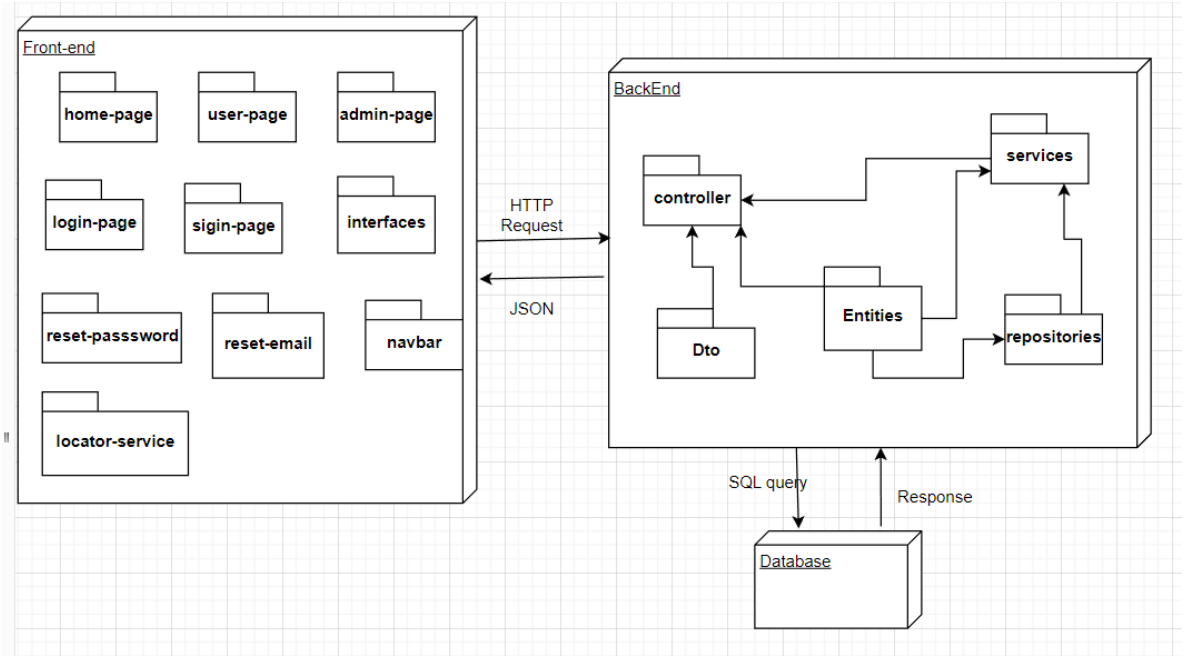
Documentatie Sisteme Distribuite. Aplicatie Web , client-server

Andercou Alexandru grupa 30241

Request-Reply Communication Paradigm Online Energy Utility Platform

- Conceptual architecture.



Site-ul web are o structura distribuita pe layere si arhitectura este de tip 3-tier. Cele trei componente sunt Presentation, Business Logic si Bazele de date. Presentation reprezinta front-endul site-utului scris in tehnologia Angular js , Node js , cu limbajele html, css si typeScript. Business Logic Layer si conexiunea la baze de date este scrisa in java folosind frameworkul spring-boot , conexiunea la baze de date realizandu-se cu ajutorul ORM. Baza de date este de tip relational , PostgreSQL.

Arhitectura mai poate fi privita sub viziunea client-server, front-endul cu ajutorul browserului reprezentant clientul care apeleaza backendul-endul sau serverul reprezentat de aplicatia java spring-boot impreuna cu baza de date , prin apeluri de tip HTTP. Atat fron-endul cat si back-endul dar si bazele de date sunt aplicatii diferitte care trebuie pornite separat pentru ca site-ul sa functioneze.

Front-endul sau site-ul web este format din 6-7 pagini web ,10 componente , un service si 7 interfete pentru comunicarea cu back-endul. Site-ul angular ruleaza pe portul 4200. Paginile principale sunt : login, home, sigin , user page,admin page, reset pasword page format din alte 2 pagini separate. Fiecarei pagini web ii este asociata o componenta care vine impreuna cu o pagina html si o pagina css, pe langa aceste componente mai avem o componenta pentru bara cu butoane , o componenta pentru rutare pe baza url-ului, si o componenta care agregeaza celalate componente pentru a crea un site unitar.

