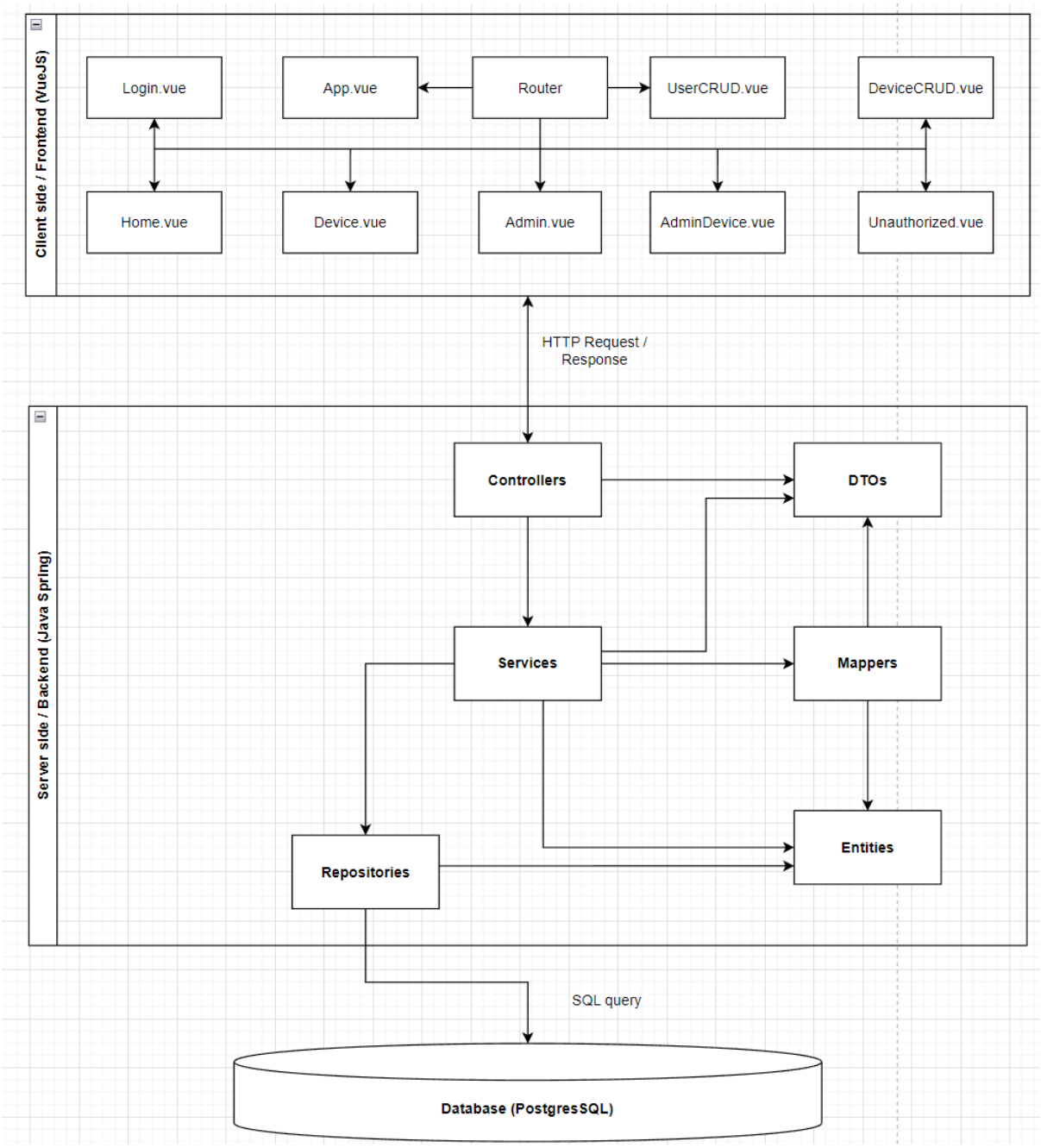
Saulea Alessandro, grupa 30242

# Online Energy Utility Platform – documentatie

## 1. Arhitectura conceptuala




Aplicatia prezinta o arhitectura de tip layered, cu 3 layere: *Presentation Layer* (Client side / frontend), *Business Logic* (Server side / Backend) si baza de date.

