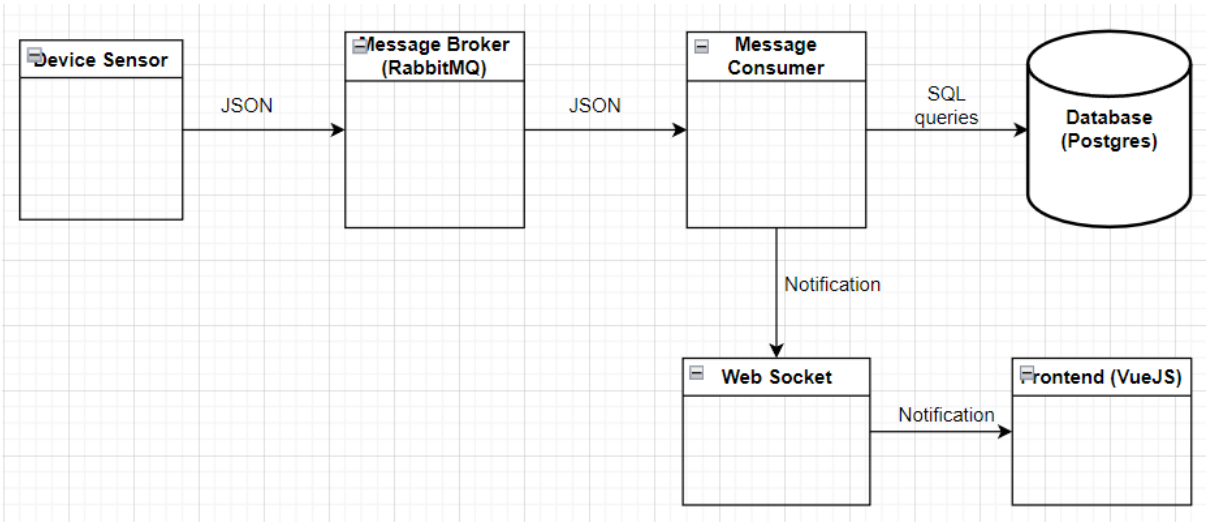
Saulea Alessandro, grupa 30242

Sensor Monitoring System and Real-Time Notification – documentatie

1. Arhitectura conceptuala



Aplicatia de mai sus, integrata in aplicatia de la assignment-ul 1, este alcatuita din: *Device Sensor Simulator*, *Message Broker (RabbitMQ)* si *Message Consumer*.

Pentru comunicarea intre *Device Simulator* si *Message Consumer*, este nevoie sa cream un *ConnectionFactory* ce creeaza conexiunile dintre sender si receiver in RabbitMQ, si este caracterizat de un URI de tipul *amqps://user:password@host/user*, unde *host* este adresa de transmisie, iar *user* si *password* sunt credentialele folosite pentru logarea user-ului la host. *ConnectionFactory* creeaza folosind metoda *newConnection()*, o conexiune propriu zisa (*Connection*), care creeaza folosind metoda *createChannel* un canal (*Channel*) in care este declarata coada folosita pentru stocarea temporara a mesajelor (numita *metering_queue*), prin metoda *declareQueue()*.