Pe pagina de admin administratorul este capabil sa vada utilizatorii atat de tip user cat si admin dar nu are acces la parole, si sa opereze CRUD pe datele utilizatorilor (exceptand parolele pentru securitate). De asemenea este capabil sa opereze CRUD pe deviceuri , pe mapari intre useri si deviceuri care pot fi cautate, create si sterse dar nu updatate si poate sa adauge si sa vizualizeze masuratori. Utilizatorul pe pagina de utilizator poate sa vada intr-un tabel datele referitoare la deviceurile acestuia si sa vada un chart cu consumul deviceurilor in functie de zi pentru fiecare ora in care s-au luat masuratori. Nici utilizatorii dar nici administratorii nu pot sa vizualizeze paginile user/administrator ala ltor useri sau administratori, aceasta functionalitate a fost implementata folosind sesssion storage pentru a pastra id-ul si tipul de utilizator stocat si prin redirectionarea in componentele paginilor user,admin care filtreaza pe baza sesiunii.

Login-ul ofera capacitatea te-a rediredctiona spre pagina de sign in dar si spre pagina de resetare a parolei. Pagina de resetare a parolei ii cere utilizatorului emaiul apoi trimite un email cu un link pentru o pagina in care se reseteaza parola.

Back-endul este o aplicatie java scrisa cu framework-ul spring-boot. Comunicareea cu front-endul se realizeaza cu ajutorul unui server tomcat generat de aplicatia spring boot. Serverul are asociat portul 8080. Aplicatia este structura sub forma de layere. Cele 4 layere sunt: Controlerele, Serviciile,Repositoriurile si Entitatiile .In layerul de Controlere mai exista un pachet de clase Data Transfer Objects (DTO) care contine clase folosite pentru comunicarea cu front-endul.

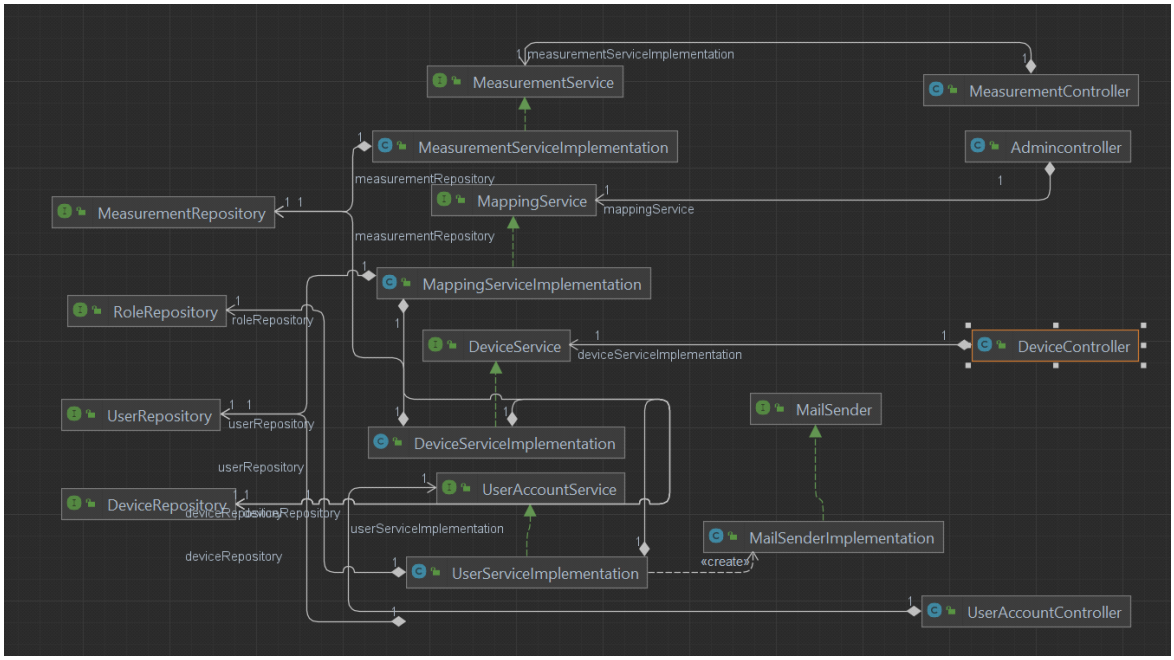
Layerul de controllere specifica pathurile de comunicare folosite pentru comunicarea http cu clientul reprezentat de front-end si tipul de metode http care vor fi folosite cu ajutorul adnotariilor spring @PostMapping si @GetMapping . Acest nivel apeleaza functionalitati din pachetul de servicii care implementeaza logica aplicatiei.Aplicatia foloseste 4 controlere pentru utilizatori, deviceuri,masuratori si administrator. Controlerele se folosesc de obiecte din clasele din pachetul Dto dar si clase definite In pachetul de Entitati care functioneaza si ca si clase model. Pentru a fi recunoscut de frameworkul springboot se foloseste adnotarea @RestController. Pentru a oferi acces clientului care ruleaza pe un port diferit sau chiar masina diferita se foloseste adnotarea @CrossOrigin.

