

ASSIGNMENT 1

1. REACT APP

Aplicația pe partea de front end este făcută în React JS. Aceasta este împărțită în mai multe componente, în fișiere diferite. În fișierul admin avem toate componentele specifice administratorului. Aceasta este împărțită în adminHome și navigation-bar, 2 componente react specifice doar administratorului și în alte 2 fișiere, user și device. În acele fișiere se găsesc componente specifice pe partea de user (show users, update, delete, insert) cât și pe partea de device. Un alt fișier principal este cel de assets unde se găsesc imaginile, iar ultimul este cel de login care va fi comun atât pentru administrator cât și pentru basic user. Ultimul fișier principal este cel de commons, unde găsim împartite API-urile integrate împartite pe user și device.

Aplicația front end apelează API-uri pentru toate aceste acțiuni, de unde își ia datele din backend, respectiv de la baza de date. Se folosesc requesturi de post, get și delete.

2. SPRING APP

Partea de backend este făcută cu ajutorul Spring Boot. Aceasta aplicatie construiește API-urile pe care le apelăm pe front end.

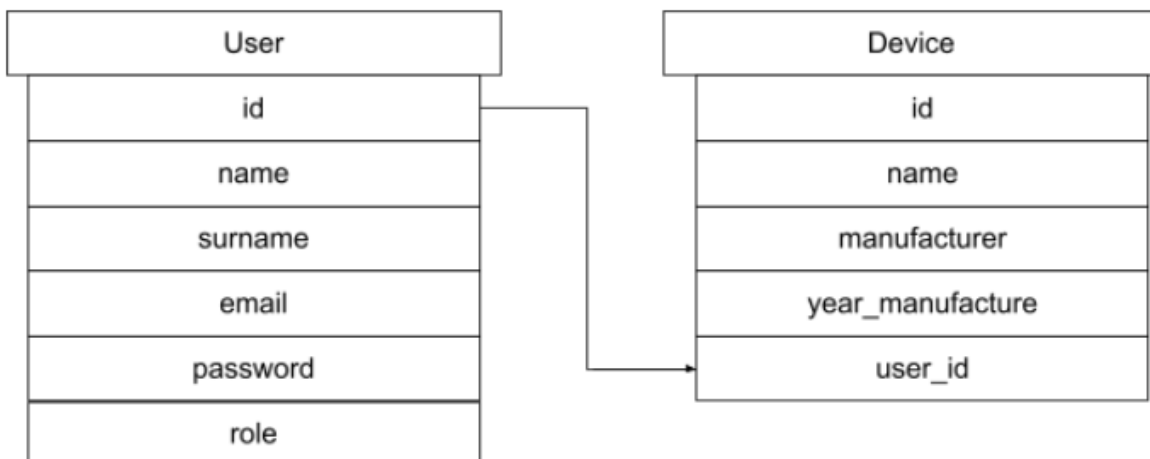
- localhost:8080/users (GET - get all users)
- localhost:8080/user/userByEmailPass (POST - primește email-ul și parola și returnează ID-ul utilizatorului și tipul acestuia)
- localhost:8080/users/insertProsumer (POST - inserează un nou user în tabela users)
- localhost:8080/users/{id} (GET - returnează userul cu ID-ul specificat ca parametru)
- localhost:8080/users/delete/{id} (DELETE - șterge din tabela userul cu ID-ul specificat ca parametru)
- localhost:8080/users/update (POST - face update la userul pe care îl primește în body. Acesta trebuie să conțină ID-ul conform căruia îl va căuta în tabela pentru a actualiza datele trimise)
- localhost:8080/device (GET - get all devices)
- localhost:8080/device/insertProsumer (POST - inserează un nou device în tabela device)
- localhost:8080/device/{id} (GET - returnează deviceul cu ID-ul specificat ca parametru)
- localhost:8080/device/delete/{id} (DELETE - șterge din tabela deviceul cu ID-ul specificat ca parametru)
- localhost:8080/device/update (POST - face update la deviceul pe care îl primește în body. Acesta trebuie să conțină ID-ul conform căruia îl va căuta în tabela pentru a actualiza datele trimise)

