Distributed systems

Assignment 2

Asynchronous Communication
Sensor Monitoring System and Real-Time Notification

Pasc Andreea-Madalina
30441

1. Requirements:

The clients of the energy distributor have installed smart meters for each device registered to measure its energy consumption. Each sensor sends data to a server periodically, in the form (timestamp, sensor_id, measurement_value), where timestamp is the time instance when the measurement was made and measurement_value is the value of the energy counter measuring the total energy consumed by the device in kWh since the sensor was installed. Implement a system based on a message broker middleware that gathers data from the sensors and pre-processes them before storing them in the database. If the queue consumer application that preprocesses the data detects a measurement power peak that exceeds the sensor maximum threshold (i.e. sensor maximum value measure in kW defined in Assignment 1) it notifies asynchronously the client on its web interface. To compute a power peak, the instantaneous power in a measurement interval is computed by averaging the energy consumption and dividing the value to the time interval. $Ppeak(t1,t2) = (measurementvalue(t2) - measurementvalue(t1)) / (t2 - t1) < MAXvalue$
A Sensor Simulator will simulate a sensor that reads data from files (sensor.csv), one value at every 10 minutes. The module will contain a timer synchronized with the local clock. The module sends data in the form < timestamp, sensor_id, measurement_value > to the message broker. The timestamp is taken from the local timer, the measurement_value is read from the file at the corresponding index, representing the energy measured in kWh, and the sensor_id is unique to each instance of the Sensor Simulator and corresponds to the sensor ID associated to a device of a client from the Energy Database. The sensor simulator should be developed as a standalone application (i.e. desktop application) to read the sensor monitored activities from the file sensor.csv, configured as a message producer and send the monitored sample data to the queue defined. The file sensor.csv can be downloaded from sensor.csv. The measurements are sent to the queue using the following JSON format:

{ “sensor_id”: “1” -> Long,

“timestamp" : 1570654800000, -> Long

“measurement_value” : 0.1 -> Float

}

2. Implementation

a. Producer:

It is a stand-alone spring boot application in which the measurements from sensor.csv are read converted into JSON object containing the sensor id (given in the main class), the measurement value and the timestamp (in milliseconds).

Every JSON object is transmitted to the queue from the cloud, from 10 to 10 minutes.

In the main class is done the connection to the cloud, by entering the credentials of the cloud account, containing the host, password to access the account.

b. Queue:

The RabbitMQ queue is built directly on cloud. I have used CLOUDAMPQ to store the queue. Here I created an account and the messages received from the producer are saved in the queue.

c. Consumer:

The consumer class was created on the backend part of the assignment1.

The connection with the cloud queue, by adding the credentials was done in application.properties.

When the peak value is greater than the maximum value of the specific sensor (determinated by the id) there is send a notification to the client in the frontend application, by using websockets.

d. Websockets:

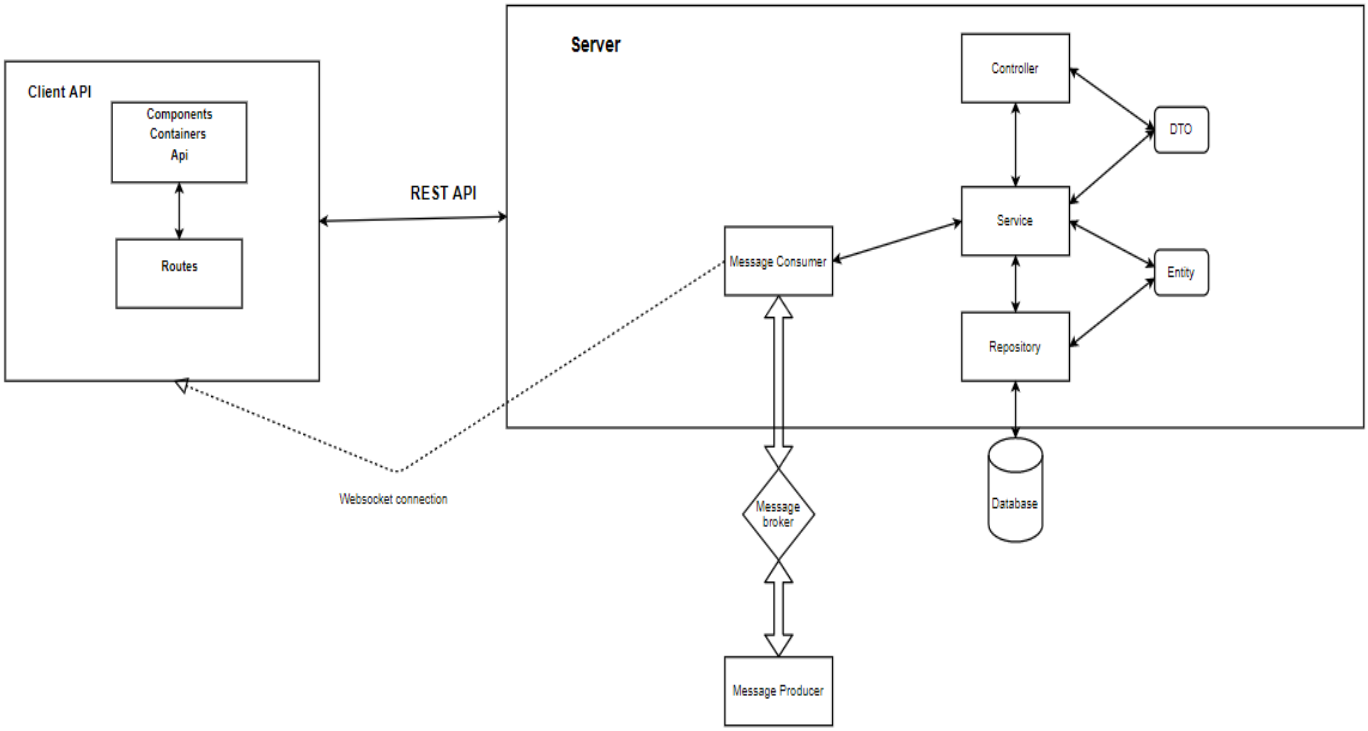
The notification from backend was send from the consumer class, by using the Simp Messaging Template library. The method convertAndSend from this library sends to the destination page the wanted message:

simpMessagingTemplate.convertAndSend("/consumptions/message", “Peak was exceeded!”)

There was necessary also a WebsocketConfig class, containing the configurations and the dependency in the build.gradle “org.springframework.boot” and “spring-boot-starter-websocket”.

In frontend the messages from the backend part were received and displayed with the help of some functions from the “sockjs-client” and “stompjs” libraries.

3. Conceptual architecture of the distributed system



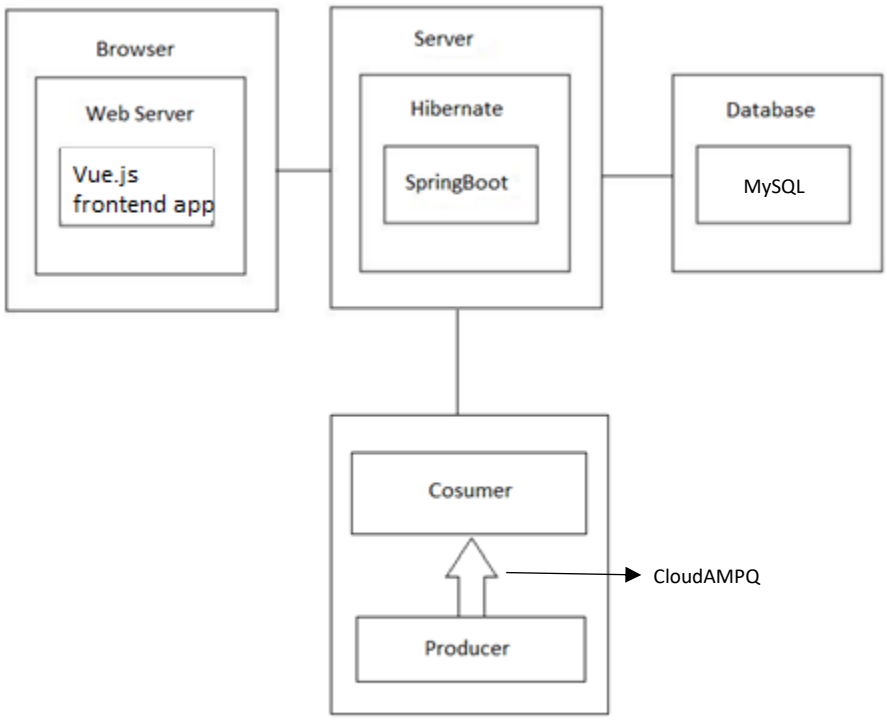
4. UML Deployment diagram

The deployment of the backend started by creating a new app in Heroku, called “ds2020-30441-pasc-andreea-be”. After pushing the backend on gitlab, the name of the application from Heroku and the key were provided in the yml file. Also, in the variables there was created the connection with the deployed data base.

The deployment of the frontend started by creating a new app in Heroku, called “ds2021-30441-pasc-andreea-fe”. After pushing the frontend on gitlab, on a new created repository, the name of the application from Heroku and the key were provided in the yml file. Also, in the connection with the backend is mandatory.

In order to run the deployed application, the address where the frontend is found is: https:// ds2021-30441-pasc-andreea-fe.herokuapp.com/#/.

The notifications received in fronted with websockets are displayed only on the client page of the frontend.


Browser
Web Server
Vue.js
frontend app
Server
Hibernate
SpringBoot
Database
MySQL
Cosumer
CloudAMPQ
Producer