemul trimite notificări clienților pentru depășirea consumului maxim pe ora prin intermediul unei conexiuni de tip WebSocket. De asemenea, componenta de Chat Management Microservice comunică cu frontend-ul tot prin conexiuni de tip WebSocket (pentru trimiterea și primirea de mesaje, pentru mesajul de Seen și cel de Typing).

PostgreSQL, cunoscut cel mai adesea ca Postgres, este o bază de date relațională open-source care poate suporta ca și limbaj relațional SQL.

Spring este un framework specific Java, care dezvoltă principiile REST:

1. Client-Server: emițătorul și receptorul sunt entități separate (comunică prin mesaje de tip request-response)
2. Stateless: fiecare cerere este o solicitare independentă
3. Cache: păstrează datele frecvent accesate
4. URL și URI
5. Protocol HTTP/HTTPS

Angular e bazat pe Typescript. De altfel, e un framework open-source pentru single-page web application.

Comunicarea dintre cele 2 microservicii de device și client s-a realizat prin crearea a 2 request-uri simultane care să actualizeze ambele baze de date când, spre exemplu, se șterge un client (automat se șterg și device-urile asociate acestuia). Cele 2 request-uri au fost realizate în Typescript-urile asociate paginilor din frontend.

Prin cozile de RabbitMQ, s-au trimis mesaje in format Json de tipul:

{

“timestamp": 1570654800000

“device_id”: “5c2494a3-1140-4c7a-991a-a1a2561c6bc2”

“measurement_value”: 0.1

}

pentru ceea ce transmite Senzor Simulator-ul. Mesajele transmise de la Device Managament conțin un Device și statusul acestuia: ADD, UPDATE sau DELETE. Mai există, de asemenea, o legătură între Device Management și Sensor Simulator, pentru realizarea scrierii în fișierul de configurare a device ID-urilor. Din fișierul de configurare (deviceConfigID.txt) preiau deviceID-urile pentru câmpul respectiv al senzorului. Dacă un device este șters din aplicație, acesta va fi șters și din fișierul de configurație. De asemenea, pot fi adăugate și șterse manual deviceID-uri.

RabbitMQ este un open-source message-broker software care a implementat inițial AMQP (Advanced Message Queuing Protocol) și de atunci a fost extins cu o plug-in architecture pentru a suporta STOMP (Streaming Text Oriented Messaging Protocol), MQTT (MQ Telemetry Transport) și alte protocoale.

WebSocket este un protocol de comunicații computerizat, care oferă canale de comunicație simultană în două sensuri printr-o singură conexiune TCP (Transmission Control Protocol).

Sistemul de trimitere al mesajelor are 3 componente:

1. Send Message: Adminul poate comunica cu mai mulți clienți, alegându-i dintr-o listă. Acest lucru este posibil datorita unicității canalului de comunicare admin-client, care este realizată prin adăugarea clientID-ului la topicul canalului. Mesajele se salvează în baza de date, pentru a se păstra un istoric, care apare atunci când se apasă pe Start Chat.
2. Seen Message: Mecanismul de seen al mesajelor presupune ca atunci când mesajul primit de user este randat, acesta să apeleze o metodă prin care să dea update la mesaj, iar variabila seen din backend devine true (se afișează în frontend Seen sub mesajul citit).

3. Typing Message: Pe input-ul unde se introduce mesajul, prin (input)="sendTypingMessage()" sesizez schimbările din input (typing-ul) și activez o metodă care trimite încrucisat notificări de typing prin canale cu topicuri unice (la fel ca la send și seen message). Pe frontend-ul de la client ascult topicul de la admin și invers.

Partea de Authorization o fost realizată utilizând Spring Security pe backend și Angular Guards pe partea de frontend și limitează accesul la pagini înainte de a te loga în aplicație sau dacă nu corespund cu rolul pe care îl ai: CLIENT sau ADMIN. De asemenea, encriptează informațiile esențiale ale unui user (ex: parola – ca și stocare, ex: role, ID și email – sub formă de token reținut în localStorage). Sistemul are incorporată și o parte de autorizare realizată tot prin Spring Security și JWT token (pe partea de User Management Microservice). JWT token-ul din acest microserviciu este salvat în localStorage. De acolo este preluat și trimis prin header pentru a fi decodificat în toate celelalte Microservicii în parte.

3. UML DEPLOYMENT DIAGRAM:

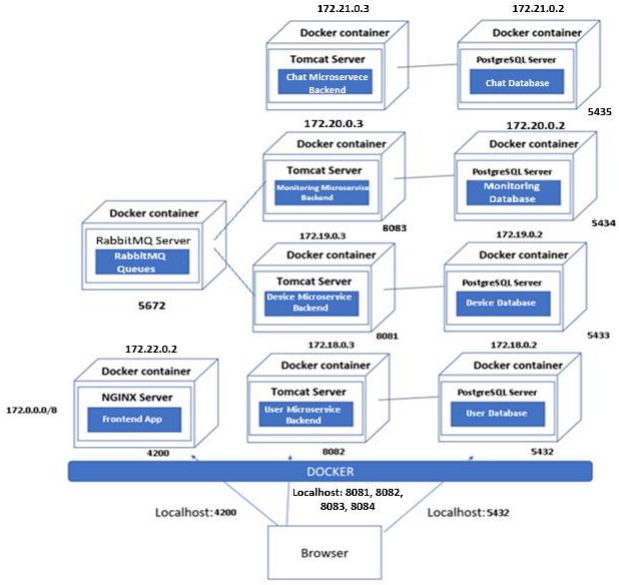


Diagrama de mai sus ilustrează modul în care funcționează componentele atunci când se realizează un exercițiu de deployment pe ele. Se poate observa că cele 4 microservicii și aplicația de frontend rulează pe subrețele diferite, iar baza de date și backend-ul pentru fiecare microserviciu se află în aceeași subrețea, doar că au IP-uri diferite pentru a putea transmite date de la unele la altele. Toate acestea trimit date în aceeași rețea-mamă, prin intermediul unui brindge (folosit pentru a comunica dintr-o subrețea în alta).

Browser-ul reprezintă partea de client a rețelei, Dockerul partea care izolează de exterior aplicația, iar aplicația, în sine, reprezintă partea de servere.