Sensor Monitoring System and Real-Time Notification

Student: Vaduva Calin-Liviu

Grupa: 30644

Table of content

1. Assigment Specification

A **Smart Metering Device Simulator** module will be the Message Producer. It will simulate a sensor by reading energy data from a file (sensor.csv - one value at every 10 minutes) and sends data in the form < timestamp, device_id, measurement_value > to the Message Broker (i.e., the queue). The timestamp is taken from the local clock, the measurement_value is read from the file and represents the energy measured in kWh, and the device_id is unique to each instance of the Smart Metering Device Simulator and corresponds to the device_id of a user from the database (as defined in Assignment 1). The sensor simulator should be developed as a standalone application (i.e., desktop application). The file sensor.csv can be downloaded from https://dsrl.eu/courses/sd/materials/sensor.csv. The measurements are sent to the queue using the following JSON format:

{
“timestamp": 1570654800000,
“device_id”: “5c2494a3-1140-4c7a-991a-a1a2561c6bc2”
“measurement_value”: 0.1,
}

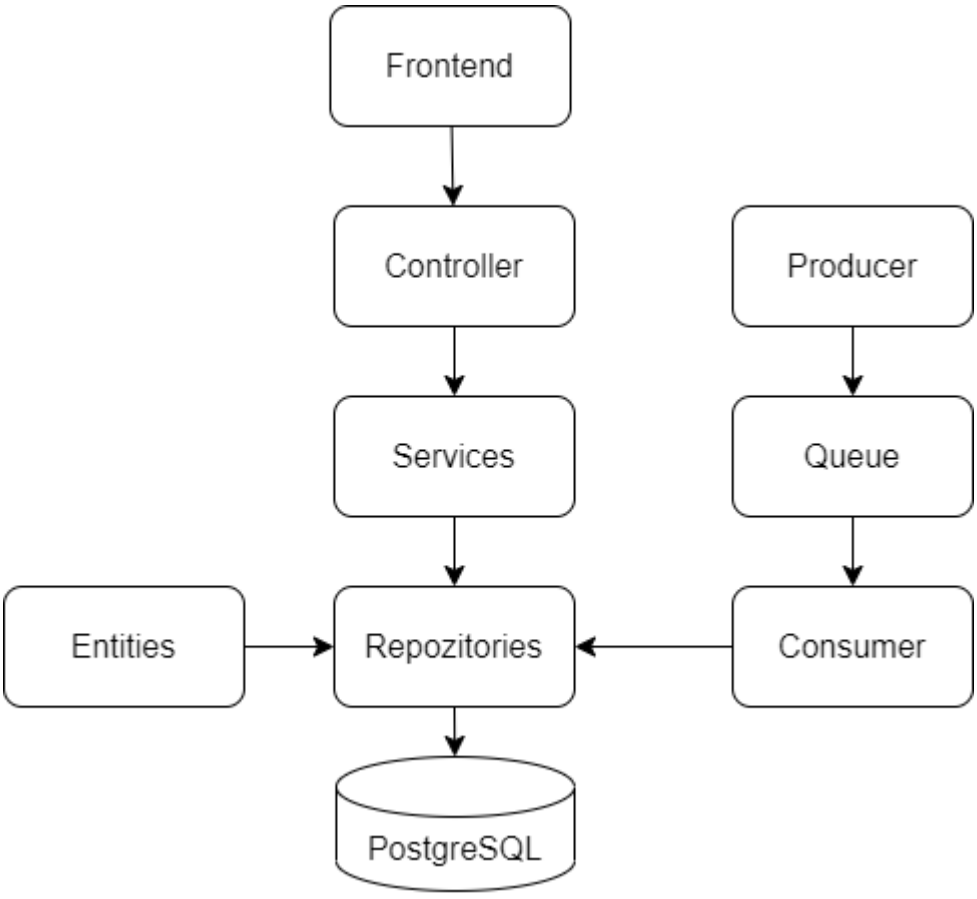
A Message Consumer application will pre-process the data to compute the total hourly energy consumption and stores it in the database. If the computed total hourly energy consumption exceeds the smart device maximum value (as defined in Assignment 1) it notifies asynchronously the user on his/her web interface.

Functional requirements: - The message broker allows Smart Metering Device Simulator to act as messages producer and send data tuples in a JSON format.

