**Energy Management System**

**Documentation**

Name: Deac Denisa Bianca

Group: 30444

Contents

[1. INTRODUCTION 3](#_Toc150200423)

[2. CONCEPTUAL ARCHITECTURE 3](#_Toc150200424)

[3. FUNCTIONAL REQUIREMENTS 3](#_Toc150200425)

[4. NON-FUNCTIONAL REQUIREMENTS 4](#_Toc150200426)

[5. UML DEPLOYMENT DIAGRAM 4](#_Toc150200427)

[6. IMPLEMENTATION DETAILS 4](#_Toc150200428)

[7. CONCLUSION 5](#_Toc150200429)

# INTRODUCTION

The Energy Management System is a distributed application that consists of a frontend and two microservices, designed to efficiently manage users and their associated smart energy metering devices. This system is designed for two distinct types of users, administrators (managers), and clients. Administrators possess the privilege to perform CRUD operations on user accounts, smart energy metering devices, and the mapping of users to devices. Clients, on the other hand, have the capability to view a list of devices associated with their account.

# 2. CONCEPTUAL ARCHITECTURE

The Energy Management System follows a layered architecture comprising entities, services, repositories, and controllers, ensuring a well-organized and maintainable structure. This architecture provides clear separation of concerns, allowing for easy development, testing, and scaling of the system.

The system consists of three main components:

* **Frontend Application:** The user interface different for both administrators and clients, enabling them to interact with the system for their own needs and possibilities.
* **User Management Microservice:** Responsible for managing user-related operations, including user account creation, updates, and deletions. This microservice maintains its own database (user\_ms), consisting of two tables, one for administrators and one for regular users. Both entities have an auto-generated id, username and password.
* **Device Management Microservice:** Manages device-related operations, such as device creation, updates, and deletions. Like the User Management Microservice, this microservice also maintains its own database (device\_ms). Each device has the following attributes: id, description, address and maximum hourly energy consumption. Beside these attributes there is also a clientId column used for the mapping between clients and their metric devices. Each client can posses many devices.

Frontend

UserMicroService

DeviceMicroService

# 

Device\_db

User\_db

# 3. FUNCTIONAL REQUIREMENTS

* Users can log in, and upon successful login, they are redirected to their respective pages based on their roles.
* **Administrator Role:**
  + Perform CRUD operations on user accounts.
  + Perform CRUD operations on devices.
  + Create a mapping between a user and a device, which automatically assigns the client ID to the device.
* **Client Role:**
  + Clients can view all devices associated with their account.

**Role-Based Access Control**

Users of one role will not be able to access pages corresponding to other roles, ensuring that the system enforces role-based security effectively. In case a users logs and tries to access the page of the opposite role( ex: admin instead of a client), by typing the specific URL, it won’t be possible. The user will be redirected to the login page.

# 4. NON-FUNCTIONAL REQUIREMENTS

* **Microservices**

The Energy Management System is divided into two microservices, ensuring modularity and scalability. Each microservice, namely User Management and Device Management, has its own dedicated database, user\_ms and device\_ms, respectively. This separation enhances data isolation and facilitates independent scaling of the microservices.

* **Security**

To enhance security, the system employs authentication mechanisms to restrict access to administrator pages. Session-based authentication is utilized, ensuring that only authorized users can access administrative functions.

# 5. UML DEPLOYMENT DIAGRAM

A diagram of a computer

Description automatically generated

# 6. IMPLEMENTATION DETAILS

The implementation of the Energy Management System relies on the following technologies:

* **Frontend Application:**
  + Developed using JavaScript-based frameworks with ReactJS, providing a dynamic and responsive user interface
  + For the authentication part, StorageSession was used. After each login, the role of the user was stored( administrator or client). For each component a PrivateRoute was used, which verified if the role in the StorageSession was the same with the one having access to the respective link.
  + The front-end application can be accessed through port 3000
* **Microservices:**
  + Built using REST architecture with Java Spring REST, ensuring efficient communication and data handling between microservices and the frontend.
  + The connection between frontend and backend was done via ports 8080 and 8081 and by using sugestiv URLs.
  + SpringBoot was used for both microservices, making the connection between microservices and databases much easier.
* **Database Architecture:**
  + Created using PostgreSql
  + The system utilizes separate databases for each microservice, user\_ms for user management, and device\_ms for device management. This separation guarantees data isolation and maintains data integrity.
* **Role-Based Data Mapping:**
  + The system ensures that administrators, when creating a new device, assign a clientID to that device. This role-based data mapping enforces the correct relationship between users and their associated devices. Beside it, each time a client is deleted, all their associated devices will be deleted as well.

Top of Form

# 7. CONCLUSION

In conclusion, the Energy Management System offers a streamlined solution for user and smart energy device management. It employs a well-structured layered architecture with separate microservices for User and Device Management, each having its dedicated database. This design ensures data isolation, scalability, and security.

The role-based access control system allows administrators to perform CRUD operations on user accounts and devices, facilitating user-device mappings. Clients can conveniently access their associated devices. The use of Java Spring REST for microservices and ReactJS for the frontend enhances responsiveness and scalability.

This documentation provides a comprehensive guide to understanding the system's architecture and functionality, making it a reliable choice for efficient energy management while maintaining data integrity and security.