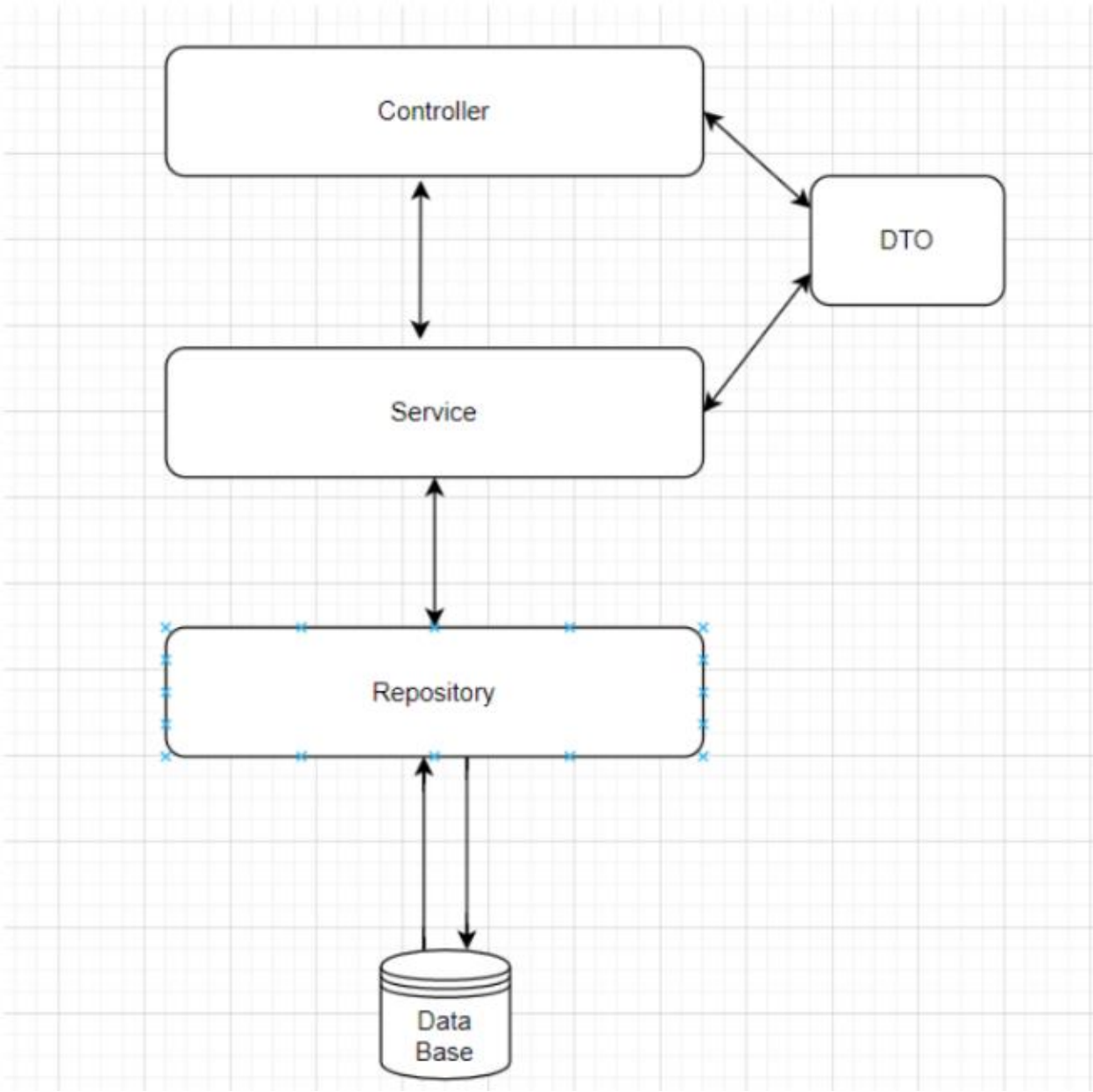
DISTRIBUTED SYSTEMS

Online Energy Utility Platform

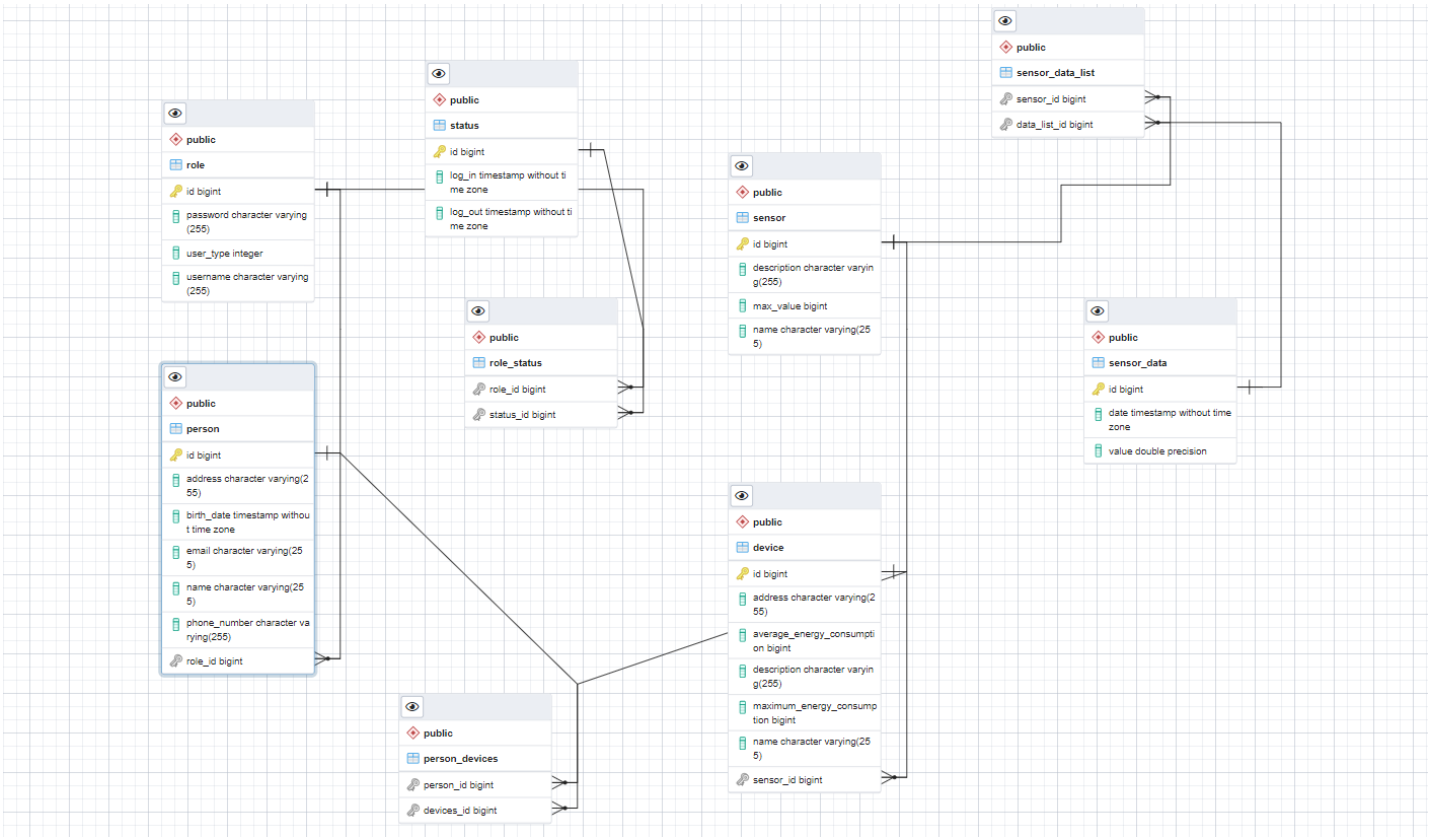
Name: Chis David Adrian

1. Conceptual architecture of the distributed system
2. DB design
3. UML Deployment diagram
4. ReadMe files containing build and execution considerations

