Logo, icon

Description automatically generated

**Assignment 2**

~**Distributed Systems**~

*Student*

*Steau Iuliu Andrei*

*06 December 2022*

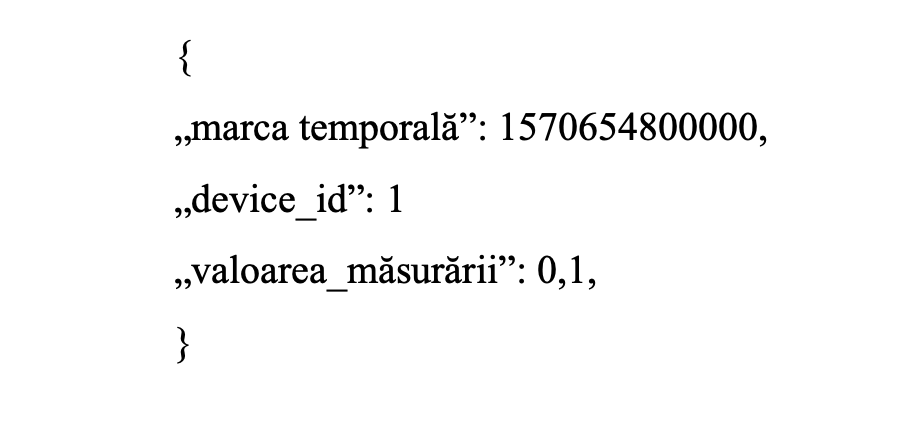
*Profesor Indrumator*

*Mirela Prata*

***Requirements***

Implementați o componentă pentru aplicația Assignment 1 bazată pe un middleware de broker de mesaje care adună date de la dispozitivele inteligente de contorizare, preprocesează datele pentru a calcula energia orară consumul și îl stochează în baza de date.

Un modul Smart Metering Simulator va fi producătorul de mesaje. Va simula un senzor prin citirea datelor de energie dintr-un fișier (sensor.csv - o valoare la fiecare 10 minute) și trimite date sub forma < timestamp, device\_id, measurement\_value > către Message Broker (adică,coadă). Timestamp-ul este preluat de la ceasul local, measurement\_value este citită din fișier și reprezintă energia măsurată în kWh, iar device\_id este unic pentru fiecare instanță a Smart Metering Device Simulator și corespunde id-ului dispozitivului unui utilizator din baza de date . Simulatorul de senzori ar trebui dezvoltat ca o aplicație independentă (adică, aplicație desktop). Fișierul sensor.csv poate fi descărcat de la https://dsrl.eu/courses/sd/materials/sensor.csv. Măsurătorile sunt trimise la coadă folosind următorul format JSON:



O aplicație Message Consumer va preprocesa datele pentru a calcula energia totală pe oră consumul și îl stochează în baza de date. Dacă consumul total de energie orar calculat depășește valoarea maximă a dispozitivului inteligent (așa cum este definită în prima tema), acesta notifică în mod asincron utilizatorul pe interfața sa web.

* 1. Cerințe funcționale:
* Brokerul de mesaje permite Smart Metering Device Simulator să acționeze ca producător de mesaje și trimiteți tupluri de date într-un format JSON.
* Componenta consumator de mesaje a sistemului procesează fiecare mesaj și notifică asincron folosind WebSockets aplicația client.
  1. Tehnologii de implementare:
* Utilizați următoarele tehnologii: RabbitMQ, WebSockets.

***Specificatia proiectului***

Scopul acestui proiect este de a realiza o aplicație prin care se ilustrează simularea unui senzor . Valorile pentru senzor se citesc dintr-un fișier csv. Clienții distribuitorului de energie au instalat contoare inteligente pentru fiecare dispozitiv la care este înregistrat consumul de energie. Fiecare senzor trimite periodic date către un server, sub formă (timestamp, senzor\_id, measurement\_value), unde timestamp este momentul în care măsurarea a fost efectuată și measurement\_value este valoarea contorului de energie care măsoară energia totală consumată de dispozitiv în kWh de la instalarea senzorului.