Arhitectura : Layered

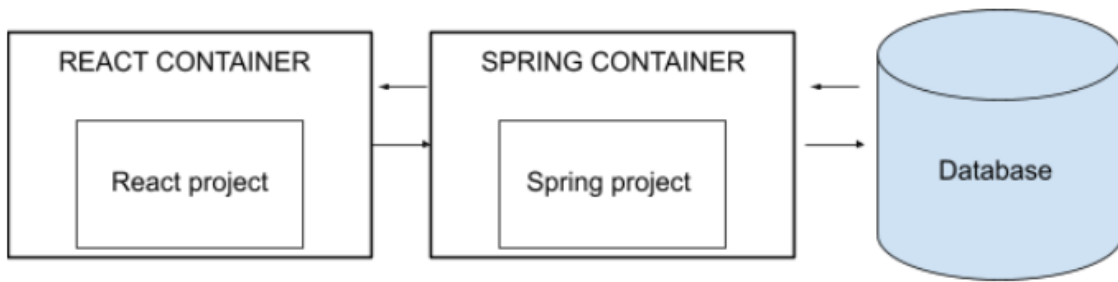


3. DATABASE

Baza de date pe care o folosesc este postgres. In baza de date sd avem tabelele : user si device



DEPLOY DIAGRAM



ASSIGNMENT 2

1. REACT APP

In plus fata de primul assignment, am adaugat partea de uses. Fiecare user se va loga folosind credentialele create de admin, iar la intrarea in aplicatie va observa graficele cu device-urile sale (consumption-oy si timestamp-ox)

2. SPRING APP

La partea de backend in Spring am adaugat un consumer. M-am folosit de rabbitMQ si am citit din coada generata de catre producer. Se conecteaza aplicatia la coada si se citeste. Datele primite se parseaza si se introduc in baza de date.

3. SIMULATOR QUEUE

La partea de backend legata de queue simulator dupa conectarea la RabbitMQ, se seteaza un interval de timp la care sa se trimita un mesaj. Mesajul este trimis upa un sablon setat, in forma de json. Se trimit deviceId(harcodat in aplicatie), timestamp (data curenta la care s-a trimis mesjul) si measurements (o marime citita din sensor.csv).

4. DATABASE

Baza de date pe care o folosesc este postgres. In baza de date, in plus fara de assgn 1 am adaugat tabela de measurements :

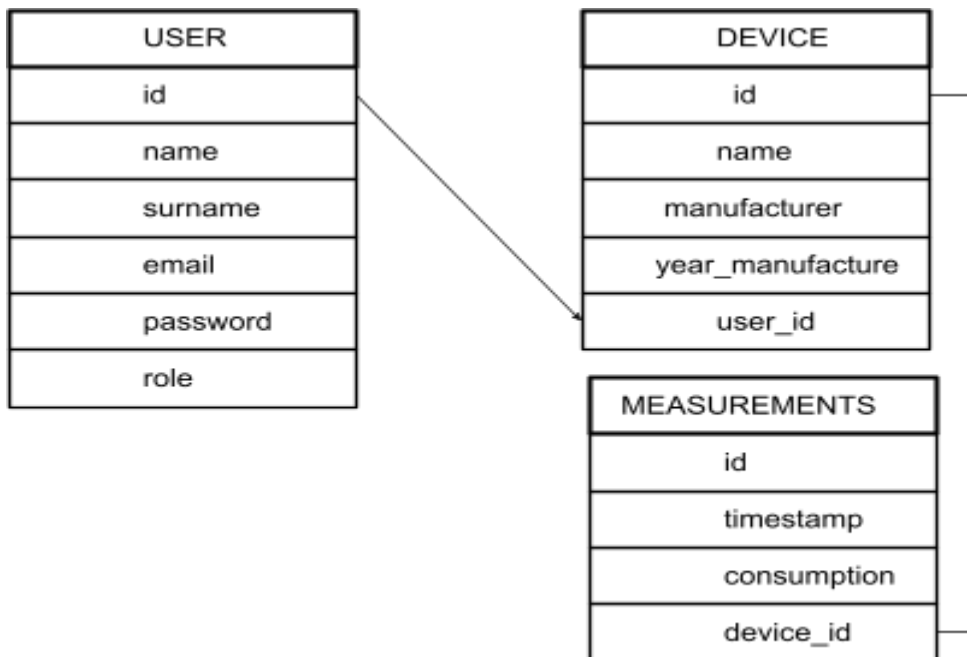


DIAGRAMA DE DEPLOY

