MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE

Online Energy Utility Platform

Vamvu Denisa-Elena

Grupa: 30241

Cuprins

1. Obiectivul temei 3
2. Decizii de proiectare 3
3. Arhitectura conceptuala 4
4. Detalii de implementare 4
5. Deployment 5

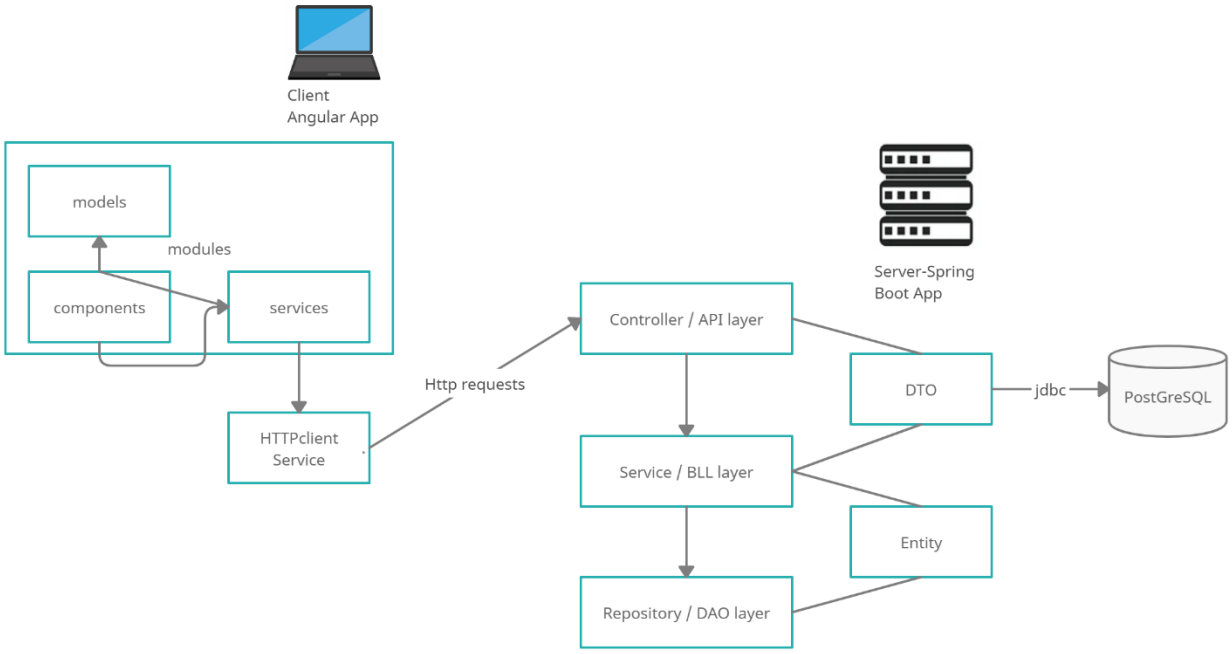
1. Obiectivul temei

Obiectivul acestei teme il reprezinta crearea unei platforme online destinata administrarii clientilor, a unor dispozitive inteligente echipate cu senzori pentru monitorizarea consumului de energie si a datelor provenite de la acesti senzori. Acest system poate fi accesat de doua tipuri de utilizatori, dupa un proces de logare: administrator si client. Administratorul poate crea, citi, modifica si sterge clienti, dispositive, senzori; de asemenea, poate atribui unui dispozitiv un senzor si poate atribui unui client mai multe dispozitive. Utilizatorul obisnuit isi poate vedea toate dispozitivele pe care le detine si poate vizualiza consumul de energie in functie de data.

2. Decizii de proiectare

Aplicatia distrubuita este construita pe baza conceptului arhitectural client-server, in care serverul gazduieste, livreaza si administreaza majoritatea resurselor si serviciilor pentru ca acestea sa poata fi consumate de catre client.

