**FACULTY OF AUTOMATION AND COMPUTER SCIENCE COMPUTER SCIENCE DEPARTMENT**

**DISTRIBUTED SYSTEMS**

**Energy Management System**

Student: Vasilache Andre-Florin

Grupa 30241, Anul IV Calculatoare Română

**Table of contents:**

1. Introduction
2. System Architecture
3. Functional Requirements
4. Implementation

**Introduction**

Welcome to the exploration of a Monitoring and Communication Microservice tailored for the Energy Management System. This microservice operates on a robust foundation of a message broker middleware, orchestrating the collection of data from smart metering devices. The gathered data undergoes meticulous processing to compute hourly energy consumption, subsequently finding its place within the database of the Monitoring and Communication Microservice.

Facilitating seamless synchronization between the databases of the Device Management Microservice and the newly introduced Monitoring and Communication Microservice is an event-driven system. This system leverages a dedicated topic for device changes, delivering device information through a queue specifically designed for the Monitoring and Communication Microservice.

Embracing the JSON format, the measurements journey to the queue, presenting details such as timestamp, device\_id, and measurement\_value. Meanwhile, the Monitoring and Communication Microservice, equipped with a Message Consumer component, assumes the responsibility of processing these measurements. The objective is clear: to compute the total hourly energy consumption and securely store it in the database. Adding a layer of intelligence, if the computed total hourly energy consumption surpasses the predefined maximum value for the device, an asynchronous notification is dispatched to the user's web interface, ensuring timely and informed responses. Get ready to dive into the intricacies of this dynamic and interconnected microservices architecture.

**Chat Microservice**

The Chat Microservice enables real-time, bidirectional communication between users and administrators. It's designed to operate asynchronously, allowing for efficient handling of multiple chat sessions.

**Features:**

* User Interface: Incorporates a user-friendly chat box in the front-end, allowing users to send and receive messages.
* Asynchronous Communication: Messages from users are sent asynchronously to administrators, who receive these messages along with the user's identifier.
* Bidirectional Chat Sessions: Enables ongoing message exchange between users and administrators during active chat sessions.
* Multiple Chats: Administrators can engage in conversations with multiple users simultaneously.
* Read Notifications: Both users and administrators receive notifications upon their messages being read by the other party.
* Typing Indicators: Displays a notification when the counterpart is typing a message.

**Authorization Component**

**Purpose**

The Authorization Component is a crucial part of the system, responsible for securing access to the chat microservice and other related microservices.

**Functionality**

* Service Designation: A specific service, such as a User Microservice or a dedicated authorization service, is designated as the authorization server.
* Token-Based Access: Generates access tokens for client applications, which are essential for accessing and interacting with the various microservices in the network.

**Implementation Technologies**

**Chat Component**

WebSockets: Utilized for real-time, bidirectional communication between clients (users) and servers (administrators).

**Authorization Component**

JWT (JSON Web Tokens): Employed for secure user authentication and authorization across all connected microservices.

**Architectural Options:**

* Shared Secret Key Model: Uses a common secret key among microservices for token validation, generated by the authorization service.
* Reverse Proxy Configuration: The authorization service doubles as a reverse proxy, thereby centralizing access control and reducing the authorization load on individual services.
* Spring OAuth2 Integration: Implements an OAuth 2 authorization service model to manage access to the microservices.

**System Architecure**

Developing an online platform is essential for efficiently managing users, their associated smart energy metering devices, and the data collected from each device. Upon completing a login process, two distinct user types gain access to the system: administrators (managers) and clients. The administrator holds the authority to execute CRUD (Create-Read-Update-Delete) operations on user accounts (specified by ID, name, and role: admin/client), registered smart energy metering devices (specified by ID, description, address, and maximum hourly energy consumption), and the association of users with devices. It's noteworthy that each user can possess one or more smart devices situated in various locations. Once the mapping is finalized, the system stores the energy consumption for each device on an hourly basis as tuples, denoted as <timestamp, energy consumption>, within the database.

The formulation of an online platform is crucial for overseeing users, their respective smart energy metering devices, and the data obtained from each device. Post the login process, two user categories emerge: administrators (managers) and clients. The administrator is endowed with the capability to execute CRUD (Create-Read-Update-Delete) operations on user accounts (characterized by ID, name, and role: admin/client), registered smart energy metering devices (characterized by ID, description, address, and maximum hourly energy consumption), and the linkage of users to devices. It's imperative to note that a user can possess one or more smart devices located in different places. Subsequent to the completion of the mapping, the database stores the energy consumption for each device on an hourly basis, represented as tuples in the form <timestamp, energy consumption>.

A diagram of a computer

Description automatically generated

**Arhitectura Conceptuala a Sistemului Distribuit**

**Functional Requirements**

Our application will materialize as a Java-based app, employing the Spring Boot framework alongside Maven for streamlined development. With the primary objective of database administration in mind, Java offers an efficient avenue for constructing such an application. The chosen Integrated Development Environment (IDE) for project development is IntelliJ IDEA. Additionally, the database aspect will be addressed using DataGrip in conjunction with MySQL. This database choice aligns seamlessly with the backend requirements of our application, ensuring a cohesive and efficient operation.

In this iteration, a message producer has been introduced to feed messages sourced from an external reservoir into the RabbitMQ platform. These messages are subsequently relayed to the consumer within the Spring application.

Java stands out as a versatile, robust, secure, and object-oriented programming language. Recognized for its high-level nature, Java's syntax mirrors English-like language constructs. Originally developed by Sun Microsystems in 1995, it is presently maintained and distributed by Oracle. Java boasts its runtime environment and API, earning it the moniker of a platform.

The Spring Framework, a Java platform, offers extensive infrastructure support for the development of Java applications. By handling the underlying infrastructure complexities, Spring allows developers to concentrate on their application logic. Renowned for its emphasis on speed, simplicity, and productivity, Spring has solidified its position as the world's most widely adopted Java framework.

RabbitMQ, known for its lightweight nature, proves easily deployable both on-premises and in cloud environments. Supporting multiple messaging protocols, RabbitMQ excels in distributed and federated configurations, addressing high-scale and high-availability requirements effectively.

**Authorization Server:**

Service Selection: Choose an appropriate service to act as the authorization server, which can be the User Microservice or a dedicated Authorization Microservice.

Token Generation: The selected service generates JWT (JSON Web Tokens) as access tokens for client applications.

Microservice Access: These tokens will serve as credentials for accessing other microservices within the system.

**Chat Component**

The Chat Component leverages Web Sockets technology to provide real-time communication features.

A diagram of a software company

Description automatically generated

**UML Deployment Diagram**