Layerul de Servicii implementeaza logica businessului : Am implementate 5 servicii : pentru utilizatori, pentru deviceuri,pentru mapari , pentru masuratoiri si pentru trimiterea de emailuri. Functionalitatiile de login ,si resetare de parole este realizata de serviciul utilizatorilor. Logica masuratoriilor este distribuita in 2 servicii: measurement service si device service datorita relatiei care se afla intre aceste clase. Serviciile se folosesc de repositoriurri pentru a avea acces la obiectele din baza de date si pentru a efectua operatii crud.Serviciile folosesc mai mult decat un singur repository. Pentru a fi recunoscute ca servicii de aplicatia springboot se foloseste adnotarea @Service.

Layerul de repositories este reprezentat de interfete care acceseaza direct baza de date si definesc metodele crud care se pot efectua pe acestea.Fiecare entitate este mapat pe un singur reposotory . Repositoriurile mostenesc interfata CrudRepository definit in libraria springboot si adnotarea @Repository este folosita pentru ca frameworkul sa il recunoasca ca si repository si sa il instantieze cu obiecte care implementeaza interfetele.

Layerul de entitati definesc structura bazei de date , tabele bazei de date fiind generate automat prin metoda ORM , atributele transformanduse in coloane , si relatiile dintre tabele realizanduse cu ajutorul adnotariilor spring-boot. Tot acesta defineste modelele care sunt folosite de alte layere. Entitatile sunt recunoscute de framework-ul springboot cu ajutorul adnotarii @Entity. Relatiile intre tabele sunt realizate cu ajutorul adnotarilor @OneToMany si @JoinColumn . Deoarece tabele relationale au nevoie de o cheie primara se foloseste adnotarea @Id impreuna cu adnotarea @GeneratedValue.

Relatiile intre layere se realizeaza cu ajutorul adnotarii @Autowired folosit pentru a instantia clase din layerul inferior.



