Energy Management System

~Assignment 2~

Student: Oprean Dan

Group: 30243

Content

Introduction

* + Project Overview
  + System Objectives
  + Microservices Architecture
  + Frontend Application
  + Backend Microservices
  + Data Consistency

Conceptual Architecture

* + Overview
  + Frontend Architecture

UML Deployment Diagram

* + Container Overview
  + Image Overview
  + Deployment Connections

Functional and Non-Functional Requirements

* + Functional Requirements
  + Non-Functional Requirements

Conclusion

* + Achievements
  + Microservices Colaboration
  + Frontend User Experience
  + Deployment and Scalability
  + Functional and Non-Functional Requirements
  + Future Enhancements
  + Closing thoughts

Bibliography

1. Introduction
   1. Project Overview

The Energy Management System project aims to provide a robust solution for efficiently managing users and their associated smart energy metering devices. This system comprises a frontend application and two microservices, namely the UserMicroservice and DeviceMicroservice, working together to ensure seamless user experiences and device management capabilities.

* 1. System Objectives

The primary objectives of the Energy Management System include:

* Facilitating user authentication and authorization for administrators and clients.
* Enabling administrators to perform CRUD operations on user accounts, smart energy metering devices, and user-device mappings.
* Empowering clients to view their associated smart devices.
* Ensuring role-based access control to restrict unauthorized access to specific system functionalities.
  1. Microservices Architecture

The backend of the Energy Management System is built upon a microservices architecture, emphasizing modularity and scalability. The two microservices, UserMicroservice and DeviceMicroservice, handle user-related operations and device management independently, promoting system resilience and maintainability.

* 1. Frontend Application

The frontend application provides an intuitive user interface for interacting with the Energy Management System. It consists of key components such as Login.js and App.js for user authentication and the main application structure, respectively. Additionally, CRUD operations for devices and users are encapsulated within dedicated components organized in the DeviceCRUD and UserCRUD folders.

* 1. Backend Microservices

The backend microservices are designed to cater to distinct functionalities:

* + 1. UserMicroservice
* **UserController**: Manages CRUD operations for user accounts.
* **UserBuilder**: Transforms user data between DTOs (Data Transfer Objects) and entities.
* **UserService**: Handles business logic related to user management, authentication, and authorization.
* **UserDetailsDTO**: Represents detailed user information for data exchange.
  + 1. DeviceMicroservice
* **DeviceController**: Manages CRUD operations for smart energy metering devices.
* **DeviceBuilder**: Facilitates the transformation of device data between DTOs and entities.
* **DeviceService**: Implements business logic for device management and data consistency.
* **DeviceDetailsDTO**: Represents detailed device information for data exchange.
* **UserID**: Ensures data consistency by storing user IDs in both UserMicroservice and DeviceMicroservice databases.
  + 1. CommunicationMicroservice
* **CommunicationController**: Manages communication-related operations, including the processing of energy consumption data from smart metering devices.
* **CommunicationBuilder**: Facilitates the transformation of communication-related data between DTOs (Data Transfer Objects) and entities.
* **CommunicationService**: Implements business logic for communication management, including the computation of hourly energy consumption and database interactions.
* **CommunicationUserID**: Ensures data consistency by storing user IDs in the CommunicationMicroservice database.
* **CommunicationDevice**: Entity class representing communication devices, including fields like device ID, user ID, description, address, and maximum hourly energy consumption.
* **CommunicationSensor**: Entity class representing communication sensors, containing fields such as sensor ID, device ID, user ID, timestamp, and measurement.
* **RabbitMQConfig**: Configuration class for RabbitMQ, providing a message converter and connection factory settings.
* **CommunicationService**: Service class responsible for handling communication-related events, such as receiving and processing data from smart metering devices.
  1. Data Consistency

The Energy Management System ensures data consistency between microservices by utilizing the UserID class. When a user is created, their ID is inserted into both the UserMicroservice and DeviceMicroservice databases, promoting a synchronized and reliable system.

1. Conceptual Architecture
   1. Overview

The conceptual architecture of the Energy Management System is thoughtfully crafted to establish a cohesive and organized framework, enabling the smooth integration of the frontend application and the trio of microservices – UserMicroservice, DeviceMicroservice, and CommunicationMicroservice. This section delves into the essential components, interactions, and data flow intricacies within the system.

