SISTEME DISTRIBUITE

Tema 3

Documentaţie

Remote Procedure Call (RPC)

Nume: Ţoc Roxana-Ştefania

Grupa: 30463

Îndrumator de laborator: Toderean Liana

Cuprins:

1. Descrierea proiectului

2. Arhitectura conceptuală a sistemului distribuit

3. Diagrama UML de deployment

4. Build and execution considerations

1. Descrierea proiectului

Prima parte a proiectului constă într-o platformă online menită să gestioneze clienții, dispozitive inteligente echipate cu senzori de consum de energie și date monitorizate de la senzori. Sistemul poate fi accesat de două tipuri de utilizatori după un proces de logare: administrator și clienți. Administratorul poate efectua operațiuni CRUD pe conturile clientilor, despozitive si senzori. Fiecare dispozitiv are un senzor care îi monitorizează consumul de energie. Senzorul înregistrează periodic, la marcaje temporale fixe, tuple de forma <timp, consum de energie>, unde consumul de energie este un contor care măsoară în kWh energia totală consumată de dispozitiv de când a fost pornit. Fiecare client își poate vizualiza dispozitivele, și consumul lor de energie actual și istoric, precum și consumul total de energie al dispozitivele lor.

A doua parte presupune implementarea unui sistem bazat pe un message broker middleware care adună informațiile de la senzori și le procesează înainte de a le stoca în baza de date la senzorul corespunzător. Dacă aplicația care consumă mesajele și care preprocesează datele detectează o activitate anormală din punct de vedere al consumului, clientul senzorului corespunzator va fi notificat in mod asincron.

Conditia care trebuie indeplinita este ca diferenta dintre doua masuratori consecutive sa nu depaseasca valorea maxima cu care a fost implementat senzorul.

Pentru a treia parte clienții au aparate electrocasnice inteligente care pot fi controlate de la distanță folosind

apel de procedură la distanță (RPC). Fiecare astfel de dispozitiv poate comunica cu serverul care va afisa pe un grafic consumul istoric de energie al clientului pentru d zile in trecut.

2. Arhitectura conceptuală a sistemului distribuit

Arhitectura sistemului este o arhitectura de tip client-server în care aplicația client poate executa request-uri pentru a primi resurse de la aplicația de tip server.

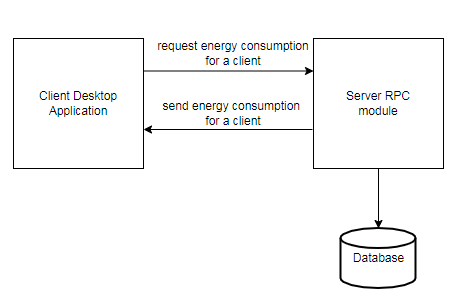
Aplicația de backend este construita cu ajutorul serviciilor Java Spring REST și are o arhitectură bazată pe layere, și anume: layerul cel mai apropiat de baza de date, cel de repositories facilitează accesul la baza de date, layerul enitities corespunde unui tabel din baza de date relațională, iar fiecare instanță a unei entități corspunde unui rând din baza de date, layerul de services reprezintă layerul de business logic care transforma Data Transfer Objects în enitități și invers, layerul DTO conține combinații ale diferitelor entități, iar layerul de controller conține funcționalitățile aplicației.

Aplicația de frontend care creează interfața grafică a fost realizată cu ReactJS, iar prin intermediul acesteia se trimit request-uri către server.

Pentru comunicarea asincrona s-a folosit o arhitectura message oriented în care mesajul este trimis prima data către o coadă de către o aplicație standalone, iar abia apoi este livrat către un consumer conectat la coada respectivă. Consumerul este implementat în backend și primește mesaje pe care le preprocesează și stochează în baza de date monitored\_value. Fiecare mesaj este livrat și extras de un singur consumator, iar dacă nu există consumatori mesajele sunt reținute în coadă până când se conectează un consumator. Pentru această comunicare s-a folosit RabbitMQ.

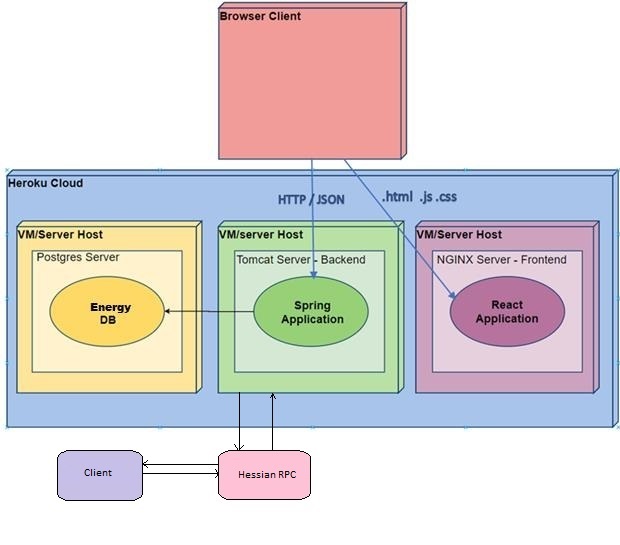
Pentru a actualiza partea de client, respectiv pentru a primi notificări în frontend s-au folosit websocket-uri, care oferă o comunicare bi-direcțională, comunicare real-time și folosește o singură conexiune per client.

Pentru a treia parte din proiect s-a adaugat în partea de backend un modul de Hessian RPC care să poată comunica cu aplicația client. Aplicația client este o aplicație desktop construită folosind arhitectura MVC și dispune de o interfață grafică dezvoltată folosind JAVA SWING. Această aplicație primește de la aplicația de backend o matrice care contine pentru fiecare zi valorile monitorizate ale clientului și le afisează în interfața grafică.



3. Diagrama UML de deployment

Modulele aplicației finale vor fi stocate în trei servere principale, unul pentru baza de date, unul pentru aplicația de frontend și unul pentru aplicația de backend. Aceste trei părți comunică între ele. Partea de backend va comunica în mod direct cu baza de date, trimițând date pentru a fi stocate sau extrăgând date prin intermediul interogărilor. Totodata, backendul comunica cu partea de client prin Hessian RPC si rezultatele vor fi afisate pe un grafic.



4. Build and execution considerations

Pentru SensorSimulator:

* Java 15;
* RabbitMQ pentru coada
* Mediu de dezvoltare: IntelliJ
* Rulare: se rulează fișierul *SensorApplication*

Pentru aplicatia client:

* Java 15;
* Hessian RPC pentru comunicarea cu serverul
* Mediu de dezvoltare: IntelliJ

Pentru Backend:

* Java 15;
* Server PostgresSQL pentru baza de date (v13.0.1) + pgAdmin, se creează o nouă bază de date;
* Configurarea unui server web–Apache Tomcat;
* Mediu de dezvoltare: IntelliJ
* Rulare: se rulează fișierul *Ds2020Application*

Pentru Frontend:

* Instalare pachet npm
* Rulare: comenzi: *npm install, npm start*

Rulare cu heroku:

* <https://react-demo-ds2021-roxana.herokuapp.com/>
* https://spring-demo-ds2021-roxana1.herokuapp.com/