UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Energy management system

Catinean Andrei

This application is designed to manage users and devices through a web interface. The system features a **frontend** implemented with **Next.js** and multiple backend services handling users and devices. The app architecture follows a **microservices** approach, where separate services manage distinct functionalities such as users and devices. It interacts with databases and allows admin users to add users/devices also allows normal users to see their personal devices. It features a Monitoring Backend for processing energy data, as well as a Device Producer to simulate smart meters.

System Architecture

The system architecture consists of two main components:

1. **Frontend (Next.js)**
2. **Backend Services (User backend and Device backend and Monitoring backend)**

Each of these components interacts with their respective databases. Below is a breakdown of each component, followed by a high-level description of how they work together.

1. Frontend (Next.js)

The **frontend** is responsible for rendering the user interface in the browser. Users can log in as admin or normal user and view either the **admin dashboard** (if admin) or their **personal devices**. This interface interacts with backend services to retrieve data related to users and devices.

- **URL**: The application runs locally on localhost:3000, accessible through the browser.
- **Framework**: **Next.js** is used to build the frontend.

2. Backend Services

a. User Backend Service

The **User Backend** is responsible for handling all user-related functionalities, including:

- User authentication (via login).
- Fetching user data from the database.
- Sending requests to Device Backend to synchronize the users information

It uses the following components:

- **Controller (User)**: Handles HTTP requests and routes them to the service layer.
- **Service (User)**: Contains the business logic for managing users.
- **DAO (User Entity)**: Responsible for interacting with the **User Database**.
- **User Database**: Stores user information, hosted at 172.18.0.3:5434.

The backend runs on 172.18.0.5:8080 and is part of the 172.18.0.0/16 subnet.

b. Device Backend Service

The **Device Backend** handles all interactions with the connected devices, including:

- Fetching device details for users.
- Managing device CRUD.

It is composed of:

- **Controller (Device)**: Receives HTTP requests for device actions.
- **Service (Device)**: Processes device-related business logic.
- **DAO (Device Entity)**: Manages interactions with the **Device Database**.
- **Device Database**: Stores device data at 172.18.0.2:5435.

The device backend is hosted at 172.18.0.4:8081 and operates within the same subnet as the user backend.

c. Monitoring Backend

The **Monitoring Backend** is a new service responsible for energy data processing and user notifications.

- **Responsibilities**:
  - **Message Consumer**: Processes messages from the message broker containing energy data.
  - Computes hourly energy consumption for each device.
  - Stores computed energy values in the Monitoring Database.
  - Sends WebSocket notifications to users if energy consumption exceeds thresholds.

d. Standalone Device Producer

The **Device Producer** is a standalone application that simulates smart meters, sending energy data to the Monitoring Backend via the message broker.

4. Message Broker Middleware

The **message broker** facilitates communication between the Device Producer and the Monitoring Backend and also between Device Backend and Monitoring Backend.

- **Responsibilities**:
  - Queues messages from the Device Producer.
  - Enables event-based synchronization between microservices (e.g., device updates).

5. Chat Microservice

The Chat Microservice enables real-time communication between users and administrators. This allows users to ask questions and receive answers efficiently.

Functional Requirements:

- Users see a chat box in the front-end application where they can type messages.
- Messages are sent asynchronously to administrators, who receive the message along with the user's identifier.
- Administrators can respond to users, initiating two-way communication.
- Notifications are displayed to:
  - Notify users and administrators when the other party reads a message.
  - Indicate when the other party is typing a response.
- Administrators can chat with multiple users simultaneously.

Implementation:

- Technology: WebSocket technology is used for real-time communication.
- Hosted as an independent microservice within the backend architecture.
- Integrated with User Backend Service for user identification and authentication.

6. Authorization Component

The Authorization Component secures access to the system's microservices, ensuring that only authenticated and authorized users can perform specific actions.