* 1. Frontend Architecture

The frontend architecture of the Energy Management System is designed with a focus on user authentication, role-based access control, and seamless interaction with the backend microservices. The components, interactions, and data flow within the frontend application play a crucial role in delivering a user-friendly and secure experience.

* + 1. Login Process

The Login.js component is the entry point for user authentication. It employs the following key features:

**User Input Handling:**

* **useState Hook:** Utilizes the useState hook to manage the state of username, password, and error messages.

**Authentication Request**

* **handleLogin Function:** Triggers the authentication process when the user submits the login form.

**Redirect Based on Role**

* Determines the user's role based on the authentication response and redirects to the appropriate page (admin or user).
  + 1. Role Based Pages

The architecture ensures that users are redirected to role-specific pages, providing a tailored experience for administrators and clients:

**Administrator Pages**

* If the login credentials match the predefined admin credentials, the user is redirected to the admin page.

**Client Pages**

* For regular users, the redirection is based on the user ID obtained from the authentication response.
  + 1. Data Interaction

The frontend components for CRUD operations, such as CRUDDevices and CRUDUsers, interact with the backend microservices through REST API calls. These components are organized in dedicated folders, providing a modular structure for managing devices and users.

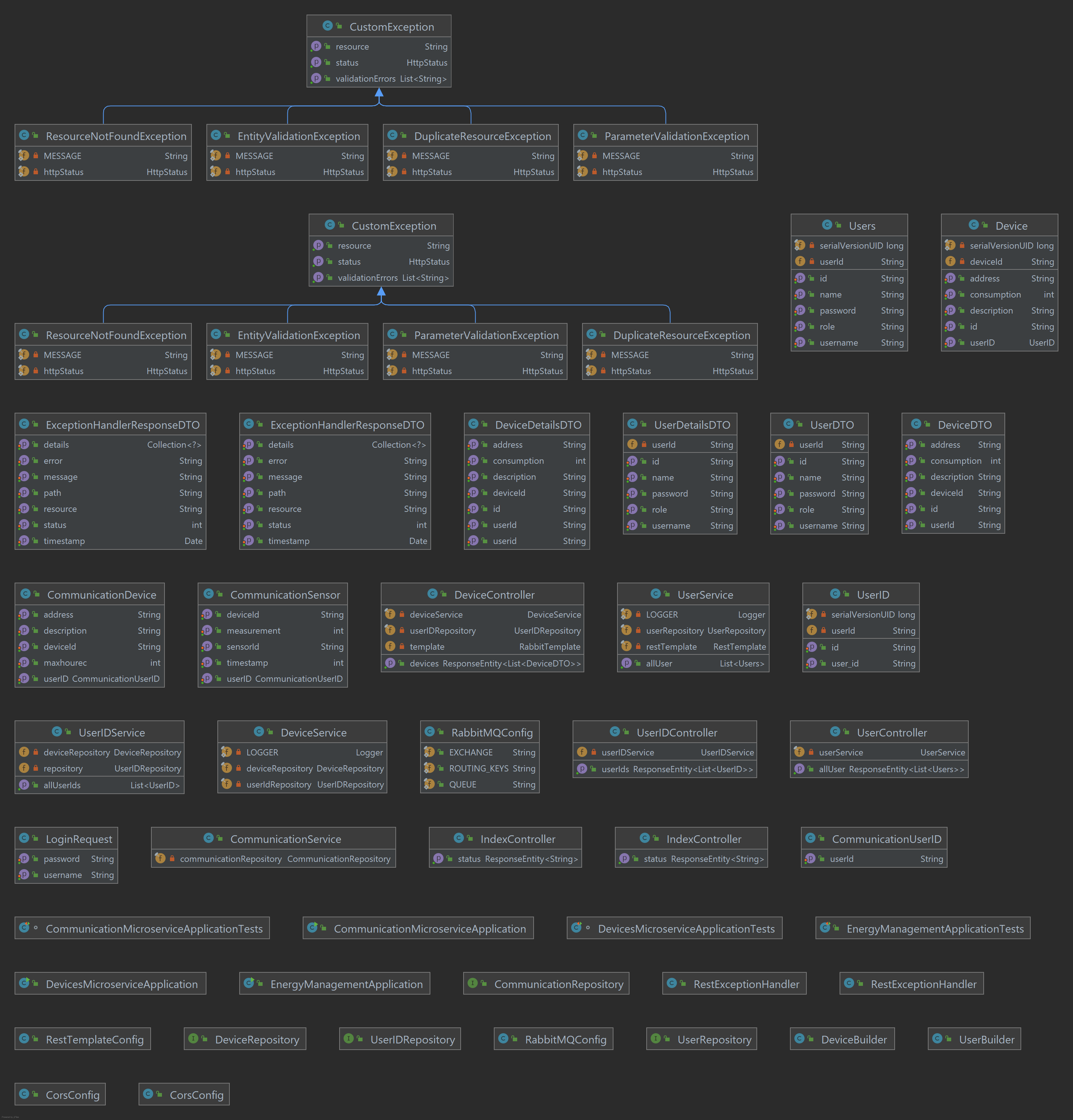
**REST API Calls**

* Utilizes the axios library to make asynchronous HTTP requests to the backend microservices.

**CRUD Operations**

* Ensures that CRUD components retrieve and manipulate data by interacting with the appropriate microservices.

1. UML Deployment Diagram



The UML Deployment Diagram visually represents the deployment architecture of the Energy Management System, illustrating how containers and images are organized and interact within the system.

* 1. Container Overview
     1. Frontend Container

The frontend-container serves as the host for the frontend application, responsible for providing the user interface of the Energy Management System. It exposes port 7000 externally, allowing users to interact with the application through their web browsers. This container is crucial for delivering a responsive and visually appealing user experience.

* + 1. UserMicroservice Container

The container-user-microservice houses the User Microservice, a backend component responsible for managing user-related operations. It communicates with the user database (container-user-database) and the frontend to handle user authentication, authorization, and other user-centric functionalities. This container exposes port 3000 externally and depends on the successful start of the device microservice and the healthiness of the user database.

* + 1. DeviceMicroservice Container

