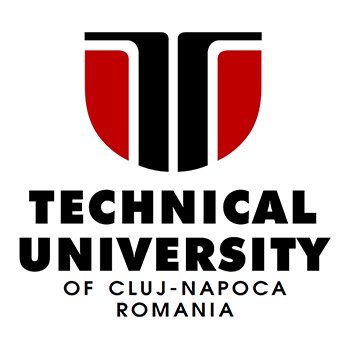
Energy Management System





Vulsan Bianca Maria

Technical University of Cluj – Napoca

Distributed Systems – 2023/24

Coordinator: Pop Cristina

|  |  |  |
| --- | --- | --- |
| **1.** | Conceptual architecture………………………….………………………… | 2 |
| **2.** | Database design………………………………….………………………. | 3 |
| **3.** | Deployment……………………………………………………………… | 4 |
| **4.** | [Build](#C4) and Execution Considerations………………………………………….. | 4 |
| **5.** | Functionality…………….……………………………………………….. | 5 |
| **6.** | Conclusions………………………..…………………………………….. | 5 |

1. **Conceptual architecture**

The Energy Management System is designed to provide efficient management of user accounts and smart energy metering devices. Its core functionality includes:

**User Authentication:** Users log in to the system, and based on their roles (administrator or user), they are redirected to their respective views. This ensures that only authorized users can access the system.

**User Management:** Administrators have the ability to create, read, update, and delete user accounts. A user account includes attributes such as ID, name, and role (admin/client).

**Device Management:** Administrators also manage smart energy metering devices. This involves CRUD operations on devices, including attributes like ID, description, address, and maximum hourly energy consumption.

**User-Device Mapping:** The system allows administrators to create mappings between users and devices. Each user can own one or more smart devices located at different addresses.

**Client Access:** Clients, after logging in, have access to view all the smart devices associated with their account.

**Device Consumption Overview:** Clients, can visualize on a diagram the total consumption/hour of a device in the selected day.

**Notifications:** Clients receive notification if a threshold is passed by one of their devices.

**Microservices Architecture**

The Energy Management System is built as a microservices architecture using Spring Boot for the backend (Java) and Angular for the frontend (JavaScript-based framework). Two primary microservices are employed:

* User Management Microservice:
  + Responsible for user-related functionality, including authentication and authorization.
  + Manages user accounts, including creation, modification, and deletion.
  + Provides secure access to user-specific data and features.
* Device Management Microservice:
  + Focuses on device-related operations.
  + Handles CRUD operations on smart energy metering devices.
  + Manages user-device mappings, allowing administrators to link users to their devices.
  + These microservices communicate with each other to ensure data consistency and are independently deployable, scalable, and maintainable components of the system.
* Monitoring Microservice:
  + Monitor the measurements received from multiple sensors
  + Process the data at a certain time interval
  + Send notification to the owner of the device if a certain threshold is passed

**Security**

Security measures are in place to restrict access to administrator pages, ensuring that only authorized personnel can perform CRUD operations on users and devices. This is achieved using Spring Security and Angular Authentication Guards.

**Authentication:** Users are required to log in to access the system. Authentication is managed using a combination of user credentials and cookies.