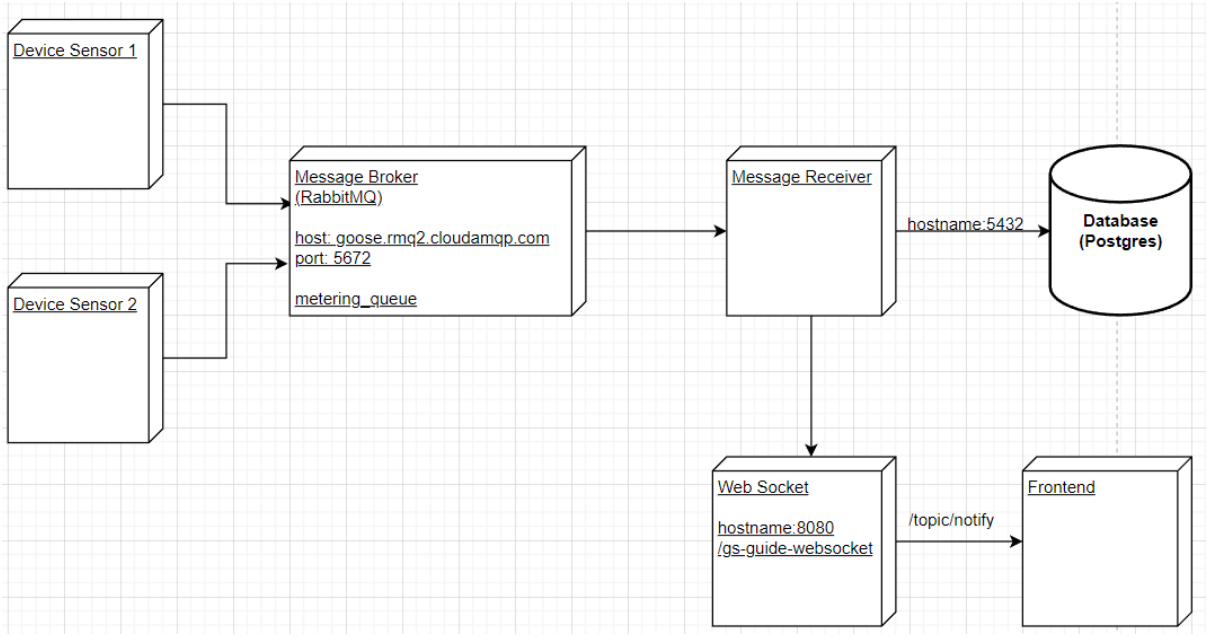
Device Sensor Simulator va fi Message Producer/Sender al aplicatiei. Primeste ca si parametru/argument id-ul device-ului ce urmeaza a fi simulat, citeste informatiile dintr-un csv *sensor.csv* ca si masurari ale consumului (exprimat in kWh) si transmite id-ul device-ului, valoarea citita din fisier si cu timestamp-ul intr-un singur JSON catre Message Broker (queue), cu metoda *basicPublish()*, transmitand JSON-ul sub forma de bytes, cu metoda *writeValueAsBytes* din *ObjectMapper*. Id-ul device-ului este transmis din linia de comanda in fisierul config.properties ca si proprietate a aplicatiei (Properties). De fiecare data cand se transmite un JSON, se foloseste id-ul citit din fisierul de configurare. Este setat cate un delay de o secunda (1000 ms) dupa fiecare transmisie pentru evitarea supraaglomerarii cozii. Senzorul este dezvoltat ca si aplicatie separata de aplicatia de backend. La final, dupa ce a citit toate valorile din fisierul csv, inchide canalul de comunicatie (*channel.close()*) si conexiunea cu broker-ul (*connection.close()*), si isi opreste executia.

Message Consumer/Receiver este integrat in aplicatia de backend. Acesta extrage byte-urile din coada RabbitMQ, extrage JSON-urile din acele bytes cu metoda *readValue* din acelasi *Object Mapper* calculeaza consumurile pe ora in functie de informatiile citite din JSON-uri, si le stocheaza in baza de date. Daca consumul de curent pe ora depaseste valoarea maxima a consumului pe ora admisa de device, Consumer-ul transmite utilizatorului device-ului in mod asincron o notificare in acest sens folosind web socket. Functionalitatea pe care o indeplineste consumer-ul la fiecare primire de JSON este configurata cu metoda *basicConsume*.

Websocket-ul este configurat in backend cu endpoint-ul */gs-guide-websocket* prin metoda *registry.addEndpoint*, iar broker-ul de websocket se afla la adresa */topic*. Adresa la care receiver-ul transmite notificarile este */topic/notify*, iar notificarile sunt distribuite in functie de proprietarii device-urilor corespunzatoare si filtrate in frontend dupa utilizatorul curent logat in aplicatie. De aceasta configurare se ocupa clasa *WebSocketConfig*, adnotata cu *@Configuration* si *@EnableWebSocketBroker*. Clientul se conecteaza la socket-ul configurat in backend folosind un protocol STOMP (Simple Text Oriented Messaging Protocol), si se aboneaza (subscribe) la */topic/notify*, unde primeste notificarile din partea receiver-ului, pe care le filtreaza in functie de utilizatorul curent. Teoretic backend-ul trimite notificari la toti clientii care acceseaza pagina web, dar vor vedea doar notificarile ce privesc device-urile lor. Pentru lucrul cu socket-uri in Javascript a fost nevoie de librariile *sockjs-client* si *stompjs*.

2. Diagrama de deployment UML

Aplicatiile de backend, frontend, baza de date, Message Producer (Device Sensor Simulator), Message Broker (RabbitMQ) si Message Consumer (integrat in aplicatia de backend) sunt procese separate si comunica intre ele in retea folosind diverse porturi si protocoale.



A se observa faptul ca unul sau mai multe instante de *Device Sensor Simulator* diferentiate dupa device ID transmit date catre Message Broker.

3. Bibliografie

- https://niruhan.medium.com/how-to-add-a-config-file-to-a-java-project-99fd9b6cebca
- https://howtodoinjava.com/jackson/java-8-date-time-type-not-supported-by-default/
- https://www.baeldung.com/jackson-deserialization
- https://spring.io/guides/gs/messaging-stomp-websocket/
- https://www.youtube.com/watch?v=vIxaVlHdpJU
- https://www.rabbitmq.com/tutorials/tutorial-one-java.html
- Shttps://www.cloudamqp.com/docs/java.html