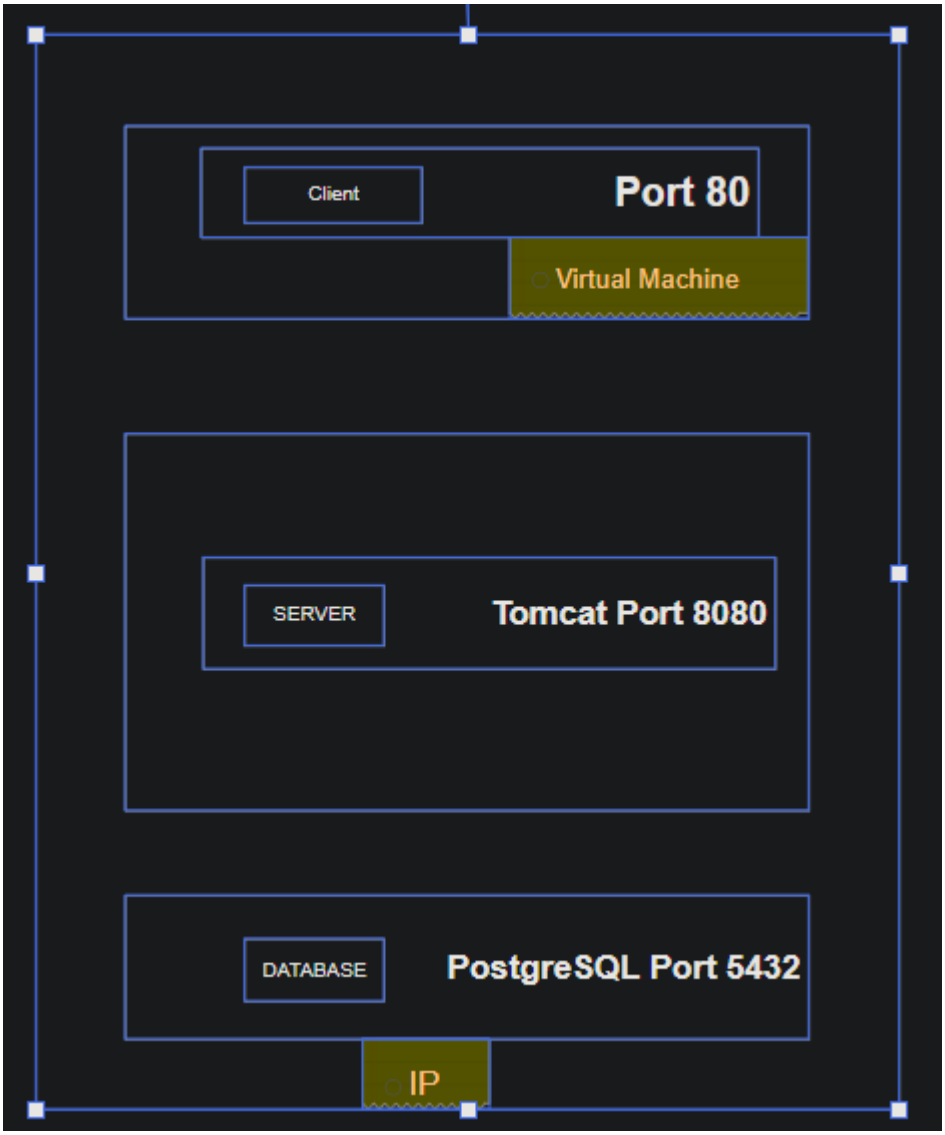
1. Conceptual architecture of the distributed system



DB Design



3. UML Deployment diagram



4. ReadMe files containing build and execution considerations

Aplicatia are la baza 2 parti , partea de BackEnd si partea de FrontEnd

Aplicatia BackEnd (BE) este o aplicatie de sine statatoare scrisa in Java se bazeaza pe Spring Framework si ruleaza pe portul 8080 default, Este o aplicatie care serveste pe tip de server pentru aplicatia de frontEnd (FE) si in acelasi timp este si o aplocatie Client pentru DB Server.

