Technical documentation of

Asynchronous Communicatio and Real\_time Notification

Student: Santa Bianca Tamara Teacher: Gabriel Antonesi

Group: 30443

# Conceptual Architecture of the Distributed System

## Overview

Energy Management System is a distributed system built with a microservices architecture. The system provides an easy and efficient way to monitor and manage multiple electronic devices connected to the Internet in an IoT fashion. Those devices continuously send data related to their energy consumption to the application, which centralizes the data and checks if their energy consumption is within the specified parameters. This is done in an asynchronous fashion, that is every time a new measurement is taken, it is immediately sent to the application. The application is scalable in the sense that any number of devices can be connected to it and send data at the same time, and it is in real time since the data is received and processed instantly.

The app has 3 main resources, Users, Devices and Measurements. The resource for Users holds information about the users that are registered in the application, the Device resource holds information about all the devices that are registered in the application and the Measurement resource is used for storing each piece of individual data that is sent from the devices to the application. The app supports two types of users, ADMINS and CLIENTS.

ADMINS have elevated privileges and can perform CRUD operations on all devices. They can also update some of the fields of users and they can effectively and immediately delete any user account. Admins also have the job of assigning and unassigning devices from users.

USERS have restricted privileges and can only view the devices that have been assigned to them by Admins and see their notifications regarding whether their devices have exceeded the maximum energy consumption allowed for them.

## 1.2 Backend

When it comes to speaking about backend, 2 new concepts were introduced in our project, RabbitMQ

**RabbitMQ** is an open-source message broker software that plays a pivotal role in facilitating communication and data transfer between various components within a system. In the context of the Energy Management System project, RabbitMQ serves as a central hub for message exchange between the Smart Metering Device Simulator, the Monitoring and Communication Microservice, and potentially other services involved. RabbitMQ acts as an intermediary, or message broker, ensuring seamless communication between different parts of the system. It follows a message queuing mechanism where messages are sent by producers (in this case, the Smart Metering Device Simulator) to a queue and then consumed by consumers (like the Monitoring and Communication Microservice). One of the significant advantages of using RabbitMQ is its support for asynchronous communication. This means that components don't need to communicate directly; they simply publish messages to a queue, and interested parties can asynchronously consume these messages when they are ready, leading to decoupled and more scalable systems.

In this project, RabbitMQ is employed to gather data from smart metering devices. The Smart Metering Device Simulator generates measurements at regular intervals and sends these measurements in JSON format to a queue in RabbitMQ. These messages contain essential data points like timestamps, device IDs, and measurement values.

More specifically, the code establishes a connection to RabbitMQ using a ConnectionFactory and creates a channel for communication. It declares a queue (smart\_meter\_data) where measurements from the Smart Metering Device Simulator will be sent. A DeliverCallback is defined to handle incoming messages from the queue. Upon receiving messages, it extracts the data and passes it to the processAndStoreMeasurement method. This method processes the measurements, calculates hourly energy consumption, and performs checks against defined thresholds for notifications.

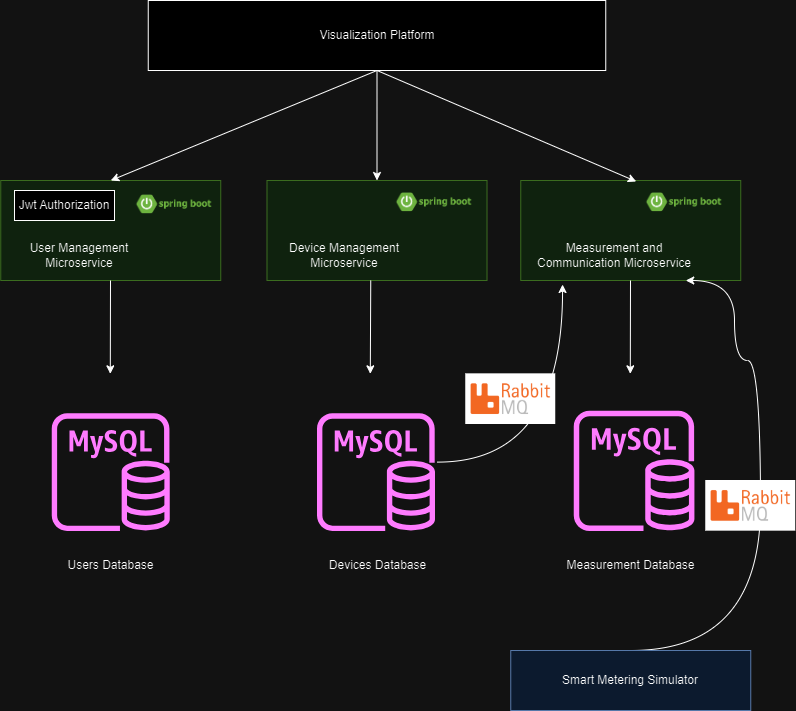
For the producer part, I created a java appplication. The Java app. serves as a simulated Smart Metering Device, generating mock energy consumption measurements and sending them to a RabbitMQ message queue. This emulation mirrors the behavior of an actual smart meter, providing test data for the system. The script starts by reading a CSV file (sensor.csv) that presumably contains energy consumption data or sensor readings. The values from this file are processed and normalized to simulate realistic energy consumption data.

It constructs a JSON-formatted message containing a timestamp, a device ID (passed as a command-line argument), and a randomly selected measurement value from the processed CSV data. inside the loop, the script establishes a connection to a RabbitMQ running on CloudAMPQ. It creates a channel, declares a queue named (QUEUE\_NAME), and publishes the generated JSON message to this queue using the RabbitMQ library's functions. The producer script introduces a wait time (10 seconds) between message generations, simulating periodic measurements from a smart meter. This time delay ensures that the system isn’t overloaded with a high volume of messages and reflects the interval between actual measurements. The script expects a single command-line argument, which represents the device ID.

## 1.3 Frontend

Using React for the frontend and a communication microservices-based backend, users can visualize their energy consumption via intuitive line or bar charts, depicting hourly energy values [kWh] for a selected day chosen from a calendar interface. The backend communicates seamlessly with the frontend, receiving timestamps and user identifiers to deliver accurate consumption information."

## 2.1 UML Diagram



## 2.1 Deployment Diagram

