Energy Management System

Contents

Chapter 1

Assigment 1

1.1 Overview

This project is an Energy Management System (EMS) designed to manage users and their associated energy devices. It comprises a frontend for user interaction and two Java Spring REST microservices: User Management Service: Handles CRUD operations for users, Device Management Service: Manages device data and user-device mappings.

The system supports two roles: admin: full access to CRUD operations on users, devices, and user-device mapping, client: restricted access, limited to viewing their assigned devices.

1.2 Conceptual Architecture

This Energy Management System (EMS) uses a microservices architecture with a frontend interface served through NGINX. The key components include:

Frontend: is built using a JavaScript framework and is served through an NGINX container. NGINX manages the delivery of static files to the client and provides reverse proxy functionality to route API requests from the frontend to the backend microservices. Role-based access control is enforced on the frontend to ensure that users can only access features within to their role. Microservices:

Microservices: user management mervice uses a Spring REST microservice responsible for handling CRUD operations for users and managing login and session data. device management service is a Spring REST microservice that manages device-related data and handles associations between users and devices. Both microservices communicate via REST APIs and are containerized for independent scaling and management.

Database: A relational database container (e.g., PostgreSQL or MySQL) stores user information, device details, and user-device mappings. Both the user and device management services connect to this database to perform CRUD operations.

1.3 UML Deployment Diagram

1.3.1 How everything works

The application operates through a series of interactions, beginning with a user sending an HTTP request to an application endpoint. This initial request is directed to the NGINX server, which, based on predefined configurations routes the request to the appropriate backend service. Each controller processes the request, applying necessary validation checks and invoking the relevant service methods. The service layer then handles the core business logic, which

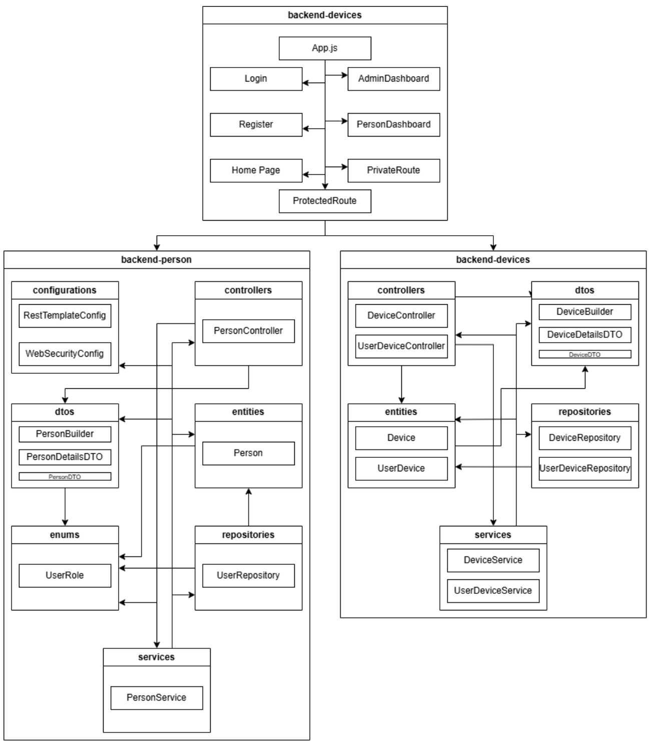


Figure 1.1: Conceptual Architecture

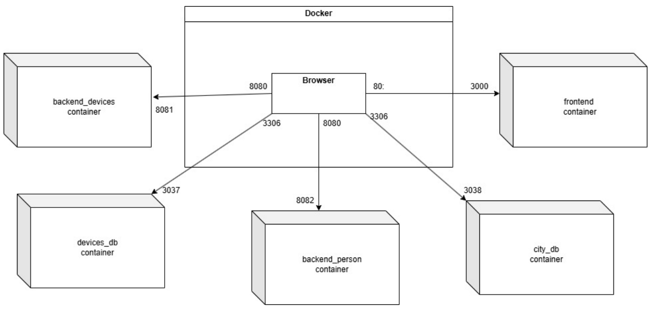


Figure 1.2: UML Deployment Diagram

could include creating, retrieving, updating, or deleting records within the database. Services interact with a database container, ensuring data persistence across container restarts. Once the requested operation is completed, the controller sends a response back to NGINX, which, in turn, forwards it to the client, thereby completing the request-response cycle. This flow ensures that requests are efficiently handled and that the system remains modular, with NGINX managing routing and access to each service.

1.4 Authentication and Security

In this project, security is managed entirely on the frontend, where user access is restricted based on roles. The frontend application verifies user credentials and assigns a role, storing session data in session storage to maintain the authenticated state. Admins are granted full access to CRUD operations on users and devices, while clients have view-only access to their assigned devices. Role-based routing directs users to the appropriate dashboard upon login, with frontend logic blocking unauthorized page access to prevent users from manually navigating to restricted areas.

Chapter 2

Assignment 2

2.1 Monitoring and Communication

The Monitoring and Communication Microservice processes energy data from smart metering devices through a message broker. It performs the following functions:

- Message Broker Integration: Receives data from device simulators via a message broker, enabling asynchronous communication.
- Energy Consumption Processing: Consumes energy data in JSON format, computes hourly energy consumption, and stores it in the database.
- Device Management Sync: Syncs with the Device Management Microservice to retrieve device limits and validate if consumption exceeds the maximum threshold.
- Exceeding Consumption Notification: If consumption exceeds limits, the system notifies the user.
- Data Storage: Stores processed hourly energy data in the database for later reporting and analysis.

