Department of Computer Science

Technical University of Cluj-Napoca

Distributed Systems

Laboratory project

Name: Onaci Andreea-Maria
Group: 30441

Contents

Chapter 1

Project specification

1.1 Assignment 1

Design an Energy Management System that includes a user-friendly interface and two separate services for managing users and their smart energy meters. The system will allow two types of users: admins and clients, each of the types need to log in. Administrators can manage user accounts, manage smart energy meters, and assign devices to users, ensuring that each user can have multiple devices.

After logging in, users are redirected to a page specific for their role in the Energy Management System. Administrators have full access to manage user accounts and devices, allowing them to create, read, update, and delete records for both microservices. They can also establish a mapping between users and their respective devices. On the other hand, clients can view all their assigned devices on their dedicated page. To maintain a minimum security, users cannot access pages designated for other roles; for example, if a client tries to access an administrator's page by simply entering the URL, they will be redirected to a blank page with a message that is informing them that they must be logged in to access that section. This ensures that each user interacts only with the features relevant to their role.

1.2 Assignment 2

The conceptual architecture for the Monitoring and Communication Microservice is centered around RabbitMQ as the core message-oriented middleware, enabling seamless data exchange between components. A Smart Metering Device Simulator acts as the producer, reading sensor data from a CSV file and sending JSON-formatted messages containing timestamped energy measurements to a RabbitMQ queue. The Monitoring and Communication Microservice includes a consumer component that subscribes to this queue, processes the incoming data, computes hourly energy consumption, and stores the results in its database. Device updates from the Device Management Microservice are synchronized through an event-based system using RabbitMQ topics. Also on the Client page, the client can see the chart corresponding to the chosen device from that ones assigned to him. Also, the application provides the feature that you can run it from 2 different browsers at the same time using different clients that are logged in.

1.3 Assignment 3

The requirements outline the development of a Chat Microservice and an Authorization Component for the Energy Management System. The Chat Microservice facilitates real-time communication between users and administrators through a chat interface, leveraging WebSocket technology for asynchronous messaging. It supports multi-user chat sessions, read receipts, and typing indicators to enhance interaction. The Authorization Component ensures secure access to system microservices by implementing user authentication and authorization using Spring Security OAuth2 and JWT, integrated into the User Management Microservice. Together, these components improve system usability, security, and engagement for both users and administrators.

Chapter 2

Description of the conceptual architecture

2.1 Assignment 1

On conceptual architecture, we have in the center two microservices: one for managing users and another for managing devices. These microservices operate independently but can communicate with each other to ensure consistency.

Users interact with the system through a frontend interface where they can log in. Depending on their role, they are redirected to different dashboards. Admins can perform various tasks, like managing user accounts and devices, while clients can simply view their assigned devices.

Data is stored in 2 databases, allowing both microservices to efficiently perform their operations. The system ensures via APIs that frontend and backend services are working properly, ensuring that users can only access the features appropriate to their role. This architecture is designed for future development, meaning that we can add more features without disrupting the functionality of the already existing implementation.

2.2 Assignment 2

The Monitoring and Communication Microservice Architecture efficiently manages energy data using RabbitMQ for communication. The MonitoringController serves as the entry point, providing endpoints to receive device measurements, retrieve energy consumption data, and check thresholds. Incoming messages from RabbitMQ, formatted as JSON (timestamp, device_id, measurement_value), are processed by the MonitoringService, which stores data in a dedicated database, calculates hourly energy usage, and performs threshold checks. This architecture ensures seamless data flow and scalability for future enhancements.

The frontend part manages user interactions by fetching energy data for a selected device and date through the `energyService`. It updates the component's state (`deviceId` and `showChart`) and calls the `updateChart` function once the data is retrieved, rendering the chart dynamically. This seamless integration of data processing and error handling visualization provides users with an interactive and intuitive interface to monitor energy usage effectively.

2.3 Assignment 3

The conceptual architecture of the system is designed to ensure real-time communication and secure user access across the Energy Management System. At the forefront is the Frontend Application, which serves as the user interface, displaying a chatbox for both users and administrators to exchange messages in real time. It handles WebSocket connections to facilitate seamless communication, providing features like typing indicators and read receipt notifications. The Chat Microservice is the backbone of the chat functionality, managing message routing and ensuring proper delivery to the intended recipients. It supports multiple concurrent chat sessions, allowing administrators to interact with several users simultaneously while providing notifications for message status updates.