The container-device-microservice encapsulates the Device Microservice, responsible for managing smart energy metering devices. It interacts with the device database (container-device-database) and exposes port 3003 externally for external communication. This container depends on the healthiness of the device database to ensure reliable data operations.

* + 1. UserDatabase Container

The container-user-database serves as the storage facility for user-related data, catering to the needs of the User Microservice. This PostgreSQL container exposes port 5404 externally and utilizes a volume to persistently store data. The health check ensures the readiness of the database, contributing to the overall reliability of the system.

* + 1. DeviceDatabase Container

The container-device-database acts as the storage facility for data related to smart energy metering devices, supporting the operations of the Device Microservice. Similar to the user database container, it is a PostgreSQL container with port 5303 exposed externally. The health check ensures the database is healthy and ready for interactions.

* + 1. CommunicationMicroservice Container

The container-communication-microservice integrates the Communication Microservice, serving as a vital backend component for the Energy Management System. It facilitates communication with the RabbitMQ message broker, processes data from smart energy metering devices, and ensures seamless synchronization with the existing microservices. This container exposes port 8083 externally and relies on the successful start of the user and device microservices.

* + 1. CommunicationDatabase Container

The container-communication-database serves as the storage facility for data related to the Communication Microservice within the Energy Management System. This PostgreSQL container is crucial for persistently storing communication-related information, supporting the operations of the Communication Microservice. It exposes port 5505 externally and utilizes a volume to ensure data durability. The health check mechanism verifies the readiness of the database, contributing to the overall reliability of the system.

* 1. Image Overview
     1. Frontend Image

The frontend image represents the production environment for the frontend application. It is built on the Node.js environment during the build phase and later deployed using the lightweight Nginx web server in the production environment. This image ensures efficient hosting and delivery of the frontend content to users.

* + 1. User Backend Image

The user-backend image encapsulates the User Microservice, written in Java 17 using Eclipse Temurin. It is responsible for handling user-related operations, including authentication and authorization. This image ensures the consistent and reliable execution of the User Microservice.

* + 1. Device Backend Image

The device-backend image encapsulates the Device Microservice, also written in Java 17 using Eclipse Temurin. This image represents the backend component responsible for managing smart energy metering devices. It ensures the efficient execution and functionality of the Device Microservice.