Aplicatia BE face legatura dintre nivelul BD si Client ul astfel este structurata pe Layere si anume DB Model Repository Service Controller .

a. DB este primul nivel reprezentat de pachetul Model din aplicatie in care sunt mapate tabelele si legaturile dintre ele.

b. Repository: pachetul ce contine intergete care fac tranzitia de la obiectul efectiv la tabel prin intermediul ORM (Object-Relational Mapping)

c. Service: pachetul care contine logica aplicatiei folosind repository-urile

d. Controller: pachetul unde se face legatura dintre Front-End si Back-End, unde se afla metodele ce gestioneaza request-urile HTTP si folosesc service-urile pentru a apela la baza de date.

Componenta Front-End va rula pe portul 3000 si aceasta va reprezenta interactiunea dintre BackEnd si utilizatori. Pentru obtinerea datelor din baza de date, in Front-End vom face request-uri HTTP catre Back-End si vom obtine datele pe care le vom prelucra mai departe. Aceasta componenta a fost prelucrata utilizand libraria ReactJS iar limbajul a fost Typescript. Aici am impartit fiecare logica in cate o componenta care este continuta in cate un pachet separat. Astfel, avem urmatoarele pachete:

Components : in care sunt incluse componentele reutilizabile in aplicatie ex tabele grafice etc

Pages : in care sunt structurate fiecare Pagina din aplicatie

Services : IN care sunt introduse serviciile FE coresponedente fiecarui Controller din BE astfel se face o organizare a codului mult mai buna si duce la o intelegere cat mai buna. Apelurile catre BackEnd sunt facute utilizand Jquery (Ajax) pentru a face codul cat mai simplu de citit pt developer.

Utils in care am stocat constante sau metode des utilizate in aplicatie

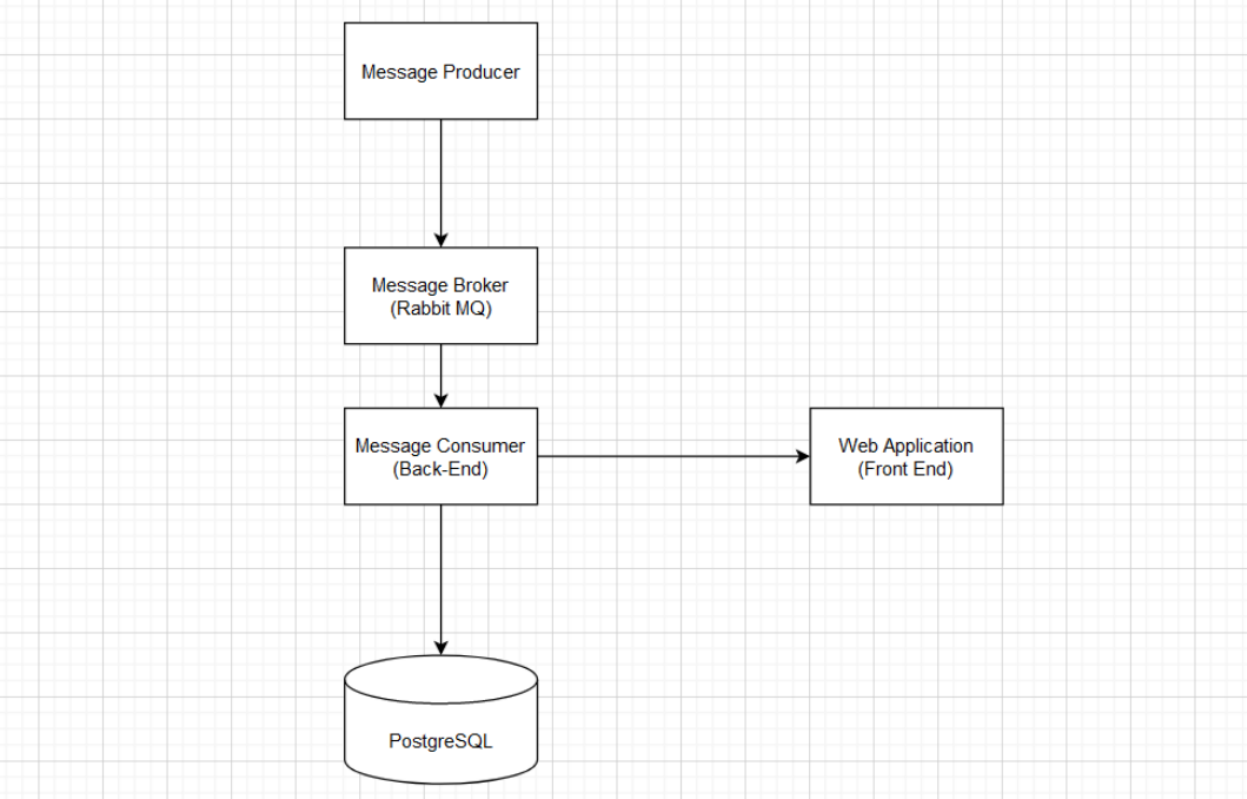
Enums in care s-au stocat enum urile folosite in aplicatie