Functional Requirements:

- Implements user authentication and authorization using **Spring Security OAuth2** and **JWT**.
- Integrates with the User Backend Service to handle user sessions and validate access tokens.
- Secures RESTful endpoints in all backend microservices, restricting access based on user roles (e.g., admin or normal user).

Implementation:

- Technology: Spring Security OAuth2 JWT.

Data Flow

1. **User Login**:
    - The user accesses the application via localhost:3000 in the browser.
    - The frontend sends a login request to the **User Backend Service**.
    - The user backend verifies credentials by querying the **User DB**.
    - Upon successful authentication, the user is directed to either the admin dashboard or personal devices interface based on their role.
2. **Admin Dashboard**:
    - Admins can view overall system data, including information about users and devices.
    - They can add new users and devices or update existing ones.
3. **Personal Devices**:
    - Personal users can view and manage their devices.
    - The **Device Backend Service** fetches device details from the **Device DB**.
    - The frontend displays the data.
4. **Monitoring:**
    - If energy consumption exceeds a device's threshold:

      Monitoring Backend sends a WebSocket notification to the frontend.

  - Users are alerted in real time on the web interface.

5. Chat:

- A user initiates a chat session by typing a message in the chat box.
- The message is transmitted via WebSocket to the Chat Microservice.
- The administrator receives the message along with the user's identifier and responds.
- Notifications update both parties about message status (read/typing).

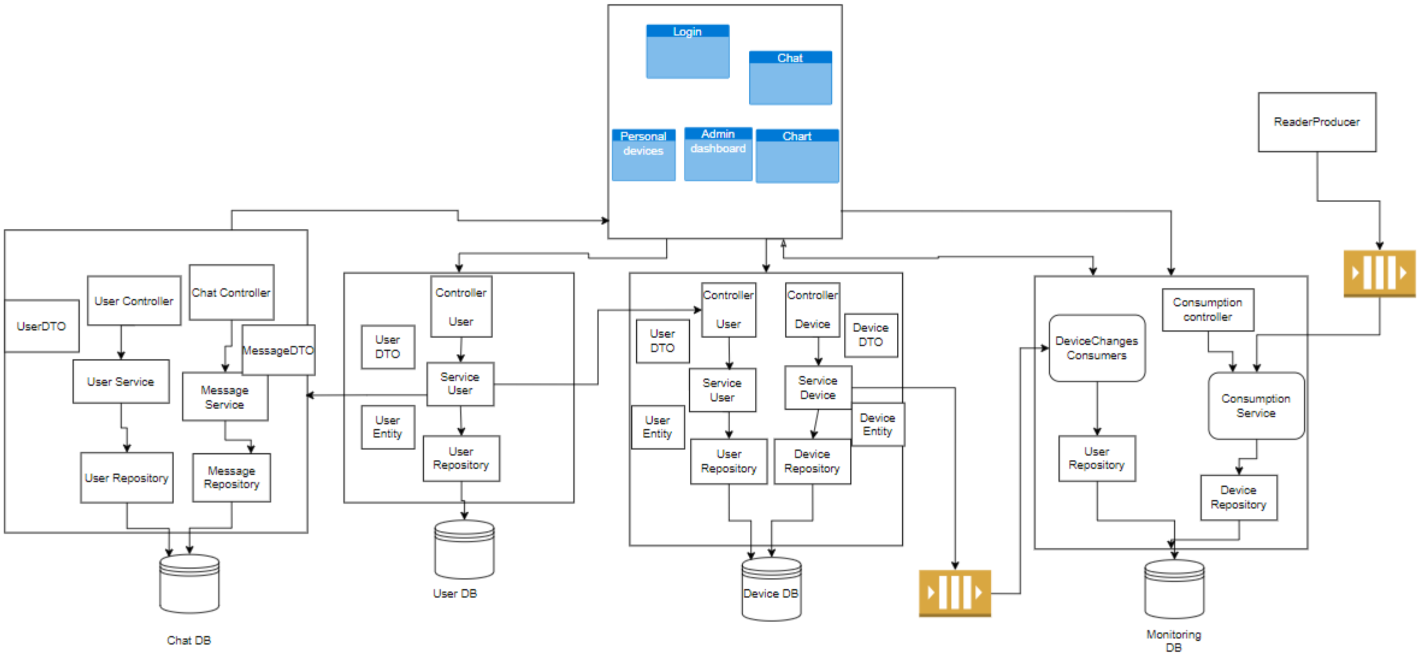
6. Authorization:

- Upon login, the User Backend generates a JWT token for the user.
- The token is included in all subsequent requests to other backend services.
- Backend services validate the token to ensure the user's identity and role before processing the request.