Proiectul este structurat în 3 module principale: Simulatorul de senzori, Coada si Consumatorul. Simulator de senzori citește date din fișiere (sensor.csv), o valoare la fiecare 10 minute. Modulul va conține un cronometru sincronizat cu ceasul local. Datele trimise respecta următorul format < timestamp, sensor\_id, measurement\_value >. Simulatorul de senzori este dezvoltat ca o aplicație autonomă (adică aplicație desktop) pentru a citi activitățile monitorizate de senzor din fișierul sensor.csv. Acesta este configurat ca un producător de mesaje și trimite datele cozii.

Coada este componenta care primește mesajele de la producător și le menține până când un consumator, în cazul nostru aplicația care se va ocupa de gestionarea mesajelor, este conectat.

Consumatorul este cel care recepționează mesajele de la coadă și le procesează înainte de a le stoca în baza de date. Acesta calculează energia pe oră și în cazul în care depășește valoarea maximă a energiei asociată senzorului trimite o notificare utilizatorului. Calculul energiei pe oră este realizat prin medierea consumului de energie și împărțirea valorii la intervalul de timp.

***Arhitectura conceptuala***

Pentru dezvoltarea aplicației au fost folosite următoarele tehnologii: pentru backend am folosit Java pentru a descrie și modela comportamentul obiectelor și Spring Boot pentru a putea efectua apeluri HTML. Pentru partea de frontend am folosit ReactJS pentru a putea imbrica elemente de css, javascript și html într-un singur mediu de dezvoltare. Pentru baza de date am folosit PostgreSQL . În plus, am folosit RabbitMQ pentru a putea trimite mesaje între simulatorul de senzori și consumatorul (backend-ul de la assigment-ul anterior).

REST API(sau RESTful API) este o metodă de comunicare care se bazează pe protocoalele existente HTTP. Acesta stă la baza comunicării dintre client și server. Astfel prin request-uri http de tip GET,POST, PUT,DELETE etc se fac anumite cereri către client în funcție de intenția utilizatorului.

Partea de frontend conține elementele de interacțiune cu utilizatorul. De aici adminii pot să efectueze CRUD-uri pe clientii și să vadă rezultatul în timp real afișat pe interfață, în tabele. Clientii își pot vedea dispozitivele asociate și consumul lor de energie pe zile, aceștia putând să selecteze ziua sau zilele pe care o/le doresc. . De asemenea a fost configurat un WebSocket pentru a trimite o notificare atunci când energia maxima a dispozitivului este depășită de energia pe oră.

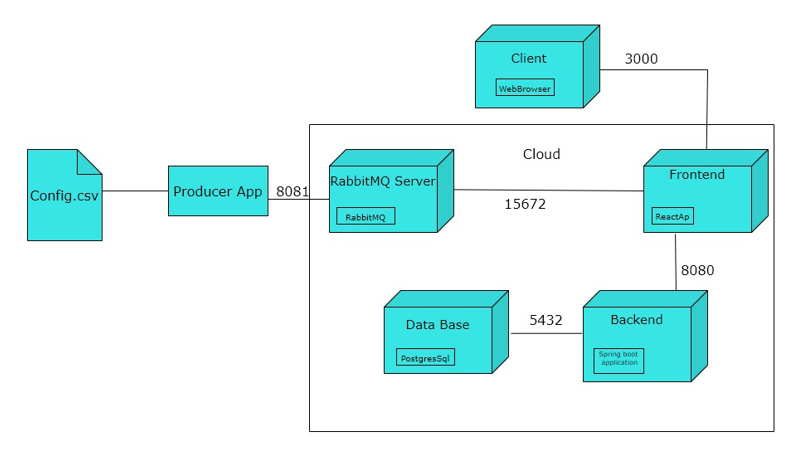
Partea de backend este structurată pe layere, informația fiind transmisă de la un layer la altul . Astfel avem nivelul inferior de repository care face legătura aplicației cu baza de date. Acesta leagă clasele de entity care modeleaza obiectele cu tabelele corespunzătoare din baza de date. Apoi avem nivelul de service în care se realizează logica aplicației. Aici, cu ajutorul layer-ului de repository vom realiza apelurile către baza de date și vom procesa datele care vor fi transmise nivelului de controller. Layer-ul de controller rezolvă cererile http pe baza layer-ului de service și a DTO-urilor, care sunt clase învelitoare pentru cele din entity, unde va fi realizată legătura cu elementele de JSON primite prin javascript de la frontend.

***Diagrama arhitecturii conceptuale***

Diagram

Description automatically generated

***Deployment diagram***



***Database diagram***

Diagram, schematic

Description automatically generated