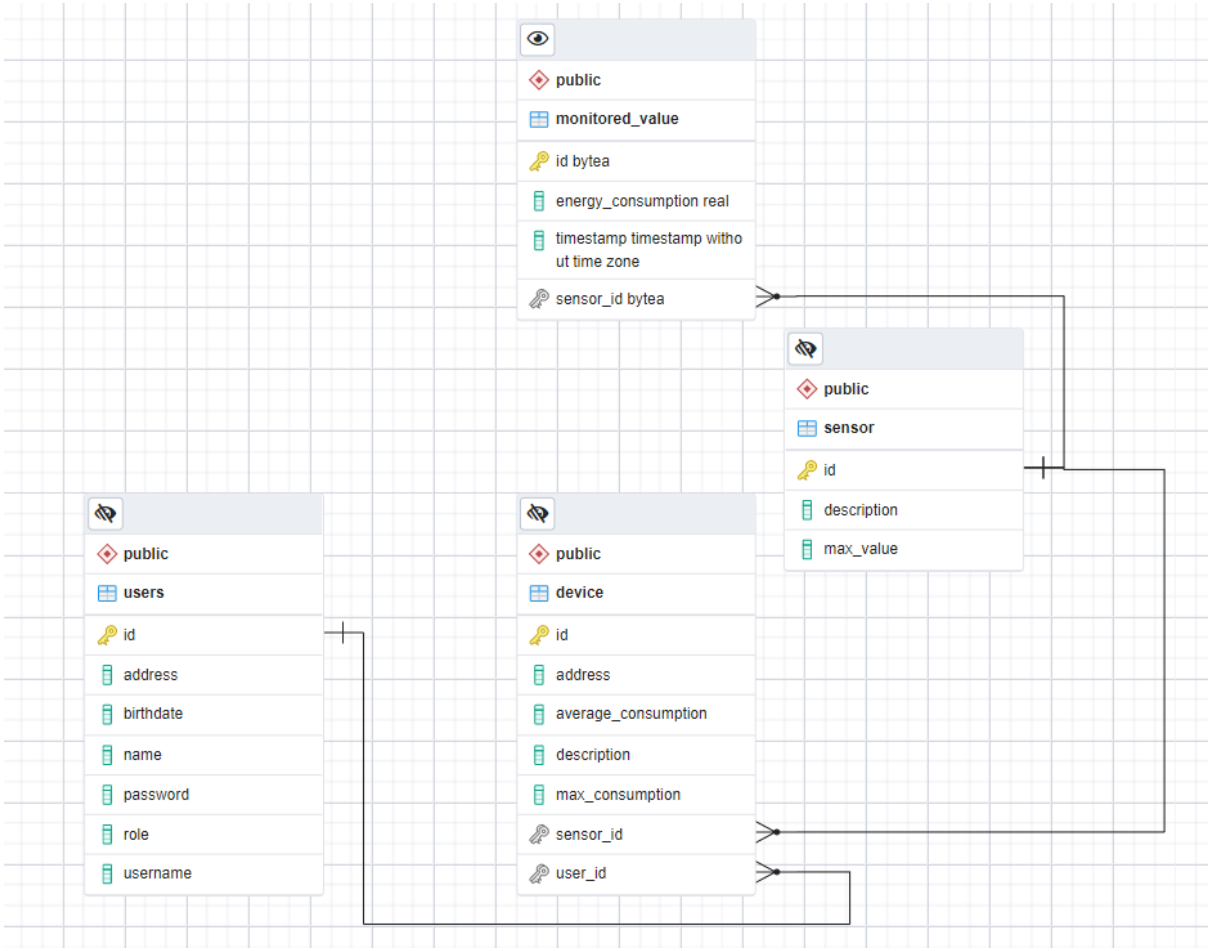
Diagrama conceptuala a sistemului distribuit reprezinta cele 3 componente necesare: o baza de date, unde se vor stoca toate informatiile necesare aplicatiei, un server, reprezentat de o aplicatie Spring Boot, ce va pune la dispozitia clientului toate datele necesare si un client, o aplicatie angular care comunica cu serverul pentru a cere, a schimba sau a sterge date.