- DB design

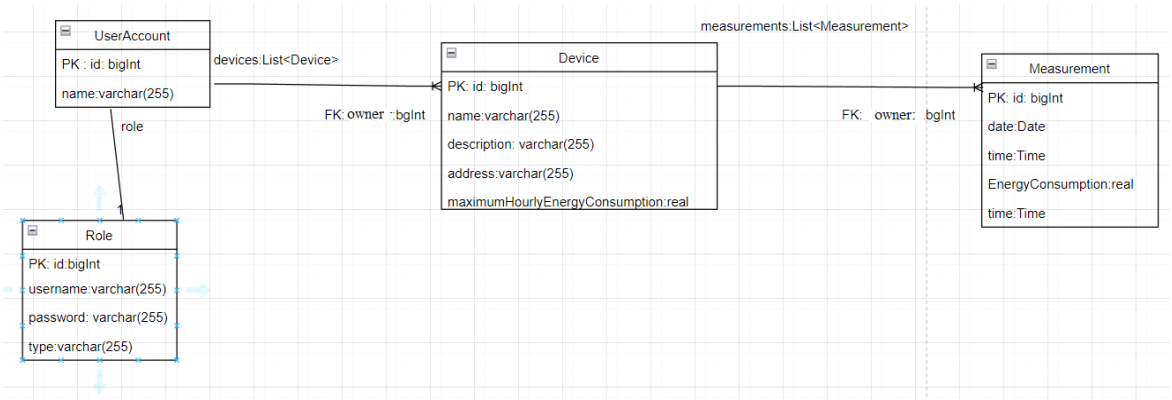
Baza de date PostrgreSQL cu utilizatorul : postgres, energy-db ruleaza pe portul 5431. Baza de date este formata din 4 tabele cu 2 relatii de one to many si una de one to one. Cele patru tabele sunt user_account,role,device si measurement. Tabele sunt generate de frameworkul java spring folosind ORM. Tabelele folosesc pe post de chei primare campuri de tip Long denumite sugestiv id care sunt generate automat cu adnotarea @GeneratedValue(strategy = GenerationType.AUTO)

Relatile one to many sunt implementate in java ca liste cu adnotarea @OneToMany, in tabele datorita adnotarii @JoinColum din spring sunt implementate ca o cheie straina de tip Long in tabela de a carui tip sunt listele catre tabelul corespunzator entitatii care contine lista , cheia este un camp indicat de atributul adnotarii,name ,numit owner . Astfel maparea user-device este realizata in felul urmator: userul contine in codul java o lista de deviceuri denumita devices , iar in schimb deviciurile contin in tabele o cheie numita owner cheie straina catre tabela user_account, acceasi metoda este folosita si pentru conectarea masuratoriulor si a devicurilor, cheia straina din masuratori se numeste tot owner.

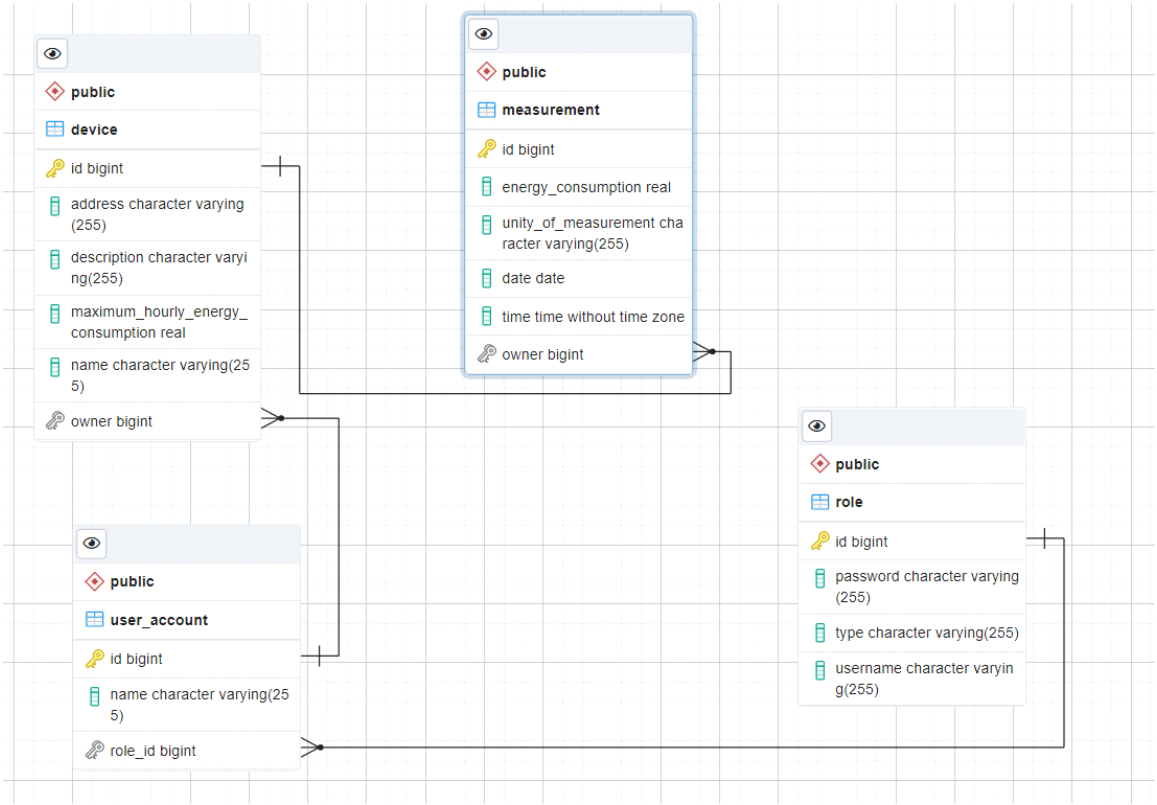
Relatia one to one intre user si role se implementeaza in cod java ca un atribut din clasa Role denumita role , si adnotata cu adnotarea @OneToOne si in tabela user_account ca o cheie straina catre tabela role numita role_id. Campul type din clasa role este un obiect de tip enum care are 2 posibilitati:USER,ADMIN , si este implementata in tabela role ca un string sau varchar(255).