*Presentation Layer* reprezinta partea de frontend a aplicatiei, implementata folosind framework-ul VueJS, ce utilizeaza limbajele HTML, CSS si Javascript si ruleaza pe portul 3000 de pe *localhost*. Este alcatuita din 9 componente *vue*, care fie sunt mapate la url-urile folosite de aplicatie de catre un *Router*, fie sunt parti din

alte componente *vue*. O componenta *vue* este alcatuita din codul HTML, incapsulat in tagul <template>, codul in Javascript care se executa in cadrul componentei, incadrat in tagul <script>, impreuna cu librariile importate, variabilele ce sunt afisate de catre codul HTML , cu o metoda created/mounted care initializeaza comportamentul paginii si alte metode folosite de aceasta si optional o sectiune de stilizare CSS inclusa intr-un tag <style>.

*App.vue* este componenta principala a aplicatiei VueJS, aparand pe orice link al aplicatiei, afisand doar app-bar-ul din partea de sus al ercanului, cu logo-ul Vuetify, numele proiectului "Online Energy Utility Platform", impreuna cu numele utilizatorului curent si butonul "Log out", in cazul in care utilizatorul este logat, si butonul "Administration" care trimite la paginile de administrare a bazei de date, daca utilizatorul este administrator.

In cadrul aplicatiei, orice utilizator inregistrat in baza de date se poate loga din fereastra de logare. In cazul in care username-ul si parola transmise sunt corecte, utilizatorul va fi redirectionat catre pagina corespunzatoare conform rolului sau (client sau administrator) si al sesiunii acestuia. Orice utilizator are acces la lista de device-uri pe care le detine, si la graficele cu consumul de electricitate la fiecare ora ale acestor device-uri dupa o zi pe care el o poate selecta dintr-un calendar. Nu poate accesa paginile de vizualizare a consumului zilnic de curent electric al unui device pe care nu il detine, ca masura de Securitate pe baza sesiunii. Conform filtrarii pe rol si sesiune, utilizatorul obisnuit (clientul) nu-si poate modifica informatiile personale (username, parola, nume) si nu poate accesa paginile din aplicatie ce tin de gestionarea bazei de date a aplicatiei (operatii CRUD pe utilizatori, device-uri si masurari de consum de curent, mapare device-uri la utilizatori), corespunzatoare rolului de administrator. De asemenea administratorul poate sa efectueze operatii CRUD pe orice alt cont din baza de date, cu exceptia propriului cont.

*Business Logic* este partea de backend a aplicatiei, implementata de framework-ul Spring din Java folosit la dezvoltarea de aplicatii web, si ruleaza pe portul 8080 de pe *localhost*, comunicand cu frontend-ul printr-un server tocmai generat de aplicatia *spring boot*. Cuprinde 4 layere (Controllers, Services, Repositories, Entities) impreuna cu DTOs (Data Transfer Objects) si mappers.

Layer-ul de entitati (*Entities*) specifica structura bazei de date, tabelele acesteia fiind generate automat prin tool-ul *Hibernate*, atributele din entitati fiind transformate in coloanele din tabele folosind adnotarea @Column, iar relatiile dintre tabele sunt generate cu adnotarile @OneToMany, @ManyToOne si @JoinColumn. Cele trei entitati pe care lucreaza aplicatia sunt User, Device si Measurement (masuratori pe device-uri) si folosesc adnotarea @Entity. Pentru reprezentarea si generarea cheii primare sunt folosite adnotarile @Id si @GeneratedValue. Id-urile userilor si ale device-urilor sunt de tip UUID, in timp ce la Measurement am folosit o cheie primara dubla, alcatuita din device-ul la care se face masurarea si timestamp-ul, si marcata cu adnotarea @EmbeddedId.

Pentru un transfer mai usor al informatiilor sau doar al celor necesare de la entitati, sunt folosite *DTOs* (Data Transfer Objects), fiind definite pentru entitati. *Mappers* realizeaza conversiile din entitate in obiect de transfer sau invers.