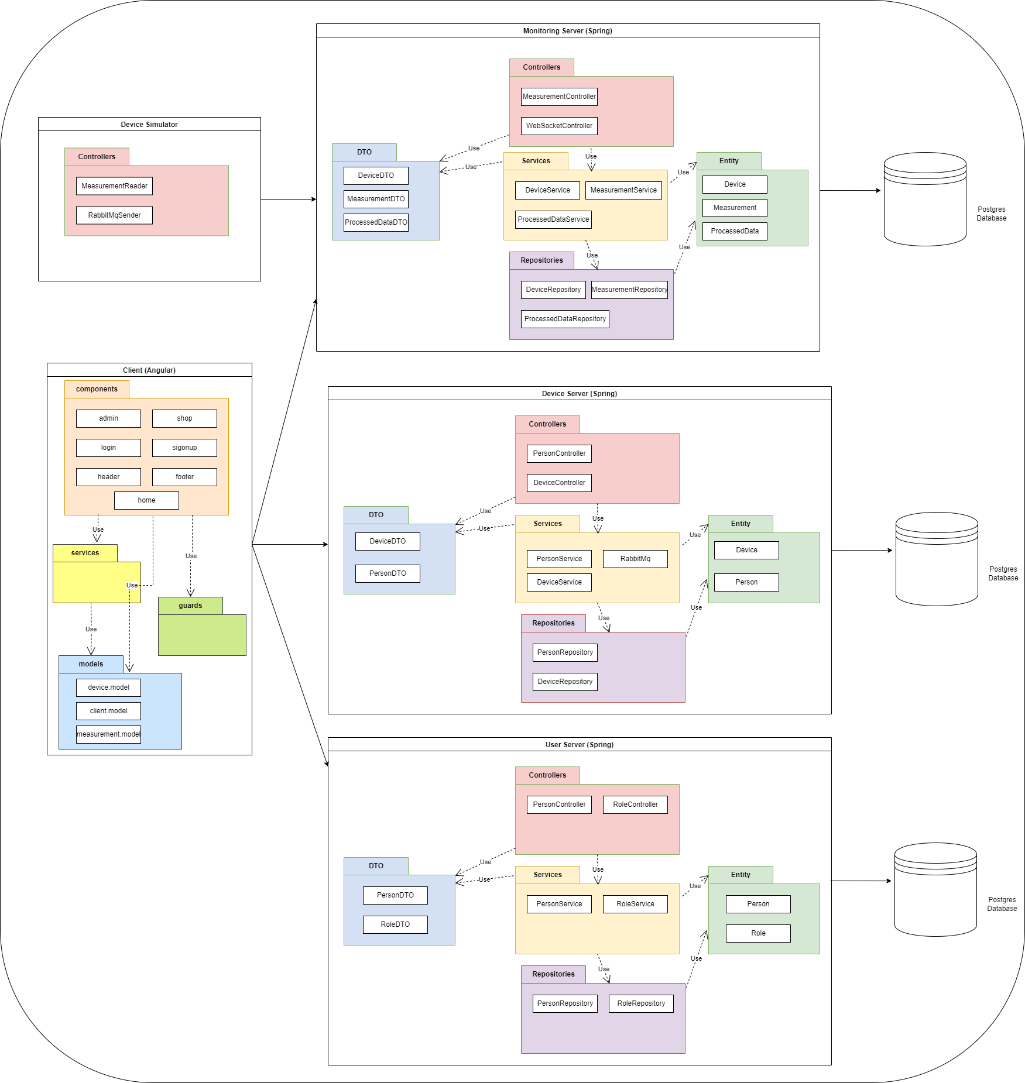
**Authorization:** Role-based access control is enforced through Angular Authentication Guards, which restrict access to administrator-specific pages. Administrators have roles that grant them access, while clients are restricted to view-only capabilities.