Flow-ul executiei poate fi descris astfel: requesturile HTTP (GET, POST, PUT, DELETE) care sunt facute de catre client, sunt directionate catre REST API controllers din server, care le vor executa apeland serviciile. Service layerul contine toata logica de care au nevoie requesturile, efectuata asupra detelor care sunt mapate la JPA prin clasele entitate.

3. Arhitectura conceptuala

Schema bazei de date poate fi determinata cu ajutorul cerintei: se cunoaste faptul ca un utilizator poate avea doua roluri, iar in cazul rolului de client acesta detine un set de dispozitive. Fiecare dispozitiv este legat la un singur senzor. Fiecare senzor inregistreaza periodic o valoare de forma <timestamp, consum de energie>, deci va contine un set de astfel de valori.



4. Detalii de implementare

Serverul este implementat sub forma unei arhitecturi pe layere, in Java, cu ajutorul framework-ului Spring Boot. Entitatile au fost create cu ajutorul frameworkului Hibernate, care mapeaza fiecare instanta a entitatii cu o inregistrare in baza de date. Pentru transportul datelor intre client si server s-au folosit o serie de DTO-uri, fiecare dintre acestea avand cate un scop clar, spre exemplu: crearea unui user, editarea unui user, afisarea unui user. DTO-urile aceleiasi entitati difera cu scopul de a elimina datele redundante unui anumit use-case. DTO-urile sunt folosite in layerul de servicii. Pentru conversia unei entitati intr-un DTO sau invers, au fost implementate clase speciale de conversie, numite buildere cu metode specific: toDTO, toEntity.

Pentru rolul de administrator au fost implementate operatii CRUD pe entitatile User, Device, Sensor. In scopul testarii acesta are dreptul de a introduce si valori monitorizate pentru anumiti senzori. In cazul stergerii unui device sau al unui senzor, obiectul ce contine cheia straina nu se va sterge din baza de date. La fel se intampla si in cazul in care se va sterge un client.

Pentru logarea in aplicatie s-a folosit Spring Security, si s-a generat cu ajutorul unui service un JWT authentication token, care are o anumita perioada in care este valabil si de care va fi nevoie pentru a autoriza toate requesturile. Tokenul este returnat intr-un DTO alaturi de rolul si id-ul userului curent.

Clientul a fost implementat in angular, un framework open-source bazat pe typescript. Aplicatia se deschide cu un form de login care apeleaza request-ul de autentificare. Se primeste inapoi in frontend userul curent, iar in functie de rolul acestuia se face redirect catre pagina de home. Un user nu poate accesa paginile altor useri sau paginile pentru care nu are drepturi. De asemenea, daca nu este autentificat, nu va reusi sa treaca de pagina de login. Toate acestea au fost implementate pe rute cu ajutorul guard-urilor, in functie de rol sau de validitatea si existenta tokenului de autentificare.

Structura proiectului este bazata pe modele, module si service-uri specifice fiecarei entitati din baza de date. Componentele care reprezinta paginile sunt grupate in module. Toate modulele sunt importate in modulul principal al aplicatiei. Service-urile efectueaza requesturile catre backend. Fiecare componenta are injectat un service sau mai multe in functie de nevoi.

Flow-ul aplicatiei este reprezentat de un navigation bar care contine link-uri catre toate functionalitatile. Rolul de admin dispune de 3 formulare pentru a crea si edita entitatile, si de 3 tabele care le afiseaza. Rolul de client dispune de o pagina in care isi poate vedea toate deviceurile asignate, alaturi de senzori atasati acestora si de o pagina in care poate selecta din calendar o zi pentru a-si vedea energia consumata sub forma unul grafic.

Utilizatorii au de asemenea posibilitatea de a se deloga din aplicatie.

5. Deployment

Pentru partea de proiect, s-a efectuat deploy atat pentru server, cat si pentru client. Astfel, baza de date care era pana acum locala, s-a mutat pe cloud cu ajutorul unui ad-on de pe Heroku. Link-ul pe care se fac requesturi este schimbat in cel al aplicatiei puse online, iar front-end-ul ruleaza de asemenea pe cloud. Aplicatia poate fi accesata direct de pe link-ul generat de Heroku.