* + 1. Communication Backend Image

The communication-backend image encapsulates the Communication Microservice, developed in Java 17 using Eclipse Temurin. This image represents the backend functionality responsible for processing data from smart energy metering devices, utilizing the RabbitMQ message broker for event-driven communication.

* 1. Deployment Connections

**Frontend to User Microservice**: The frontend communicates with the User Microservice through HTTP requests. This interaction enables user authentication, role-based redirects, and other user-related functionalities.

**Frontend to Device Microservice**: HTTP requests facilitate communication between the frontend and the Device Microservice. This connection enables the frontend to manage smart energy metering devices and retrieve relevant data.

**User Microservice to User Database**: The User Microservice interacts with the User Database to perform CRUD operations on user-related data. This connection ensures the persistence and retrieval of user information.

**Device Microservice to Device Database:** The Device Microservice communicates with the Device Database to manage data related to smart energy metering devices. This connection ensures the reliability and consistency of device-related information.

**CommunicationMicroservice to CommunicationDatabase**: The Communication Microservice communicates with the Communication Database to store and retrieve communication-related data. This connection is essential for managing information processed by the Communication Microservice, contributing to the overall reliability and functionality of the Energy Management System.

**Frontend to Communication Microservice**: HTTP requests establish communication between the frontend and the Communication Microservice. This connection enables the frontend to interact with the Communication Microservice, facilitating features such as real-time notifications and communication-related functionalities.

**Communication Microservice to Device Microservice**: The Communication Microservice communicates with the Device Microservice through message broker middleware. This interaction is crucial for processing data from smart energy metering devices, computing hourly energy consumption, and storing relevant information.

**Communication Microservice to User Microservice**: The Communication Microservice interacts with the User Microservice to access user-related functionalities, authentication, and authorization services. This connection ensures seamless communication between the Communication Microservice and the User Microservice.

1. Functional and Non-Functional Requirements
   1. Functional Requirements
      1. User Microservice
2. **User Registration:**

* Users can register by providing necessary details.
* Registration includes validation of username, password, and role.
* Upon successful registration, a unique user ID is generated and stored.

1. **User Authentication:**

* Users can authenticate using their credentials (username and password).
* Differentiates between regular users and admin users during authentication.
* Successful authentication results in the issuance of a user ID.

1. **User Retrieval:**

* Users can retrieve their information based on their unique user ID.
* Admin users can retrieve information for any user based on their ID.

1. **User Update:**

* Users can update their profile information, including name, password, and role.

1. **User Deletion:**

* Users can request the deletion of their account, removing associated data.
* Admin users can delete any user account based on the user ID.

1. **User Listing:**

* Admin users can retrieve a list of all registered users.
  + 1. Device Microservice

1. **Device Registration:**

* Users can register smart energy metering devices, associating them with a specific user ID.
* Device registration includes details such as address, consumption, and description.

1. **Device Retrieval:**

* Users can retrieve information about their registered devices based on device ID.
* Admin users can retrieve information for any device based on the device ID.

1. **Device Update:**

* Users can update information about their registered devices.

1. **Device Deletion:**

* Users can request the deletion of a registered device, removing associated data.
* Admin users can delete any device based on the device ID.

1. **Device Listing:**

* Users can retrieve a list of all registered devices associated with their user ID.
* Admin users can retrieve a list of all devices in the system.

1. **Device Retrieval by User:**

* Users can retrieve a list of devices associated with a specific user ID.
  1. Non-functional Requirements
     1. Performance

**Response Time:**

* The system should provide responses within a reasonable time frame (e.g., under 1 second for most operations).

**Scalability:**

* The architecture should support scaling to handle an increasing number of users and devices.
  + 1. Reliability

**Availability:**

* The system should aim for high availability, minimizing downtime for users.

**Data Consistency:**

* Data consistency should be maintained across microservices to avoid discrepancies.
  + 1. Usability

**User-Friendly Interface:**

* The frontend should provide a user-friendly interface for easy navigation and interaction.

**Error Handling:**

* The system should provide clear error messages to users in case of invalid inputs or system errors.
  + 1. Interoperability

**Microservices Interaction:**

* The communication between the User and Device microservices should be seamless and reliable.

**API Documentation:**

* APIs should be well-documented to facilitate integration with other systems.