**RabbitMQ**

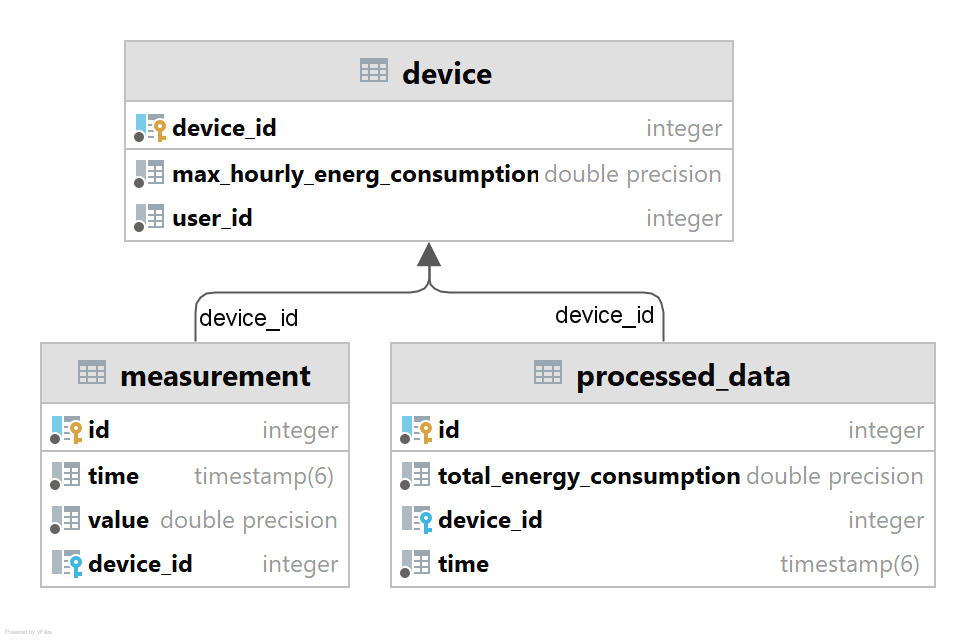
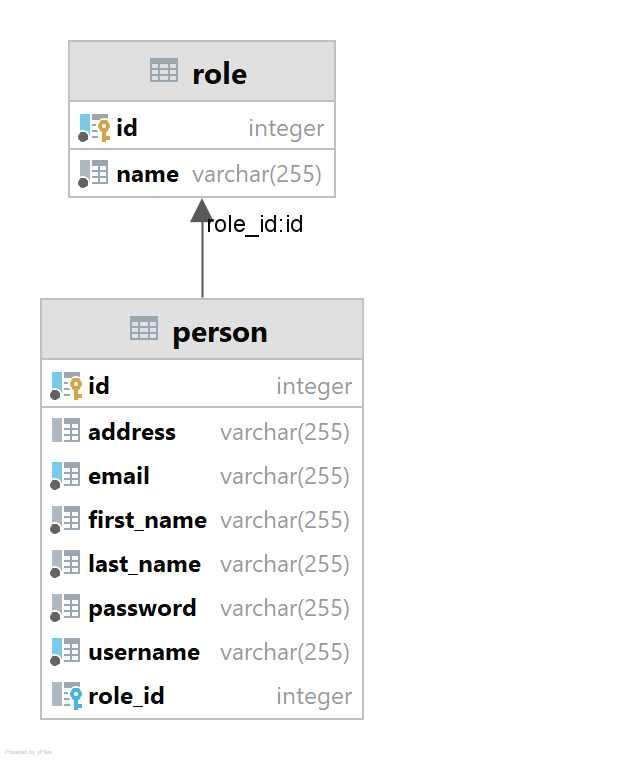
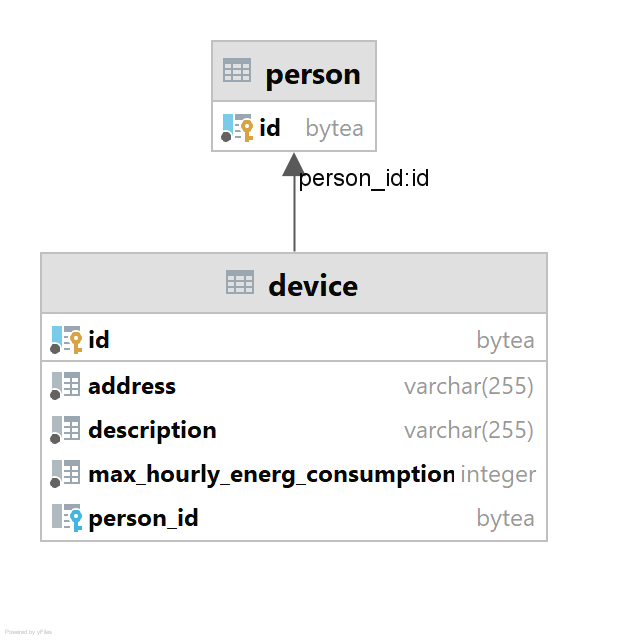
RabbitMQ is an open-source message broker that facilitates communication between different parts of a distributed system. It implements the Advanced Message Queuing Protocol (AMQP) and supports other messaging protocols like MQTT and STOMP. Key concepts include message queues, exchanges, bindings, and the publish/subscribe pattern. RabbitMQ provides features such as durability, persistence, fault tolerance through clustering, and a web-based management interface. It is widely used in distributed systems and microservices architectures for reliable and scalable message queuing.

