Energy Management System
Assignment 3
Student: Enache Mihai
Group: 30643

Table of Contents

I. Project Specification

Develop a chat microservice and an authorization component for the Energy Management System. The authorization component should provide secured access of users to systems' microservices. The chat microservice should allow communication between the users and the administrator of the system, allowing them to ask questions and receive answers.

Chat microservice:

➢ The front-end application displays a chat box where users can type messages.

➢ The message is sent asynchronously to the administrator, that receives the message together with the user identifier, being able to start a chat with the user.

➢ Messages can be sent back and forth between the user and the administrator during a chat session.

➢ The administrator can chat with multiple users at once.

➢ A notification is displayed for the user when the other administrator reads the message and vice versa.

➢ A notification is displayed for the user (e.g., typing) while the administrator from the other end of communication types of its message and vice versa. Authorization component:

➢ One of the services is chosen as authorization server (e.g. User Microservice, or a newly implemented microservice for authorization and authentication). This service generates access tokens to the client application. The tokens will be used to access other microservices.

Implementation technologies

➢ Chat component: web sockets technology

➢ Authorization component: JWT based authorization - for user's authentication and authorization to all microservices.

Choose between the 3 proposed design architectures from the laboratory presentation:

o One authorization service that generates tokens that will be recognized by other microservices which share the same secret key as the authorization service

o One authorization service that generates tokens and is configured as reverse proxy for all services. The services behind the reverse proxy do not need authorization, since they are protected from outside access by the LAN

o Spring OAuth2 – implement an OAuth 2 authorization service to grant access to all microservices

1. User Requirements:
    - The platform must allow users to create accounts and login.
    - A user that has an administrator role must be able to perform CRUD operations on user accounts, devices and on the mapping of users to devices and chat with the clients.
    - A user that has a client role must be able to see his devices, get notifications when a device exceeds its hourly consumption limit and chat with admin.
2. Technical Requirements:
    - The backend of the web application will be built using Java and Spring Boot.
    - The frontend of the web application will be built using Angular.
    - The web application will use PostgreSQL for the database management system.
3. Non-Functional Requirements:
    - Microservices: User Management Microservice, Device Management Microservice, Monitoring and Communication Microservice, Chat Microservice
    - Security: use authentication to restrict users to access the administrator pages (cookies, session, etc.)

II. Elaboration

1. Domain Model

- **User**: This entity represents a registered user of the application. It contains attributes such as name, uid, email, password and the role he has.

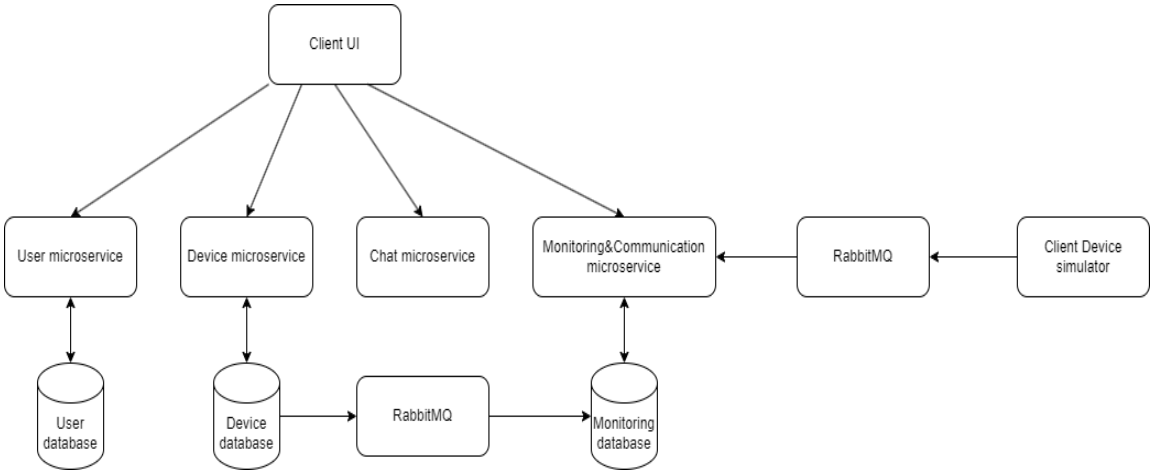
- **Device**: This entity represents a registered device of a user. It contains attributes such as description, address, maximum hourly energy consumption and user uid (for both Device Management and Monitoring & Communication Microservices)

- **Consumption**: This entity represents a hourly consumption for a device. It contains attributes such as id, device uid, hourly energy value (Monitoring & Communication Microservice).