In baza de date timestampul este implementat separat ca time si date si se realizeaza o tranzitie intre formate la nivelul serviciilor.

Model concepual al bazei de date

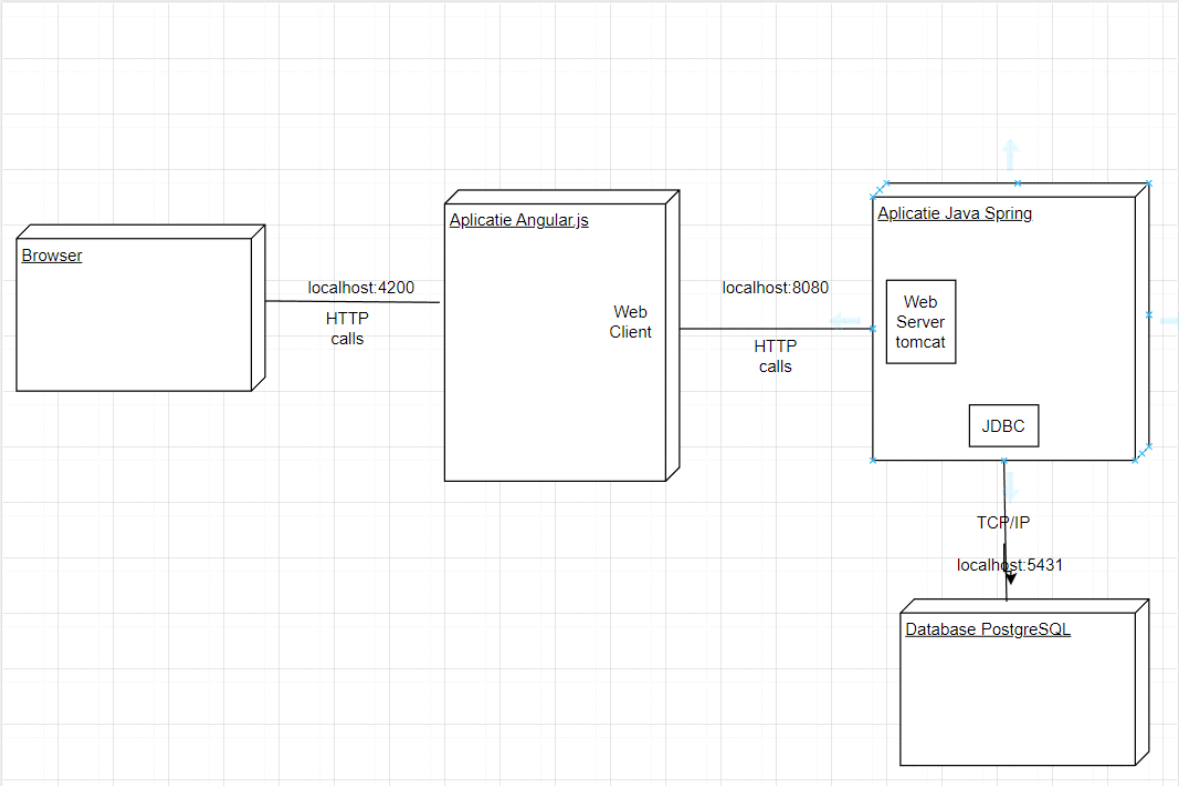


Model rezultat din ERD tool



- Deployment diagram.

