Online Energy Utility Platform

Profesor îndrumător: Student: Burian Andrei

Toderean Liana Maria Grupa: 30641

Universitatea Tehnică din Cluj-Napoca

Facultatea de Automatică și Calculatoare

Disciplina: Sisteme Informatice Distribuite

2022-2023

# Cerintele aplicatiei

Primul modul al sistemului constă într-o platformă online menită să gestioneze clienții, dispozitive echipate cu senzori de consum de energie asociate clientilor și date monitorizate de la senzori. Sistemul poate fi accesat de două tipuri de utilizatori după un proces de conectare: administrator și clienți.

Administratorul poate efectua operatii CRUD pe conturile clientilor (definite prin ID, nume, data nașterii, adresa), pe dispozitivele inteligente înregistrate care au senzori atașați pentru a monitoriza consumul de energie (definite prin ID, descriere, adresă (locație), consum maxim de energie per ora) și privind maparea clienților la dispozitive (fiecare client poate deține unul sau mai multe dispozitive care sunt monitorizat), iar fiecare dispozitiv are un senzor care își monitorizează energia consum. De obicei, dispozitivul este fie o casă echipată cu un contor inteligent de putere, fie o parte din o casă, caz în care un client are mai multe dispozitive. Senzorul înregistrează periodic, la marcaje temporale fixe, tuple de forma <data si ora, consum de energie>, unde consumul de energie este un contor care măsoară în kWh energia totală consumată de dispozitiv de când a fost pornit. Fiecare client își poate vizualiza dispozitivele, și consumul lor de energie actual și istoric, precum și consumul total de energie al dispozitivelor.

De asemenea, trebuie sa citim valorile prezentate de un senzor, la interval fix. Aceste valori vor fi in formatul unui fisier CSV, iar acestea trebuie procesate intr-un fisier JSON, care mai apoi sa fie transmis unei cozi de tip RabbitMQ. Pe urma elementele cozii vor trebui citite si procesate de server si in cazul unei depasiri a limitei pe ora se va notifica clientul.

O ultima functionalitate care trebuie implementata este optiunea clientilor de a putea trimite un mesaj catre administratorul site-ului, astfel putand sa aiba loc o conversatie in timp real intre clienti si administrator. De asemenea, administratorul poate primi si trimite mesaje cu mai multi clienti in acelasi timp.

# 2.Conceptul arhitectural al sistemului

Diagram

Description automatically generated

Pentru realizarea acestui proiect am ales sa folosesc o arhitectura de tipul server-client. Astfel am impartit proiectul in doua componente, una dintre ele ocupandu-se de partea de Backend, iar cealalta de Frontend, asa cum ne-a fost prezentat si in exemplele de la laborator. Dar mai avem si o a treia componenta, si anume o aplicatie care simuleaza citirea de valori de la un senzor.

Pentru partea de Backend am ales sa folosesc limbajul de programare Java in combinatie cu framework-ul Spring. Spring Framework este o platforma cu open source pentru simplificarea crearii aplicatiilor in limbajul Java, dar exista si o versiune pentru Platforma .NET.

Desi este folosit in principal pentru platforma Java EE, Spring poate fi utilizat pe orice aplicatie Java.

Acest framework este unul foarte folositor deoarece exista multe adnotari folositoare, dar si pentru prelucrearea datelor pentru baza de date. Acesta usureaza munca programatorului si prin detinerea unor interfete specifice cum ar fi Java Persistance API (JPA). Aceasta interfata permite utilizatorlui sa nu mai fie nevoit sa scrie manual metodele din repositories, deoarece aceastea sunt deja implementate. Dezvoltatorii Spring au facut posibila utilizarea directa a API-urilor Hibernate si JPA. Acest lucru necesita totusi un management transparent al tranzactiilor, deoarece codul aplicatiei nu isi mai asuma responsabilitatea de a obtine si inchide resursele bazei de date si nu accepta traducerea exceptiilor.

Pentru partea de Frontend am ales limbajul de Programare JavaScript in combinatie cu Libraria React.