2. Architectural Design

2.1 Conceptual Architecture

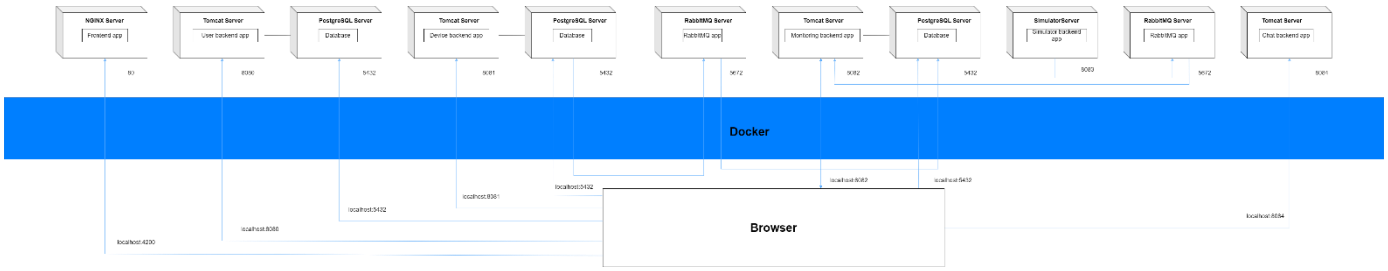
The application is designed as a collection of small, loosely coupled microservices, each responsible for specific functionalities. Microservices enable scalability, maintainability, and flexibility. They allow for independent development, deployment, and scaling of services, which aligns well with modern, dynamic software development practices. Each microservice communicates with the others using RESTful APIs, adhering to the principles of Representational State Transfer (REST). In comparison to the previous theme, an additional microservice has been implemented for monitoring device consumption. Two RabbitMQ queues have been employed, one for communication between the microservice handling devices and the monitoring microservice, and the other for transmitting data from the simulators to the monitoring microservice.



Conceptual Architecture diagram

2.2 Deployment Diagram

For the application, a local deployment has been set up using Docker. This involves utilizing an Nginx proxy server for the frontend and deploying the backend using a JAR file. Dockerfiles have been written for both the frontend and backend components.



Deployment diagram

III. Elaboration – Iteration 1.2

1. Data Model

1. **User Entry Model**: This model stores information about a user and is used for managing user-related functionalities, authentication and access control within the system.
2. **Device Entry Model**: This model stores information about a device and is used for managing device-related functionalities. This model is also used in the Monitoring and Communication Microservice.
3. **Consumption Entry Model**: This model stores information about the hourly energy value for devices and is used for visualization of the device consumptions.

2. Testing

For this project, the system has been tested using manual testing. I performed exploratory testing and verified that the software meets the specified requirements and user expectations. Even though I did not use any automated testing strategies (such as unit testing or validation testing), I made all the validations necessary so that the system is correctly implemented. Every user input is validated before sending the information to the database. For the validation of the user input I used method like data-flow, partitioning and boundary analysis to ensure that the designed system has no errors.

3. Security

For this project, I set up a solid security system with the help of Spring Security, JWT Tokens and Auth Guard. I made sure that administrators and clients have different access rights (each user has a role and each role has a set of permissions). When a user with a client role tries to access an admin-only link, the Auth Guard redirects the user to a forbidden page. The JWT Token has two purposes: to authenticate users and authorize them to access specific resources based on their role. I also used session storage to hold onto the JWT token and the user's role as they navigate through the app. For this assignment, security has been added also for the Device and Monitoring microservices as described above.

For securing the Chat microservice, I have implemented the following logic: when a user wants to access the chat page, a request will be made. The JWT token will be sent in the request's header for user authentication. If the token is valid, the server generates a temporary external authentication token, stores it in the Authentication Cache and returns it to the client. The client makes a WebSocket handshake request with the external authentication token passed as a query string parameter in the handshake endpoint URL. The server checks the cache to see if the external authentication token is valid. If valid, the handshake is established and the HTTP upgrade occurs to the WebSocket protocol. The new authentication token is cached. The client has now been authenticated and bidirectional communication can now occur. For this microservice, I used interceptors for the requests that are made.

4. Implementation Details

Compared to the previous assignment, we have implemented an additional microservice dedicated to monitoring device consumptions. Each device maintains an hourly consumption list. To synchronize databases between the device microservice and the monitoring microservice, we implemented a Message Broker based on a queue using RabbitMQ. The data synchronization mechanism between the databases of the two microservices is triggered in three cases: when the administrator updates details related to a device, deletes a device, or maps a device to a user. We implemented the simulator using threads. UID values for devices to simulate are read from a .txt file. Each thread reads a UID from the file, then reads values from the .csv file. Each thread sends messages to the monitoring microservice using another queue. Two values are read from the measurement file, the difference between the measured values is calculated, and the necessary information is sent to the monitoring microservice. Since values are read every 10 minutes, the sixth received value represents the device's hourly consumption. Even if multiples of six values are not read, the consumption is updated each time a message is received from the simulator. For each received message, it is checked whether the hourly consumption exceeds the maximum limit

for a device, and if so, a notification is sent to the client using WebSocket. On the browser, in the top right corner, the client can see a notification alerting them to which device has exceeded the consumption limit. For this assignment, I had to implement the security on the Device, Monitoring&Communication and Chat microservices. For the Device and Monitoring&Communication microservices I used Spring Security for securing all the endpoints. The Chat microservice has been secured as explained in the Security section.
The reading and typing notification has been implemented using WebSockets. When the user clicks on the text box in the UI, a message will be sent to the other user and the "Seen" text will appear. For the typing notification, every time a user is typing, a message will be sent the "X is typing" will appear on the UI, where X is "Client" or "Admin".

IV. Bibliography

https://www.toptal.com/spring/spring-security-tutorial

https://sailssoftware.com/auth-guard-in-angular/

https://www.yourteaminindia.com/blog/angular-authentication-using-route-guards

https://www.npmjs.com/package/ngx-toastr

https://www.baeldung.com/rabbitmq-spring-amqp

https://sailssoftware.com/auth-guard-in-angular/
https://www.yourteaminindia.com/blog/angular-authentication-using-route-guards
https://www.npmjs.com/package/ngx-toastr
https://www.baeldung.com/rabbitmq-spring-amqp