**WebSockets**

WebSockets is a communication protocol allowing bidirectional, full-duplex communication over a single, long-lived connection. It reduces latency compared to traditional HTTP, maintains a persistent connection, and supports both binary and text data. The WebSocket handshake initiates the connection, and security measures ensure encrypted data transmission. Commonly used in real-time applications like chat and gaming, WebSockets have a standardized protocol (RFC 6455) and associated APIs and libraries for simplified implementation in various programming languages. They are crucial for scenarios requiring low-latency, real-time communication between clients and servers.Top of Form



1. **Database design**



1. **Deployment**

Deploying the Energy Management System using Docker allows for a consistent and isolated environment. Docker ensures that the application and its dependencies are packaged and run consistently across different environments.

**Prerequisites**

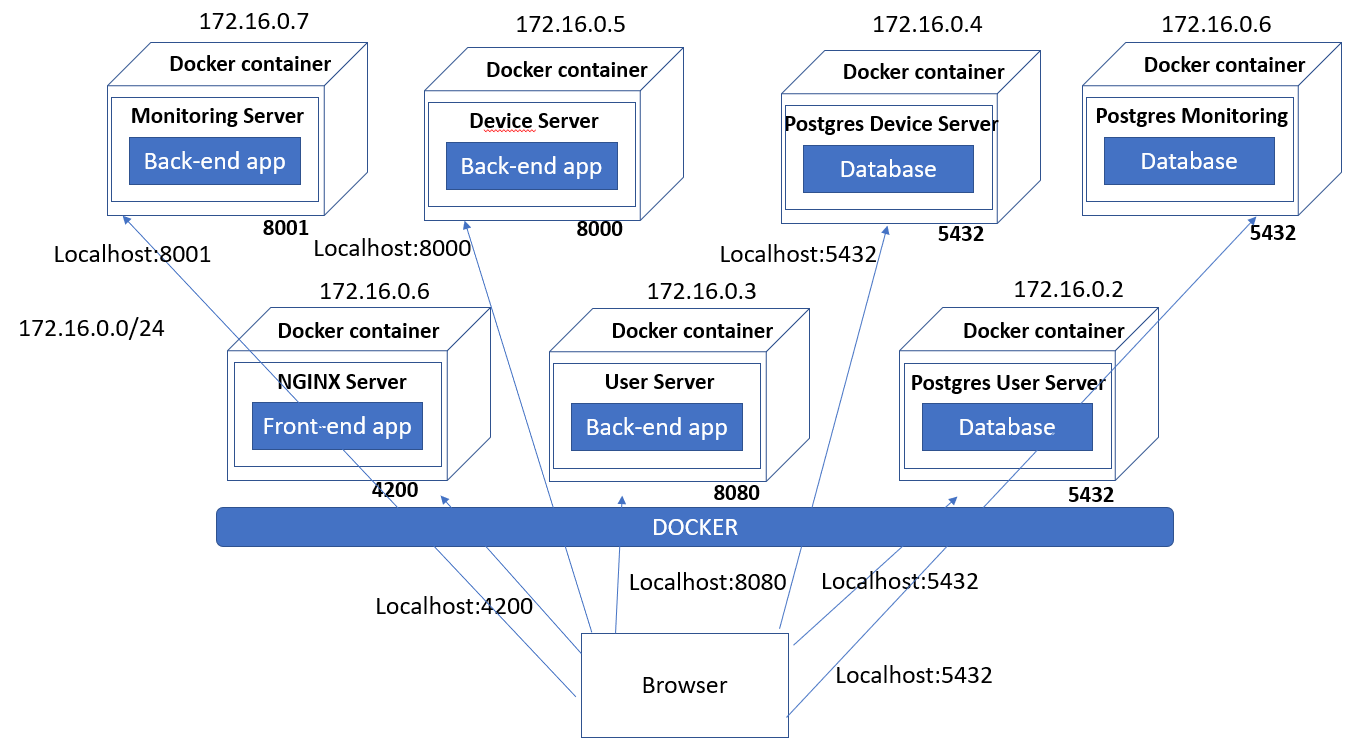
Before starting the deploy with Docker, we need to have the following prerequisites in place:

* + Docker: Install Docker on your target deployment machine.
  + Docker Compose (Optional): Docker Compose simplifies multi-container Docker application deployment.

**Building Docker Images**

Before deploying the services, we need to create Docker images for your Spring Boot applications (User Management and Device Management), the Angular frontend and the databases associated to each microservice. Docker images are built using Dockerfiles.

After the images are build using **docker build -t image\_name .** command, we can move on to running the docker compose file. This file specifies details about the port of each application, the ip, the corresponding image, the dependencies on other applications and so on. To automize the process, in this project there is one single docker-compose.yml file, but there can be more if we want to use different containers for our application. The docker-compose file is executed using the command **docker compose up -d**.



1. **Build and Execution considerations**

Before building and running the Energy Management System, ensure the following prerequisites are installed on the development environment:

**Backend Prerequisites:**

* Java Development Kit (JDK) 8 or higher: Required for running Spring Boot applications.
* Maven: A build tool for managing project dependencies. Make sure Maven is properly installed and added to the system's PATH.
* Database: PostgreSQL or any other relational database supported by Hibernate. Ensure it's installed and configured correctly.

**Frontend Prerequisites:**

* Node.js and npm: Required for running and managing Angular applications. The version used in this project is 16.17
* Angular CLI: Install the Angular CLI globally using npm.

**Backend (Spring Boot with Hibernate):**

* Open a terminal and navigate to the root directory of your backend project.
* Build the project using Maven. Run the following command: mvn clean install This command compiles the code, resolves dependencies, and creates an executable JAR file in the "target" directory.
* The built JAR file can be found in the "target" directory.

**Frontend (Angular):**

* Open a terminal and navigate to the root directory of your Angular project (where the package.json file is located).
* Install project dependencies using npm: npm install This command will download and install the required packages defined in the "package.json" file.
* Once the dependencies are installed, you can start the Angular development server using the Angular CLI: ng serve By default, the server will run on http://localhost:4200/

1. **Functionality**

The application works with two types of users: administrator and client. Both user types should login to access any data from the server and the database respectively. New accounts can be created for both user types from the client application.

The administrator users can perform CRUD operations on the Device and Person entities. They can also see the list of all devices, the list of all clients and map devices to clients. The client users can see their account details and their devices.

The client can view a chart with the hourly consumption of the selected device on the selected day. He also receives real time notifications when a device exceeds the maximum hourly consumption.

For the database actualization and communication, an ORM was used. Hibernate provides an easy-to-configure and use environment for all the basic CRUD operations, but also for more complex ones (such as JOINS).

1. **Conclusions**

The application was definitely challenging to bring up and running. However, the flexibility allowed by the freedom of choice for the programming languages, frameworks and tools made the experience a lot better. Because of this, however, certain problems were met during the deployment phase, but we managed to get over them. Finally, new concepts were learnt from this project, such as password encryption, session-validation using JWT, RabbitMQ, WebSockets or what the actual deployment phase of an application means.