A black background with red lines

Description automatically generated

**<Assignment 3>**

**Analysis and Design Document**

**Student: Vasilachi Eugen**

**Group: 30644**

1. **Introduction**

This documentation outlines the design and implementation of an Energy Management System (EMS) that includes a frontend application and two microservices. The system is developed to manage users and their associated smart energy metering devices. Users can be of two types: administrators (managers) and clients. Administrators have the authority to perform CRUD operations on user accounts, smart energy metering devices, and manage the mapping of users to devices.

A Smart Metering Device Simulator application will be implemented as the Message Producer. It will simulate a smart meter by reading energy data from a sensor.csv file (i.e., one value at every 10 minutes) and sends data in the form < timestamp, device\_id, measurement\_value > to the Message Broker. The timestamp is taken from the local clock, 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. The device simulator should be developed as a standalone application.

The Monitoring and Communication Microservice will have a Message Consumer component that will process the measurements to compute the total hourly energy consumption and store it in the database. If the computed total hourly energy consumption exceeds the device defined maximum value it notifies asynchronously the user on his/her web interface.

Chat microservice and an authorization component will be implemented for the Energy Management System.

The authorization component should provide secured access of users to systems’ microservices.

The chat microservice should allow communication between the users and the administrator of the system, allowing them to ask questions and receive answers.

1. **System Overview**

The Energy Management System (EMS) comprises the following components:

* Frontend Application: A user interface that provides access to the EMS for both administrators and clients.
* User Microservice: A microservice responsible for managing user accounts.
* Device Microservice: A microservice dedicated to managing smart energy metering devices.

1. **User Roles**

The system defines two primary user roles:

* Administrator (Manager): Administrators have full control over user accounts, smart energy metering devices, and user-device mappings.
* Client: Clients can access and manage their own smart energy metering devices

1. **Functionality**
   1. **Administrator Functions**

Administrators can perform the following CRUD operations:

* Create, read, update, and delete user accounts.
* Create, read, update, and delete smart energy metering devices.
* Manage the mapping of users to devices, where each user can own one or more devices in different locations.
  1. **Client Functions**

Clients can:

* View and manage their own smart energy metering devices.
* Monitor the energy consumption of their devices.

1. **Microservice Architecture**

The system employs a microservice architecture to ensure scalability, modularity, and maintainability.

* 1. **User Microservice**

This microservice manages user-related data, including user accounts and their roles.

Endpoints:

* Create User
* Read User
* Update User
* Delete User
  1. **Device Microservice**

This microservice manages information related to smart energy metering devices.

Endpoints:

* Create Device
* Read Device
* Update Device
* Delete Device
* Map User to Device
* Unmap User from Device

1. **Non-Functional Requirements**

The EMS is designed to meet the following non-functional requirements:

* **Scalability:** The system must accommodate an increasing amount of user and device data as it grows.
* **Fast Response Time:** The application must provide quick responses to user requests.
* **Security:** User data and privacy must be protected through robust authentication and authorization mechanisms.
* **User-Friendly Interface:** The frontend should have an intuitive and user-friendly design for easy navigation.
* **Maintainability:** The codebase must be well-documented and modular, allowing for easy modification and updates in the future.

A diagram of a software system

Description automatically generated

**RabbitMQ Architecture**

A diagram of a rabbit

Description automatically generated

**Deployment Architecture**

A diagram of a computer

Description automatically generated

**8. Bibliography**

<https://docs.spring.io/spring-framework/reference/index.html>

<https://react.dev/learn>

<https://docs.docker.com/get-started/>

<https://docs.nginx.com/nginx/admin-guide/installing-nginx/installing-nginx-docker/>

<https://www.nginx.com/resources/wiki/start/topics/examples/full/>

<https://spring.io/guides/gs/messaging-stomp-websocket/>