## Objective

To implement an Energy Management System using a message broker middleware. This microservice collects data from smart metering devices, and sends it to a cloud topic storing hourly energy consumption, in its own database. The synchronization between the databases of the Device Management Microservice and the new Monitoring and Communication Microservice is done through an rabbitMq topic for device changes.

To simulate smart meter data, a Smart Metering Device Simulator application is implemented as the Message Producer. This simulator reads energy data from a sensor.csv file, where each line represents a timestamped measurement (taken every hour) in the format <timestamp, device\_id, measurement\_value>. The timestamp is obtained from the local clock, and the device\_id is unique to each instance of the simulator, corresponding to the device\_id of a user from the database. The Smart Metering Device Simulator is developed as a standalone desktop application.

## Implementation

The system's implementation utilizes RabbitMQ queues to facilitate synchronization between the Device Service and the Monitoring Service. Two distinct queues, named "devices" and "readings," are employed for these purposes. The "devices" queue is responsible for transmitting changes from the Device Service to the Monitoring Service, while the "readings" queue is dedicated to conveying data read from the CSV file by the Smart Metering Device Simulator to the Monitoring Service.

Upon receiving data from the simulator, the Monitoring Service processes the information and persists the relevant messages into a designated table. If a measured value exceeds a certain threshold, a notification is dispatched to a private topic, "/user/" + user?.id + "/queue/messages," corresponding to the connected user. This ensures that users receive real-time notifications tailored to their specific devices.

WebSocket functionality is pivotal to this real-time communication, with the frontend client playing a crucial role in receiving messages from the server. The notifications are displayed within a notification component, enhancing the user experience.

To streamline communication between the Device Service and the Monitoring Service, modifications have been made to the Device Service to incorporate a new queue named "devices." Each message transmitted through this queue is equipped with an "action" property, delegating actions such as create, update, or delete. This ensures the seamless management of data within the respective tables.

On the frontend, users are empowered to visualize the power consumption of devices they own. A graph provides a comprehensive overview, detailing power consumption at each hour of the day, using data sourced from the CSV reader. The admin, on the other hand, has the privilege to view corresponding graphs for every user, providing a comprehensive perspective on overall energy consumption patterns.

The user experience is enriched through notifications, particularly when a measured value surpasses the maximum consumption threshold declared for a specific device. A separation of concerns approach is adopted to maintain clarity in the system's logic. Specifically, a useLogic component is implemented to handle data fetching and parsing, while dedicated UI pages showcase the processed data, ensuring a modular and maintainable design.

## UML Deployment diagram

O imagine care conține text, captură de ecran, proiectare

Descriere generată automat