Energy Management System

Student: Budiul Cristian-Carol

Contents

1. Introduction 3
2. Conceptual Architecture 3
Overview 3
Communication flow 3
3. UML Deployment Diagram 4
4. Readme file 5
Build and Execution Considerations 5
Prerequisites 5
Building and Configuring Microservices 5
Building and Configuring the Frontend 5
Database Configuration 5
Running the System 5
User Credentials 5

1. Introduction

The Energy Management System is a solution for managing the multitude of devices on the power grid. This solution was developed using the React framework for the frontend, Spring Boot 3, Spring Web, Spring Security and RabbitMQ for the backend, and PostgreSQL as the database management system. This document provides a comprehensive overview of the system's architecture, including a conceptual architecture, a UML Deployment diagram, and a readme file with considerations for building and running the system.

2. Conceptual Architecture

Overview

The energy management system consists of:

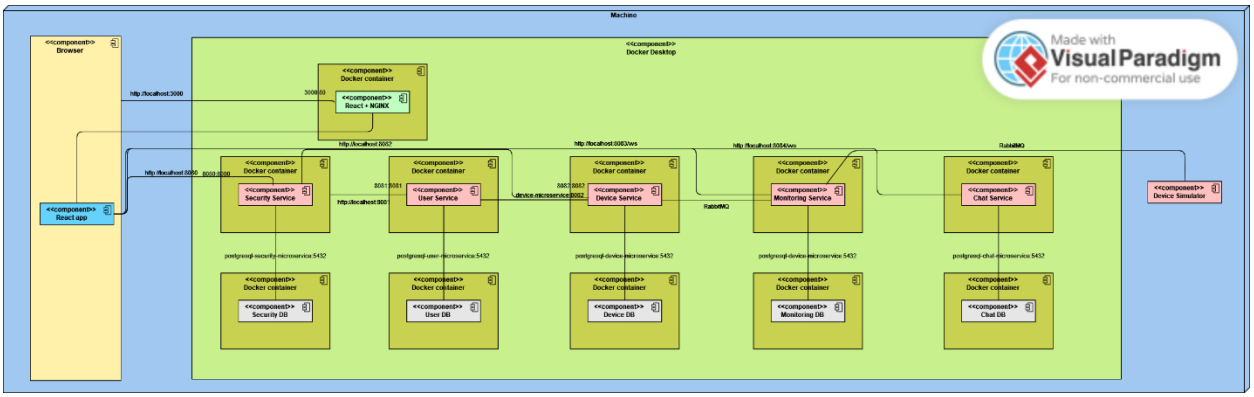
- Frontend Application: The user interface, implemented using React, comprises a login page, an admin page for user and device CRUD operations, and a user page for viewing user-associated devices. The admin and user also have chat pages to communicate with other users or with themselves.
- Backend Application: Split into microservices, as follows:
    - User Management Microservice: The User Management Microservice, developed using Java Spring Boot, is responsible for the management of user entities. These user entities are characterized by unique UUIDs, usernames, passwords, names, and roles, which can be either "Admin" or "User." This microservice provides endpoints that enable various CRUD operations on user entities.
    - Device Management Microservice: The Device Management Microservice, also implemented using Java Spring Boot, oversees two core entities: devices and users. Devices are described by their UUIDs, User UUIDs, descriptions, addresses, and maximum energy consumption (Max Wh). In addition to this, the microservice manages user entities that serve as proxies, ensuring that only existing users can be associated with devices, even if the User Management Microservice experiences downtime. This microservice also sends commands to the Device Monitoring Microservice every time a device is created, updated, or deleted. It also sends a “initialization” command to the Device Monitoring Microservice when it starts – the command contains a “create” command containing all the devices that are currently in the database.
    - Device Monitoring Microservice: The Device Monitoring Microservice, also implemented using Java Spring Boot, uses RabbitMQ to keep track of what devices need to be monitored by listening to the commands issued by the Device Management Microservice. For every device that exists, it opens a RabbitMQ queue using the device’s UUID to listen to device measurements. When a device measurement is received, the hourly data for that device is updated. Finally, this microservice uses Websockets to issue notifications to the user and to send the graph data when requested.
    - Chat Microservice: The Chat Microservice, also implemented using Java Spring Boot, uses WebSockets and a database to transmit and store messages sent between users. This microservice also includes support for seen and typing notifications.
    - Security Microservice: The Security Microservice, also implemented using Java Spring Boot, secures all REST endpoints using JWT tokens. The only unsecured endpoint is the login endpoint. It also acts as a proxy between the React application and the other microservices that have REST endpoints (the User microservice and the Device microservice).
- PostgreSQL Database: Used to store data related to users and devices, including device measurements, messages, and logins.

Communication flow

Users log in through the React frontend application and access the appropriate pages based on their roles. The React frontend communicates with the Security, User Management and Device Management using REST APIs, and Device Monitoring and Chat microservices through Websockets. Spring Security is configured for secure authentication and authorization, ensuring restricted access to administrator-specific features.

3. UML Deployment Diagram

Below is a simplified UML Deployment diagram illustrating the distribution of components within the Energy Management System:



4. Readme file

Build and Execution Considerations

To build and run the Energy Management System, follow these steps:

Prerequisites

- Java Development Kit (JDK) 17 or higher
- Docker desktop

Building and Configuring Microservices

1. Clone the backend repository.
2. Build and deploy each microservice using the provided docker files / docker compose.

Building and Configuring the Frontend

1. Clone the React frontend application repository.
2. Build and deploy the frontend using the provided docker files.

Database Configuration

There is no need to do additional configuration, as the default configuration is included in the microservice compose.yaml file. However, if another configuration is needed, some environment variables are available.

Running the System

1. Deploy the microservices, the database and the React frontend application.
2. Access the system in your web browser, and perform the following actions:

- Administrators: Log in with your admin credentials to perform CRUD operations on users, devices, and user-device mapping.
- Clients: Log in with your client credentials to view your associated smart energy metering devices.

User Credentials

- Administrator:
  - Username: carol8
  - Password: secret
- Client:
  - Username: maria
  - Password: user