Layer-ul de *Repositories* se ocupa cu persistenta datelor in baza de date. Fiecare repository este definit in functie de entitatea pe care lucreaza si extinde interfata *JpaRepository<T, ID>*. Implicit, repository-urile implementeaza metodele *findAll*, *findById*, *save* si *delete* din *JpaRepository*, alte metode de tip *findBy* sau *findAllBy*, care filtreaza dupa una sau mai multe coloane trebuind introduse manual in repository. Aceste metode nu au o implementare manuala de catre programator, fiind implementate automat de sistemul JPA. Se pot defini si alte tipuri de metode, spre exemplu *findAllByDeviceAndDate* din *MeasurementRepository*, carora li se specifica si query-ul pe care il executa, cu adnotarea @Query. Crearea sau modificarea unui obiect in baza de date se face prin metoda *save* din repository, in timp ce *delete* sterge obiectul din baza de date.

Layer-ul de *Services* implementeaza business-logic-ul aplicatiei. Cele 3 service-uri folosesc adnotarea @Service pentru a fi recunoscute de Spring ca si services. Fiecare dintre acestea opereaza pe cate un repository definit pe entitatea cu care lucreaza, si implementeaza operatiile CRUD pe entitati apeland metode din repository. Fiecare metodea implementata de services primeste ca parametriu un DTO, il in entitate folosind mappers, iar dupa ce il prelucreaza si il salveaza in baza de date cu metoda *save* din repository, il transforma inapoi in DTO folosind mappers si returneaza obiectul obtinut. *UserService* are in plus implementat si login-ul, primind ca parametru un LoginDTO (contine username-ul si parola user-ului ce incearca autentificarea in aplicatie), iar la metodele *insert* / *update*, inainte de inserarea/actualizarea unui user verifica daca exista deja un user cu username-ul primit prin DTO.

Layer-ul de *Controllers* asociaza fiecarui endpoint prin care comunica cu frontend-ul unei metode din service. Endpoint-urile sunt diferentiate dupa url si dupa metoda HTTP (get, post, put, delete), folosindu-se adnotarile corespunzatoare @GetMapping, @PostMapping, @PutMapping si @DeleteMapping, fiind posibila utilizarea mai multor tipuri de request pe acelasi url. Controllers sunt marcate cu adnotarea @RestController

pentru a fi recunoscute de Spring, si cu adnotarea @CrossOrigin pentru accesul unui client ce ruleaza aplicatia pe un port diferit sau pe o alta masina  iar adnotarea @RequestMapping specifica root-ul tuturor endpoint-urilor folosite de fiecare controller. Metodele din controllers primesc ca parametri DTOs, pe care le transmit mai departe service-urilor.

## 2. Design baza de date

Diagrama UML a aplicatiei

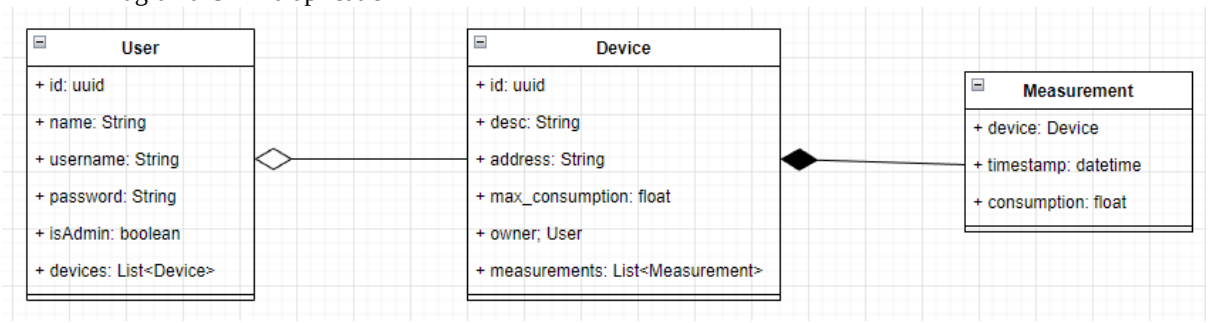
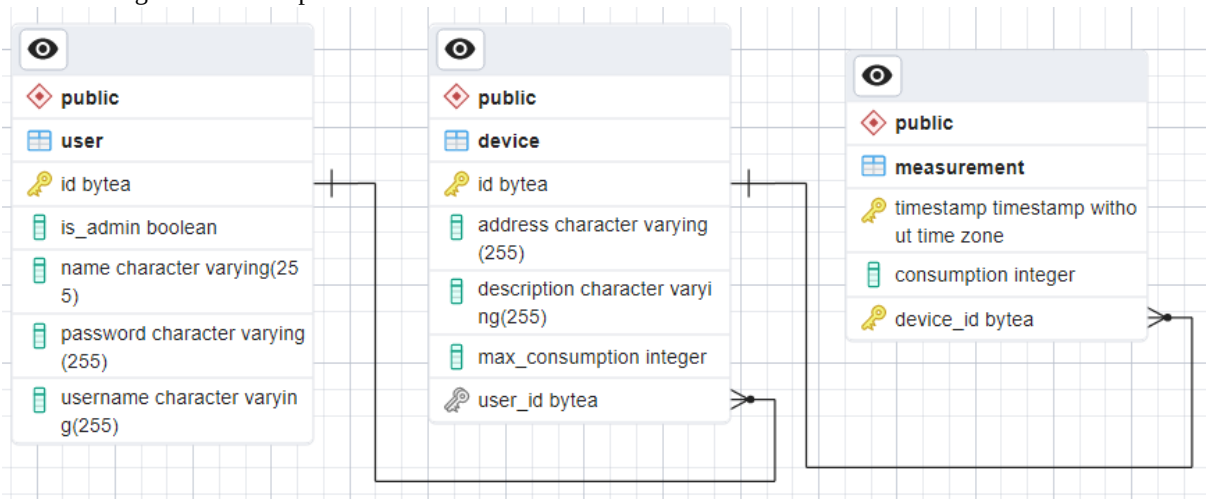