Aplicatiile de front-end ,back-end si baza de date sunt procese separate ,si pot fi considerate noduri separate ,acestera au porturi dedicate si comunica in retea folosind mesaje de tip HTTP si TCP/IP. Porturile folosite sunt:4200,8080,5431.



D) READ ME INFORMATION

Folderele unde se afla aplicatiile sunt: energy-utility-platform_angular3 pentru aplicatia front-end angular si spring-demo pentru aplicatia server,back-end java.

Aplicatia angular ruleaza in 2 moduri , modul de development folosind comanda: ng serve ,si production folosind ng build.

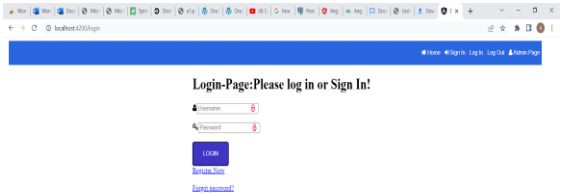
Dependintele aplicatiei energy-utility-platform sunt :Node.js ,angular/cli , angular-datatables,chart.js,jquery,ng-charts.js,ng2-charts,rxjs,tslib,zone.js,dependinte care se afla in fisierul : package.json , angular.json, https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css pentru awesome-font se afla in fisierul index.html

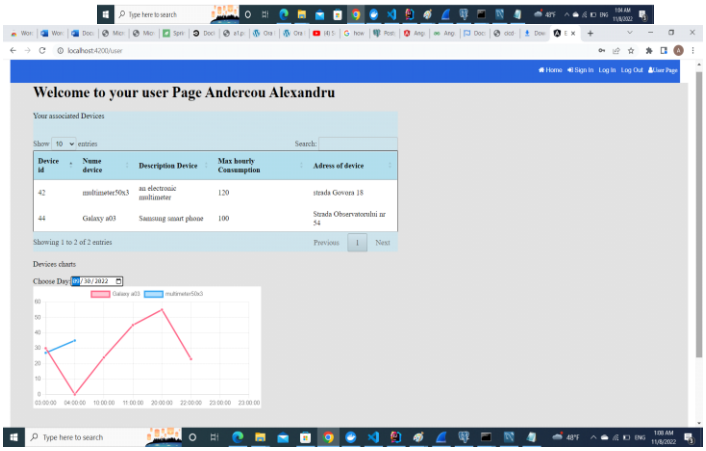
Aplicatia java-spring este o aplicatie java maven. Cu dependintele in fisierul pom.xml si informatii de tip resurse ale aplicatie(precum informatiile bazei de date,user,password, portul aplicatiei, ,cont pentru aplicatie de tip email) in spring demo/src/main/resources/application.propeties . Pentru rulare se poate folosi comanda mvn package care genereaza un fisier .jar si aplicatia se poate porni cu java –jar ds-2022-0.0.1-SNAPSHOT.jar sau dintr-un ide precum intelij /eclipse.