1. Conclusion

In the dynamic landscape of energy management, the presented Energy Management System (EMS) stands as a comprehensive and meticulously designed solution to address the intricate needs of both end-users and administrators. This documentation has extensively covered various aspects, from the conceptual architecture and deployment details to functional and non-functional requirements. Let's recap the key points and conclude the documentation.

* 1. Achievements

The EMS achieves a fine balance between flexibility, functionality, and robustness. The microservices architecture, comprising the User Microservice and Device Microservice, fosters modularity, scalability, and maintainability. Leveraging containerization with Docker and orchestrating with Docker Compose, the deployment process is streamlined, promoting ease of use and efficient resource management.

* 1. Microservices Colaboration

The collaboration between the User Microservice and Device Microservice is vital for ensuring data consistency and a seamless user experience. The UserID mechanism acts as a linchpin, establishing a link between user-related data in the two microservices. This sophisticated interplay is orchestrated with precision, underpinning the system's reliability and integrity.

* 1. Frontend User Experience

On the frontend, the Login module and various CRUD functionalities for both devices and users deliver an intuitive and responsive user interface. The Login.js component ensures secure user authentication, providing a gateway to personalized experiences based on user roles. Each CRUD operation, meticulously implemented, contributes to the overall usability and efficiency of the system.

* 1. Deployment and Scalability

The deployment, facilitated by Docker containers and orchestrated by Docker Compose, showcases an infrastructure ready for scaling. Each container, from the frontend to microservices and databases, plays a distinct role in crafting a resilient and scalable architecture. The use of Nginx as a reverse proxy for the frontend enhances security and performance.

* 1. Functional and Non-Functional Requirements

The detailed exploration of functional requirements encompassed user registration, authentication, device management, and more. These specifications ensure that the EMS caters to the diverse needs of users and administrators. Non-functional requirements, spanning performance, reliability, security, usability, and interoperability, set the standard for a system that is not just functional but also dependable, secure, user-friendly, and adaptable.

* 1. Future Enhancements

While the EMS is robust in its current form, there's always room for evolution. Future enhancements might include the integration of advanced analytics for energy consumption patterns, real-time monitoring capabilities, and the incorporation of emerging technologies like machine learning for predictive maintenance.

* 1. Closing Thoughts

In conclusion, the Energy Management System is a testament to meticulous planning, technical expertise, and a user-centric approach. It stands poised to empower users in managing their energy consumption effectively while providing administrators with a comprehensive toolset. As technology evolves, the EMS is well-equipped to adapt, ensuring its relevance and effectiveness in the ever-changing landscape of energy management. This documentation serves as a guide, a blueprint, and an ode to the continuous pursuit of excellence in energy management systems.

1. Bibliography

The Energy Management System (EMS) documentation draws upon a variety of sources to provide a comprehensive and well-informed overview. The following bibliography lists key references and sources that have influenced the development and documentation of the EMS project.

* 1. Research Papers and Articles

1. **Microservices Architecture: Design and Challenges**

Author: Martin Fowler

Link: microservices.io

1. **Docker Documentation**

Author: Docker Inc.

Link: Docker Documentation

1. **RESTful API Design Best Practices**

Author: Vinay Sahni

Link: [API University](https://www.api university.com/blog/api-design/restful-api-design-best-practices)

* 1. Framework and Technology Documentation

1. **Spring Framework Documentation**

Author: The Spring Team

Link: Spring Framework Documentation

1. **React Documentation**

Author:React.dev

Link: React Documentation

1. **PostgreSQL Documentation**

Author: The PostgreSQL Global Development Group

Link: PostgreSQL Documentation

* 1. Tutorials and Online Courses

1. **Docker and Kubernetes: The Complete Guide**

Author: Stephen Grider

Platform: Udemy

Link: Udemy Cours

1. **Spring Boot - Introduction to Microservices**

Author: Java Brains

Platform: YouTube

Link: Java Brains - Microservices with Spring Boot

1. **React - The Complete Guide (incl Hooks, React Router, Redux)**

Author: Maximilian Schwarzmüller

Platform: Udemy

Link: Udemy Course

* 1. Online Forums and Communities

1. **Stack Overflow**

Platform: Stack Overflow

1. **Reddit - r/programming**

Platform: Reddit - r/programming