Dto incare s-au stocat obiectele secundare folosite pentru transmiterea datelor din FE in BE

Model un pachet inc are sunt introduse cate o interfata pt fiecare model din BE astfel facanduse un match 1-1 cu BE ul

1.ARHITECTURA CONCEPTUALĂ A SISTEMULUI DISTRIBUIT

Arhitectura conceptuală reprezintă componentele din cadrul unui sistem informatic distribuit și modul în care acestea funcționează și interacționează între ele. Astfel că în cadrul acestei diagrame componentele de tip repository comunică în mod direct cu baza de date Postgres, (pentru căutare în baza de date, ștergere, modificare). Componentele de tip service sunt cele care folosesc obiectele de tip repository pentru a realiza diverse cereri ale clienților, iar obiectele de tip controller sunt cele care conțin URL-urile care se accesează pentru a îndeplini diverse cereri. Entitățile conțin datele care se stochează în baza de date, iar DTO-urile sunt obiectele folosite pentru a comunica cu serverul de front-end pentru a trimite sau a răspunde la diverse cereri.



RabbitMQ

Pe lângă aceste aplicații a mai fost nevoie de crearea unei noi aplicații desktop, în cadrul căreia citim măsurători specifice unui anumit senzor. Totodată în cadrul acestei aplicații se folosește fișierul application.properties ca și fișier de configurare și în cadrul acestui fișier se specifică ID-ul unui sensor pe care dorim să îl folosim la implementarea acestui proiect. Pentru a putea vedea aplicația de front-end este nevoie să se acceseze https://chis-david-ds.herokuapp.com, iar de acolo site-ul și funcționarea lui sunt foarte suggestive