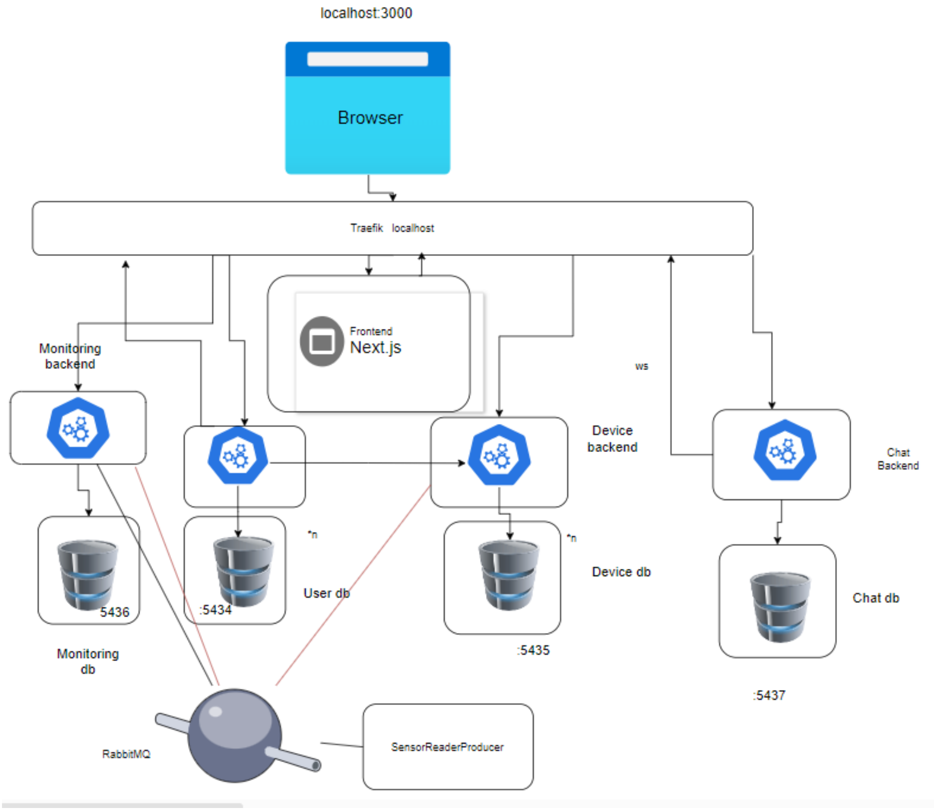
Technology Stack

- **Frontend**:
  - Framework: **Next.js**
- **Backend**:
  - Framework: Java Spring-Boot
  - Communication via RESTful APIs.
- **Databases**:
  - User DB: Stored at 172.18.0.3:5434.
  - Device DB: Stored at 172.18.0.2:5435.
- **Chat Microservice:**
  - **Technology**: WebSocket
- **Authorization:**
  - **Technology**: Spring Security OAuth2 JWT
- **Device Producer**
  - Language: **Java**
  - Data Format: **JSON**
  - Middleware: **RabbitMQ**

Conceptual architecture of the system:



Deployment diagram:


localhost:3000
Browser
Traefik localhost
Frontend
Next.js
Monitoring backend
Device backend
ws
Chat Backend
5436
:5434
*n
User db
*n
Device db
Chat db
Monitoring db
:5435
:5437
RabbitMQ
SensorReaderProducer