To secure the system, the Authorization Component leverages Spring Security OAuth2 with JWT-based authentication, ensuring that only authorized users can access system resources. This component integrates with the User Management Microservice, which handles user data, roles, and permissions. Together, they validate user credentials and issue tokens for secure access to various microservices. The Administrator Interface complements this setup by enabling administrators to manage multiple chat sessions efficiently, viewing user messages alongside identifiers and responding through the Chat Microservice. Finally, a centralized Database stores chat messages, timestamps, read statuses, and typing indicators, while also serving as the repository for user data required by the authorization and user management components. This architecture ensures secure, scalable, and real-time communication between users and administrators, aligning with the system's functional requirements.

Chapter 3

Conceptual architecture diagram

3.1 Assignment 1

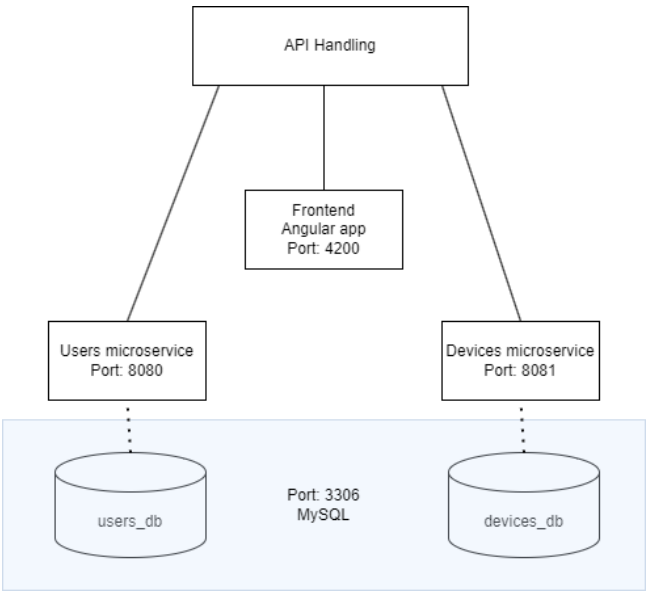

Figure 3.1: Conceptual architecture diagram

Chapter 4

Conceptual architecture diagram

4.1 Assignment 2

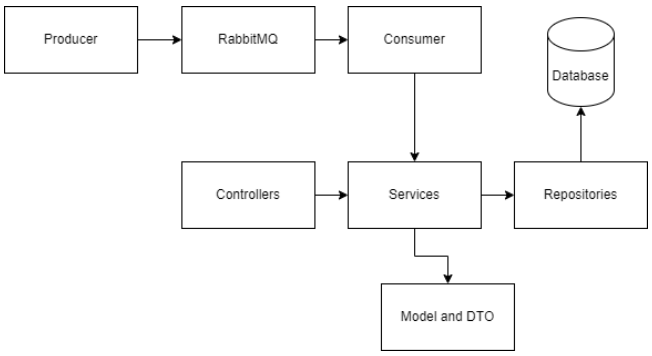

Figure 4.1: Conceptual architecture diagram

Chapter 5

Conceptual architecture diagram

5.1 Assignment 3

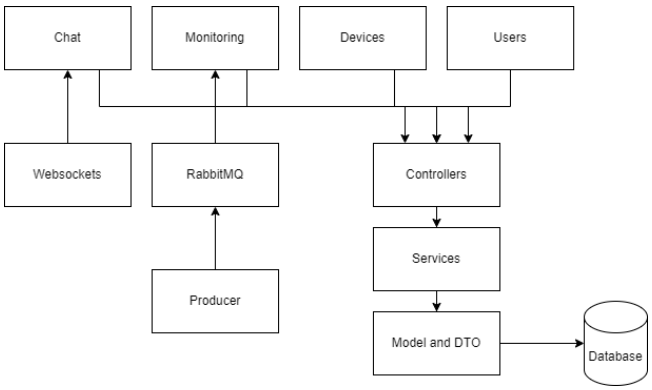

Figure 5.1: Conceptual architecture diagram

Chapter 6

Deploy diagram

6.1 Assignment 1

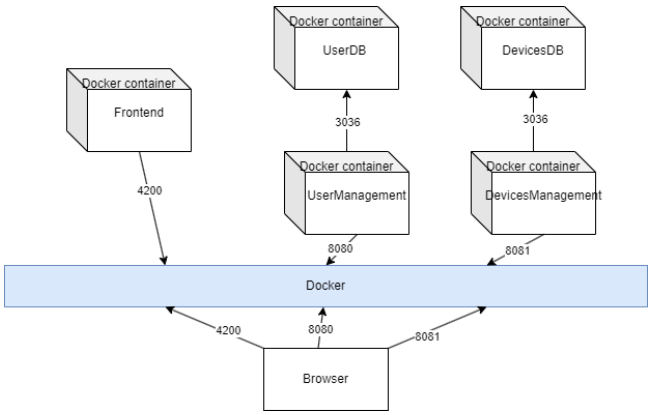

Figure 6.1: Deploy diagram

6.2 Assignment 2

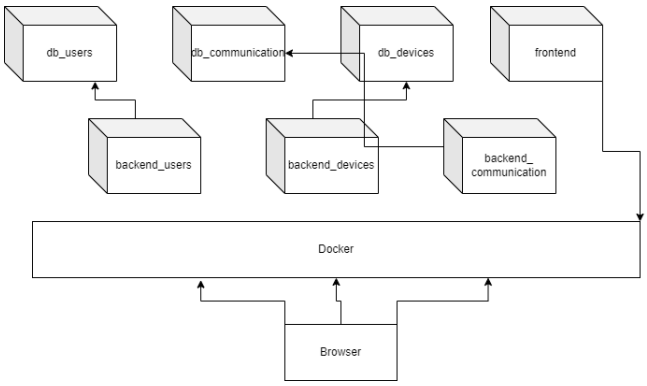

Figure 6.2: Deploy diagram

6.3 Assignment 3

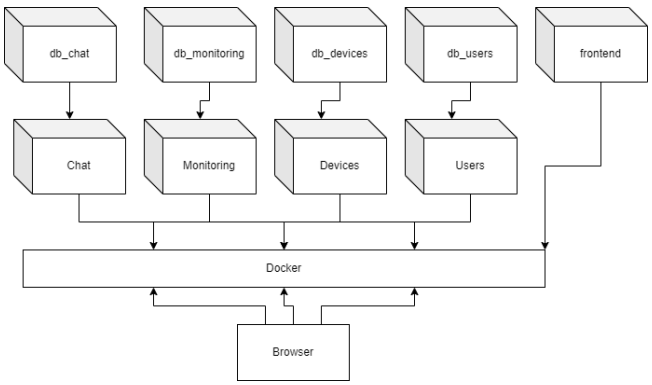

Figure 6.3: Deploy diagram

Chapter 7

Readme

For local running, take the corresponding src for each microservice and add it in a new project created for each of them in the IDE. For the frontend part, it is deployed in Angular, so we need to make sure on having npm installed and also ng.

The application is deployed on Docker, so first we need to run Docker with the command

```
docker-compose up --build
```

This command ensures that the containers are created and also in a running state. After creating the containers, in the same folder we open a new command prompt and create the databases with the following command

```
docker exec -it name-of-db mysql -u root -p
```

and so, we get a MySQL console where we paste the database console to create the database and do the same for both databases. After that, we can run the containers and we should have no error. Access `http://localhost:4200/login` and enjoy the navigation and functionalities of the application!

7.1 Assignment 2

For this, you need to create an account here https://kangaroo.rmq.cloudamqp.com//queues and to follow the tutorial on creating a queue in RabbitMQ. After, you need to run a producer (Device simulator) and then to run the rest of the application in docker.

7.2 Assignment 3

To use the system, log in through the frontend to receive a JWT token for secured access. Users can access the chat feature to send messages to the administrator, who can respond and manage multiple conversations in real-time. Notifications for typing and read receipts enhance the chat experience. Ensure the backend services are running, and WebSocket endpoints are accessible for seamless communication.