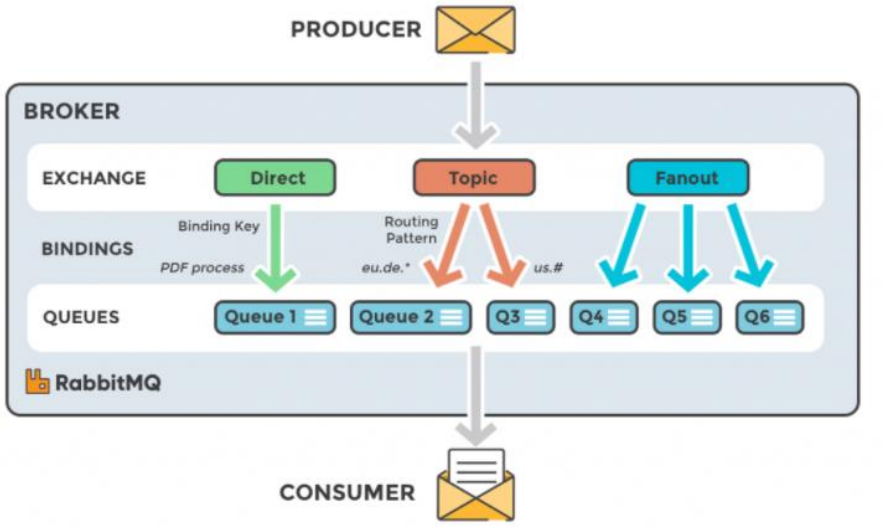
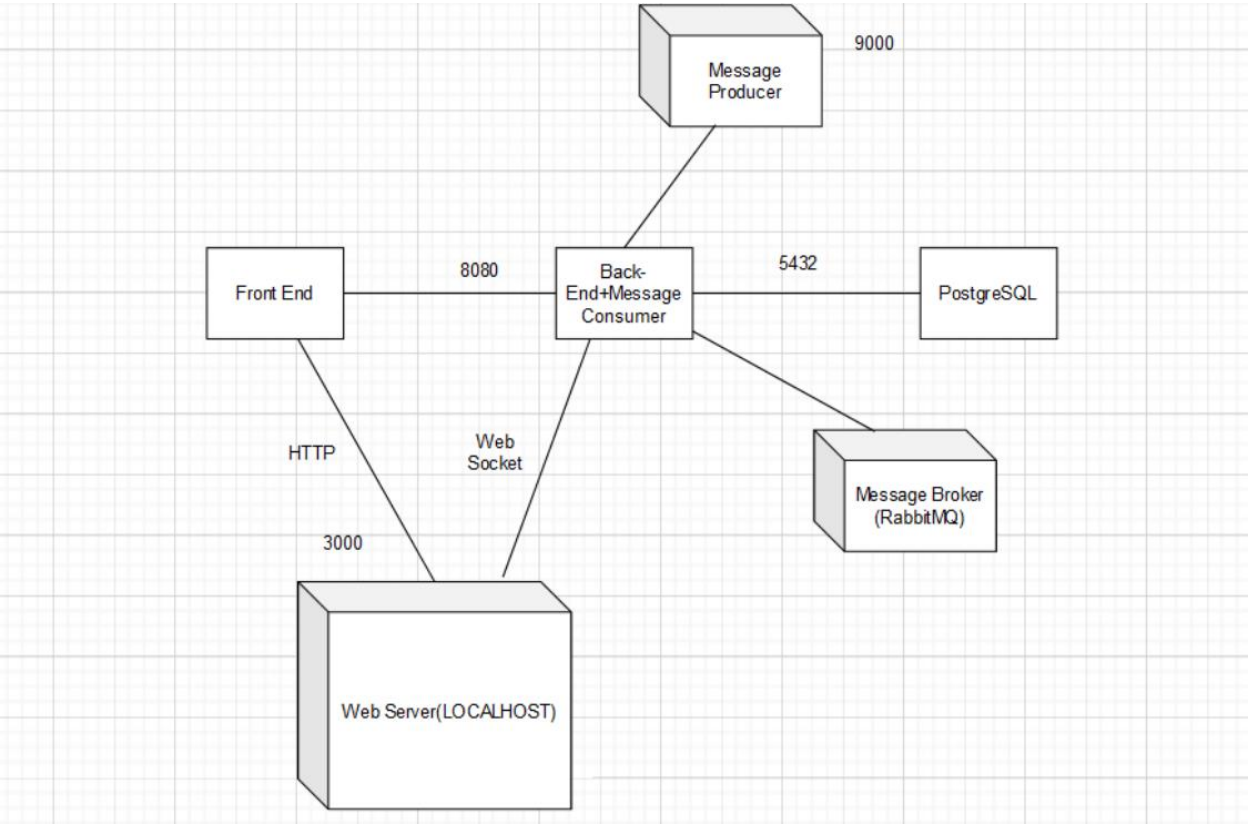


DIAGRAMA DE DEZVOLTARE UML

Diagrama de dezvoltare UML ilustrează nodurile din cadrul aplicației și modul în care acestea comunică între ele. Deoarece am făcut aplicația atât pe Heroku, cât și local am decis că voi pune diagrama de dezvoltare pentru ambele. LOCALHOST.