JavaScript este un limbaj de programare care sta la baza Internetului, cum il stim noi astazi, 98% din site-urile web folosindu-se de JS pentru monitorizarea comportamentului clientului. React.js este un o librarie gratuita pentru front-end folosita pentru construirea de interfete de utilizator sau componente ale interfetei de utilizare. Este detinuta si intretinuta de Facebook si de o comunitate de dezvoltatori individuali dar si de multe alte companii. React poate fi folosit ca baza in dezvoltarea de aplicatii web sau mobile.

Aplicatia care simuleaza senzorul citeste aceste valori dintr-un fisier CSV, converteste aceasta citire, impreuna cu un sensorID (dat ca paramtru in fisierul application.properties) si cu un data si ora citirii si le pune intr-un fisier JSON pe care il trimite in coada RabbitMQ. Dupa care, obiectele din aceasta coada sunt citite din backend de catre o metoda consumer, care ia aceste date, le proceseaza si adauga intrari in baza de date, iar in cazul unei depasiri a valorii maxime pe ora se notifica clientul prin intermediul unui websocket.

De asemenea, partea de chat in timp real a fost implementata folosind WebSockets. Am initializat un destination prefix pentru useri in backend impreuna cu o metoda in controller (WebSocketController) care gestioneaza mesajele trimise de catre useri din frontend. Astfel toate mesajele trimise catre mapping-ul "/private-message" din backend vor fi redirectionate inapoi catre frontend catre user-ul corespunzator, utilizandu-ne de maparile corespunzatoare fiecarui user. De exemplu fiecare user trebuie sa dea subscribe la topicul "/user/numeUser/private", deoarece toate mesajele ce il au destinatar pe numeUser vor fi trimise intr-acolo. In frontend de fiecare data cand un mesaj nou este primit se actualizeaza lista conversatiilor (in special in pagina adminului), dar si lista mesajelor dintr-o conversatie, atunci cand ea deja exista. Administratorul poate selecta in ce conversatie doreste sa scrie un mesaj, iar atunci cand un mesaj nou este primit se deschide o noua conversatie si este afisat mesajul primit.

# 3.Diagrama bazei de date

A picture containing graphical user interface

Description automatically generated

Datale aplicatiei au fost stocate folosind baza de date pgAdmin oferita de catre PostgresSQL. Aici avem un tabel pentru conturile de utilizator (accounts), unul pentru rolurile utilizatorilor (admin sau client), unul pentru dispozitive (device-uri) si ultimul pentru memorarea citirilor senzorilor (history). Partea cu rolurile nu este vizibila pe partea clientului, deoarece atunci cand un cont nou este creat se asigneaza automat ca si fiind un cont de tip client.

Partea de Frontend a fost realizata folosind limbaje precum JavaScript, Html, Css in combinatie cu libraria React. Pentru fiecare pagina aferenta (admin, customer, login) am creat cate un director in folderul src.

4. Diagrama UML pentru Deployment

Graphical user interface

Description automatically generated

Aceasta diagrama arata comunicarea dintre partea de server si partea de client a aplicatiilor Web. Backend-ul comunica cu frontend-ul prin data transfer objects (dto). Serverul de Java este legat de baza de date cu care comunica prin intermediul repository-urilor si ia informatii de la aceasta prin interogari. Clientul (browserul web) comunica cu serverul prin intermediul requesturilor de tip HTTP.

Partea de deployment a aplicatiei a fost realizata folosind Docker, unde am reusit sa formez doua imagini pentru frontend si pentru backend si atunci cand le rulam in containere aplicatia mea este functionala doar din acele containere de docker, dar asta intamplandu-se doar local, pe calculatorul meu.

# 5. Readme

Pentru utilizarea aplicatiei local, se creeaza 3 directoare pentru backend, frontend si simulatorul de senzori si se cloneaza cele 3 repository git aferente. Apoi se deschide proiectul de backend si se ruleaza, iar deschiderea aplicatiei de frontend se face cu comanda : npm start. Dupa acestea se porneste simulatorul de senzori si se da run. Atunci incepe sa se populeze coada RabbitMQ.