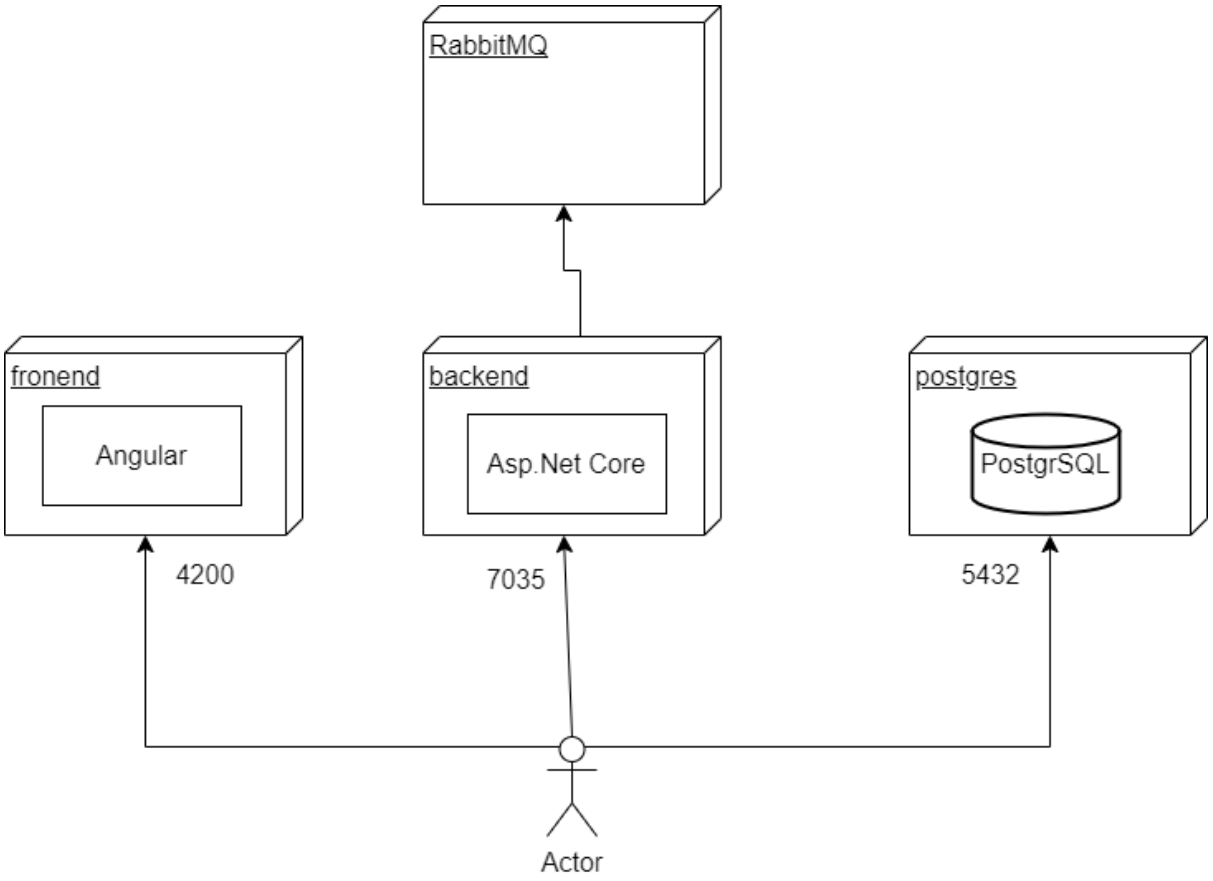
- The message consumer component of the system processes each message and notifies asynchronously using WebSockets the client application.

2. Architecture design

For implementing the requirements I used Docker for the local deployment of the application, I created an image each for backend, frontend and postgres and I ran the application in a docker container. If the running process is started, the application can be accessed from the browser at http://localhost:4200 without running the application from the IDE. For the microservices of the producer which read value form the sensor and send it to the consumer I used RabbitMQ on AQP. In the backend I created a class for the consumer which takes data from the Queue and put the information in the database. For the clients connection I created a Web socket with SignalR, so when the energy recived exceeds energy limit the app sends a message through web socket for the client.



3. UML Deployment



4. Database design

For the database design I created a table for the users with the fields id for identification, name, role for identification between Client and admin, password and username for login.

For the table devices I created the Id as primary key, and fields like description, address, the maximum consumption of a device and the userId which is for the identification of the owner of the created device.

The table Energy is for the energy consumption of each device it has a Id, timestamp for the day and hour at which the consumption was taken and the deviceId for identification of the device.