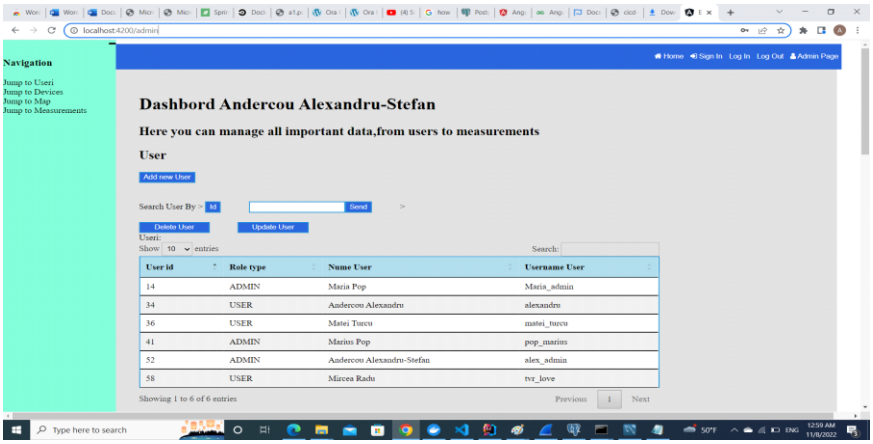
O alta metoda de rulare a aplicatiei este folosind docker. Pentru a crea containerele docker , se merge in folserele root ale aplicatilor java si angular si se foloseste din terminal comanda docker build . Care genereaza o imagine din fisierul Dockerfile. Pentru a crea si a porni un container se foloseste comanda docker-compose up –d care genereaza un container folosind fisierul docker-compose.yml

Result images:

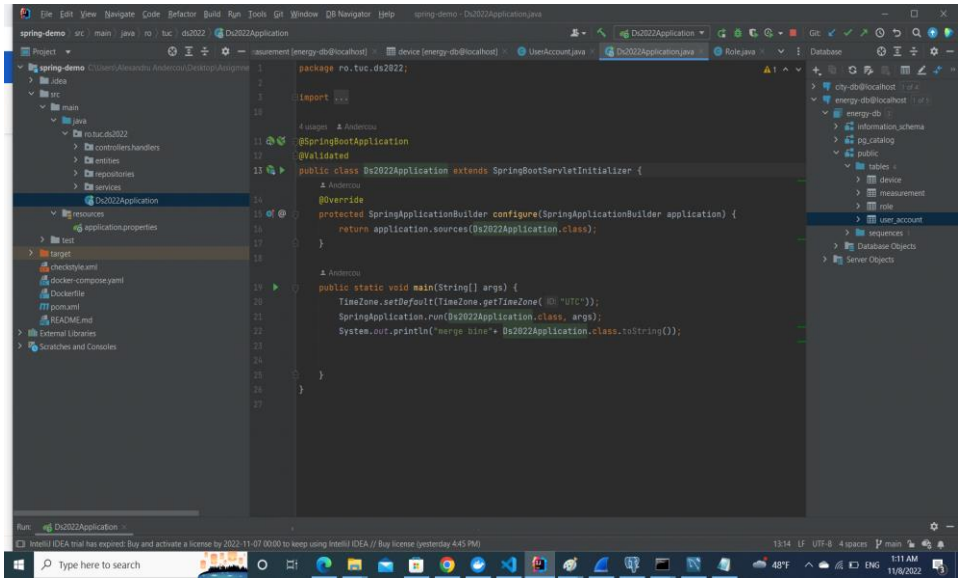
Front-end:

