# Universitatea Tehnică din Cluj-Napoca

# DISTRIBUTED SYSTEMS

# Assignment 2

# Asynchronous Communication

# Sensor Monitoring System and

# Real-Time Notification

Sirbu Oana-Bianca

Cerinte

Proiectul presupune o platformă online implementată pentru a gestiona utilizatorii, dispozitivele lor asociate și datele monitorizate de la fiecare dispozitiv. Sistemul poate fi accesat

de două tipuri de utilizatori după un proces de conectare: administrator și clienți.

Administratorul poate efectua operațiuni CRUD (Creare-Read-Update-Delete) pe conturile de utilizator (definite prin ID,nume, rol: admin/client), dispozitive inteligente de contorizare a energiei înregistrate (definite prin ID, descriere,adresa, consumul maxim de energie pe oră) și pe maparea utilizatorilor la dispozitive (fiecare utilizatorpoate deține unul sau mai multe dispozitive inteligente în locații diferite).

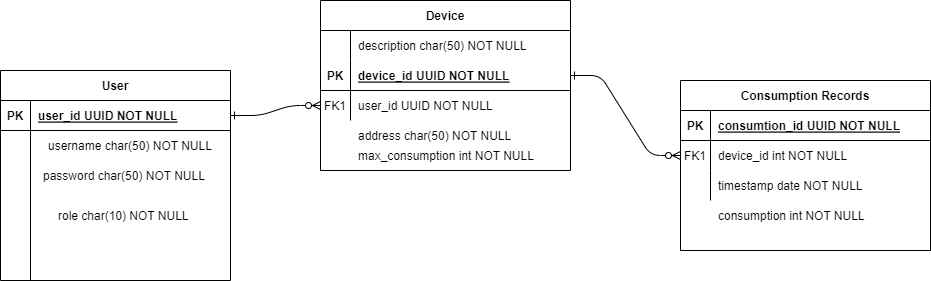
Ulterior, aplicatiei a fost adaugat feature-ul de broker de mesaje care adună date de la device-uri, preprocesează datele pentru a calcula consumul zilnic pe ore si îl stochează în baza de date. Broker-ul de mesaje va fi de fapt un simulator care citeste valorile senzorilor de la un anumit device presetat la o anumita perioada de timp si trimise in coada in format JSON.

Tehnologii folosite

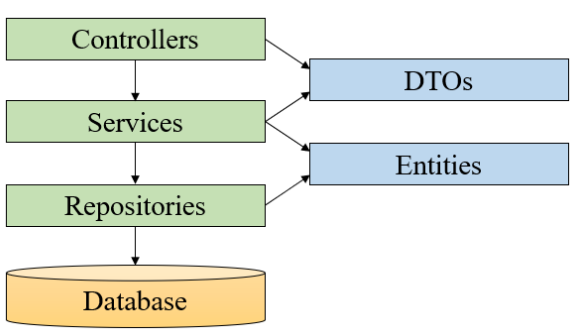
Aplicatia web a fost implementata cu ajutorul tehnologiilor:

* Backend: servicii REST -> **Java Spring REST, RabbitMQ (CloudAmpq queue), WebSockets.**
* Frontend: framework baza pe **JavaScript ->ReactJS**
* Baza de date folosita :**PostgreSQL**

Design-ul bazei de date



Arhitectura conceptuala a aplicatiei



**DTOs**: data transfer object folosite in proiect sunt cele specifice pentru client, device si login. Cateva exemple de DTO:

-ClientDetailsDTO are rolul de a contine si parola unui user, fiind folosit doar in cazul de update a parolei de catre admin

-LoginResponseDTO contine id-ul user-ului si rolul sau cu scopul de redirectionare in functie de autentificare.

**Entities**: corespund tabelelor din baza de date si fiecare instanta corespunde unui rand din tabela. Exemple:

-Client: are campurile id (tip UUID), username(tip String), password(tip String) si role(user sau admin)

-Device: are campurile id (tip UUID), description(tip String), address(tip String), maximum-consumption(tip int), user\_id(tip UUID)

-ConsumptionRecord:are campurile id (tip UUID),device\_id (tip UUID), consumption(tip int) si timestamp(tip Date)

**Repositories**: contine clase care faciliteaza accesul la baza de date, avand posibilitatea de a crea interogari custom:pentru user am creat findUserByUsername, iar pentru device findDeviceByOwnerId

**Services**:acest layer reprezinta business logic-ul aplicatiei. Transforma obiectele DTO in entitati si invers, folosind clasele Builder.

**Controller**: reprezinta un API care manipuleaza apelurile HTTP pentru :

-AuthController: se ocupa de request-urile destinate partii de autentificare

-ClientController: se ocupa de request-urile destinate clientului:add/update/delete client, cautarea tuturor device-urilor proprii

Broker-ul de mesaje este compus din:

-RabbitMqService

- odata la 5 secunde se citeste din coada

-se filtreaza obiectele de tip consum de la ora curenta

-daca suma lor depaseste valoarea consumului maxim al device-ului respectiv prin websocket se trimite la frontend un mesaj de alerta (se afiseaza o notificare/alerta pe pagina clientului). Frontend-ul se “aboneaza” la STOMP endpoint pentru a se putea realiza conexiunea cu backend-ul.

-WebsocketConfig – adauga endpoint-ul STOMP prin SocketJS pentru a se putea realiza conexiunea cu frontend-ul.

-NotificationController – reprezinta intermediarul care trimite mesajul trimis ca parametru la endpoint-ul STOMP, iar cei care s-au “abonat” la acel endpoint vor primi mesajul.

Deployment diagram