2.2 Producer

The Smart Metering Device Simulator generates and sends simulated energy data to the message broker. Its key features are:

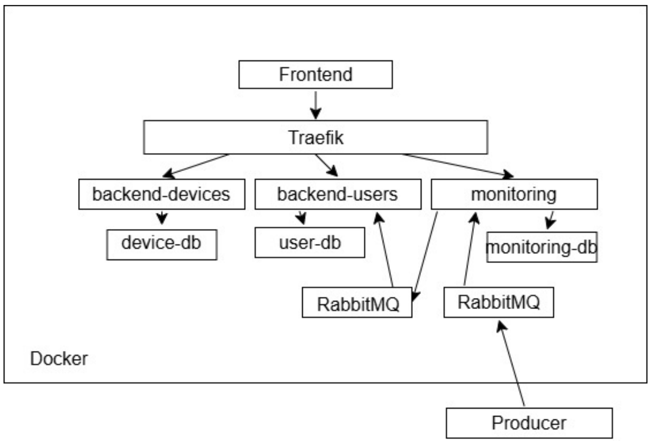


Figure 2.1: Deployment Diagram

- Simulated Device Data: Reads energy values from sensor.csv and sends data in a JSON format, including timestamp, device ID, and energy measurement.
- Standalone Application: The simulator is a desktop application that uses a configuration file to set the device ID, linking it to a user's device in the Device

Chapter 3

Assignment 3

3.1 Overview

The third assignment introduces a real-time chat system designed to streamline communication between admins and users in a secure and interactive environment. Built using WebSocket technology, the system enables instant messaging with live update. Admins have the added ability to broadcast messages to multiple users at once, ensuring efficient communication. All interactions are safeguarded with JWT (JSON Web Token) authentication, ensuring secure access and role-based permissions. The frontend is thoughtfully designed for an intuitive and responsive experience, making the chat system user-friendly for both admins and users.

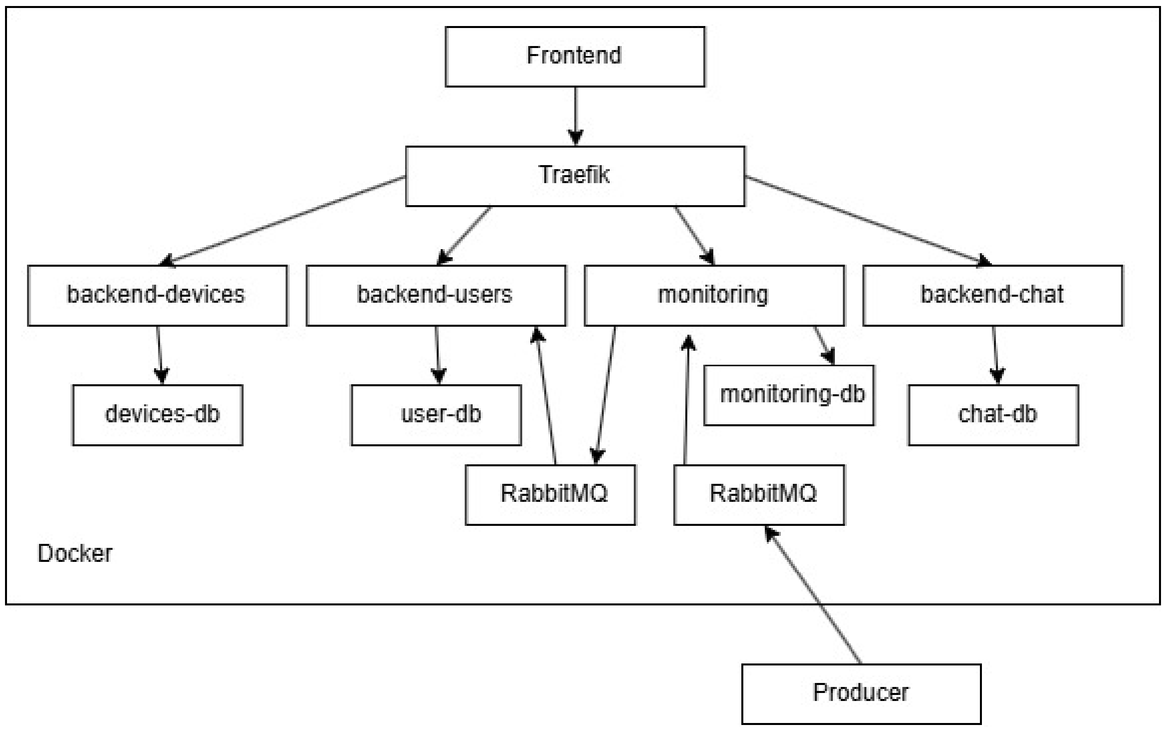
3.2 Key Features

3.2.1 Chat-Microservice

The real-time chat system enables instant messaging between admins and users using WebSocket technology, with messages stored in the database and displayed in real time on the frontend. Admins can efficiently communicate with multiple users simultaneously through broadcast messaging, selecting recipients and sending messages to all chosen users at once. Notifications enhance user engagement by alerting participants to new messages.

3.2.2 JWT

The chat system is secured with JWT (JSON Web Token) authentication, ensuring that only authorized users and admins can access its features. When logging in, each user or admin receives a unique JWT token, which is included in WebSocket headers and API calls for secure communication. Role-based access control is enforced through the token's claims, distinguishing between admin and user privileges. Tokens are rigorously validated for every connection and request, and any expired or invalid tokens immediately result in disconnection or request rejection, maintaining a safe and secure environment for real-time communication.


Frontend
Traefik
backend-devices
backend-users
monitoring
backend-chat
devices-db
user-db
monitoring-db
chat-db
RabbitMQ
RabbitMQ
Docker
Producer