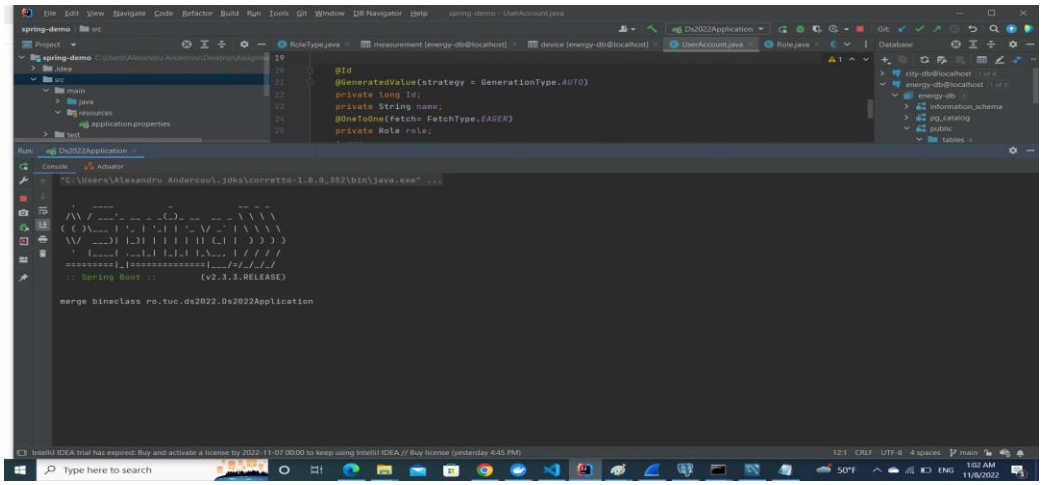




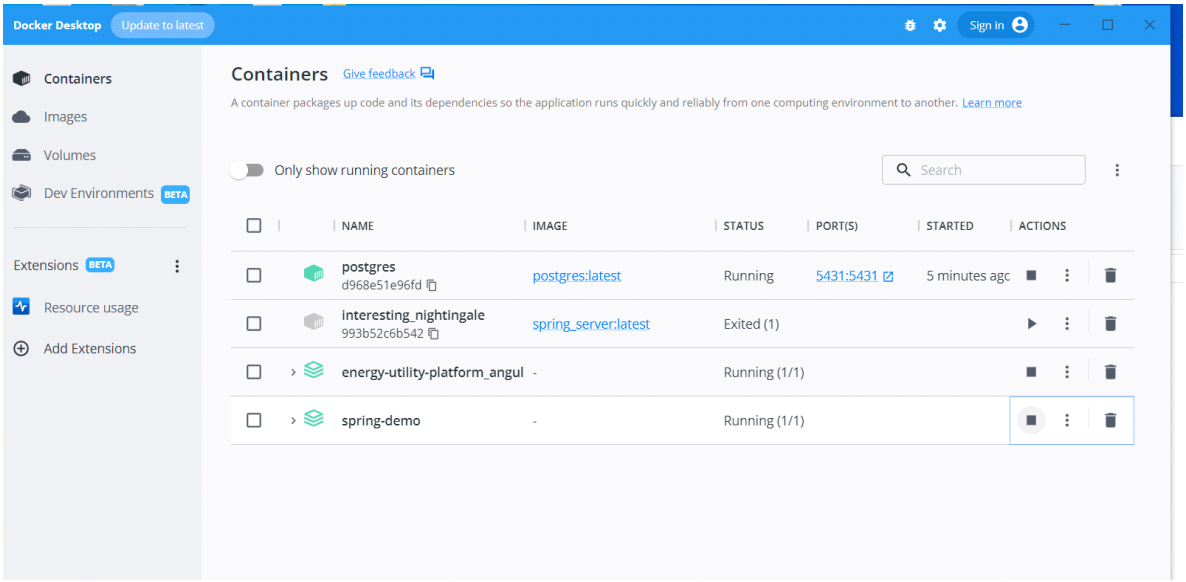


Back End:


package ro.tuc.ds2022;
@SpringBootApplication
@Validated
public class Ds2022Application extends SpringBootServletInitializer {
@Override
protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
return application.sources(Ds2022Application.class);
}
public static void main(String[] args) {
TimeZone.setDefault(TimeZone.getTimeZone("UTC"));
SpringApplication.run(Ds2022Application.class, args);
System.out.println("merge bine"+ Ds2022Application.class.toString());
}
}


@Id
@GeneratedValue(strategy = GenerationType.AUTO)
private long id;
private String name;
@OneToOne(fetch= FetchType.EAGER)
private Role role;
merge bineclass ro.tuc.ds2022.Ds2022Application

Docker:



Bibiliography:

https://dsrl.eu/courses/sd/materials/la11.pdf

https://dsrl.eu/courses/sd/materials/cicd-azure.pdf

http://www.tutorialspoint.com/hibernate

http://www.javatpoint.com/hibernate-tutorial/

https://maven.apache.org/

https://angular.io/docs

https://www.geeksforgeeks.org/angular-cli-angular-project-setup/

https://www.chartjs.org/docs/latest/

http://l-lin.github.io/angular-datatables/#/getting-started

https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css

https://fontawesome.com/docs

Spring Boot Tutorial - Bootstrap a Simple App | Baeldung

https://www.baeldung.com/spring-email