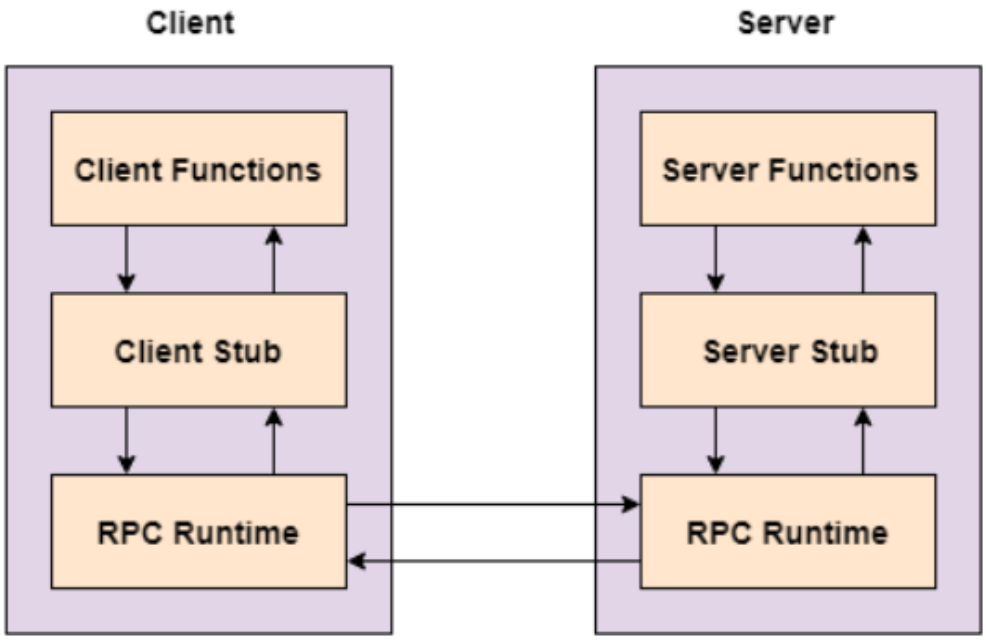
Remote Procedure Call (RPC)

O aplicatie RPC consta intr-un client si un server, serverul fiind localizat pe masina ce executa procedura Aplicatia client comunica prin retea cu procedura de pe calculatorul la distanta transmitind argumentele si receptionind rezultatele Clientul si serverul ≡ procese pe masini diferite RPC realizeaza comunicarea dintre ele prin socket-uri TCP/IP (uzual, UDP), via doua interfete stub (ciot)

Readme file containing build and execution considerations

În cadrul aplicației pentru assignmentul 3 am folosit protocolul de comunicare JsonRPC, astfel în aplicația inițială de backend am create o clasă service nouă în care am implementat o metodă care returnează toate valorile măsurate din baza de date. Clasa service este adnotată cu **@JsonRpcService** și are ca și parametru un string care reprezintă url-ul pe care noi o să-l accesăm pe partea de backend. Metoda o implementăm în pachetul de implementare, unde am avut nevoie de un repository și un mapper care să mapeze datele. Clasa este adnotată cu @Service și @AutoJsonRpcServiceImpl pentru implementarea propriu-zisă.

Pentru a putea transmite datele către aplicația de front, în cazul nostru, ne-am creat o clasă de configurări unde am creat un Bean care să exporte datele primite în partea de front , Bean-ul se numește AutoJsonRpcServiceImplExporter și returnează un exp care este un nou AutoJsonRpcServiceImplExporter.



Json-RPC este **remote procedure call protocol** encodat într-un JSON. Este similar cu XML-RPC care definește câteva date și comenzi. Spre deosebire de XML-RPC,JSON-RPC permite notificări(date care sunt transmise de la serevr fără să primească vreun răspuns), iar în cazul call-urilor multiple care sunt transmise de la server acestea sunt asincrone.

JSON-RPC funcționează prin trimiterea unei cereri către un server care implementează acest protocol. În acest caz, clientul este de obicei un software care intenționează să apeleze o singură metodă de la distanță. Mai mulți pramateri de intrare pot fi transferați metodei la distanță ca matrice sau obiect, în timp ce metoda în sine poate returna și date multiple de ieșire.

Date care sunt transmise de către server le primesc pe partea de front astfel, prin url-ul transmis http://localhost:8080/rpc. Accesăm url-ul și afișăm datele într-un tabel, datele care reprezintă ultimele 7 valori din ultima săptămână , apoi le putem selecta și afișa graficul cu aceste valori, precum și media.