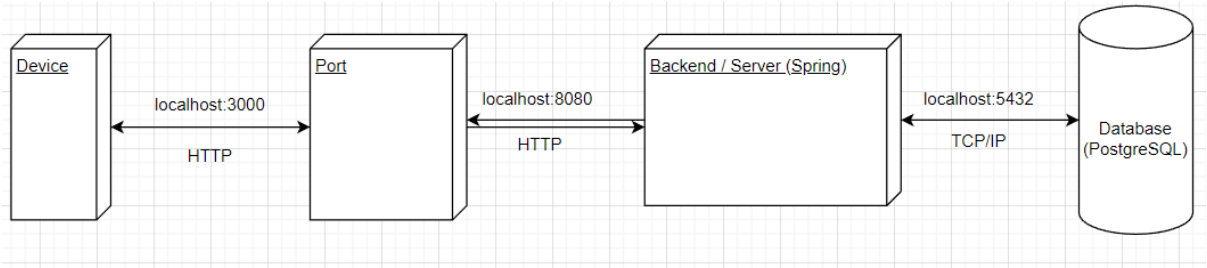

Diagrama ERD a aplicatiei




Baza de date este de tip PostreSQL, se numeste “energy” si ruleaza pe portul 5432. Cuprinde 3 tabele: User, Device si Measurement. User-ul este caracterizat de un id de tip UUID generat printr-o secventa, nume, username, parola, o valoare booleana isAdmin ce specifica daca user-ul este administrator sau nu. User-ul are asociata si o lista de device-uri, reprezentata in Spring cu adnotarea @OneToMany si @JoinColumn. Device-ul este caracterizat de un id de tip UUID generat printr-o secventa, descriere, adresa, consumul maxim de curent electric pe ora (max_consumption) si user-ul caruia ii apartine (referinta la id-ul user-ului). Referinta la user este adnotata cu @ManyToOne. Measurement (masura de consum de curent electric) are o cheie primara dubla, alcatuita din referinta device-ului caruia se face masurarea (prin id), adnotata cu @ManyToOne si timestamp (ora la care s-a facut masurarea). Am ales sa implementez astfel cheia primara deoarece oricum nu pot exista mai multe masuri pe aceeasi ora la acelasi device. Measurement este caracterizata si de consumul de curent facut de device la ora respectiva (consumption).

## 3. Diagrama de deployment UML

Aplicatiile de back-end, front-end si baza de date sunt procese separate si comunica intre ele in retea folosind diverse porturi si protocoale.


Device
localhost:3000
HTTP
Port
localhost:8080
HTTP
Backend / Server (Spring)
localhost:5432
TCP/IP
Database
(PostgreSQL)