UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

DISTRIBUTED SYSTEMS

Assignment 3

Energy Management System

Student: Drăguș Andreea

Grupa: 30643

1. Conceptual architecture of the distributed system

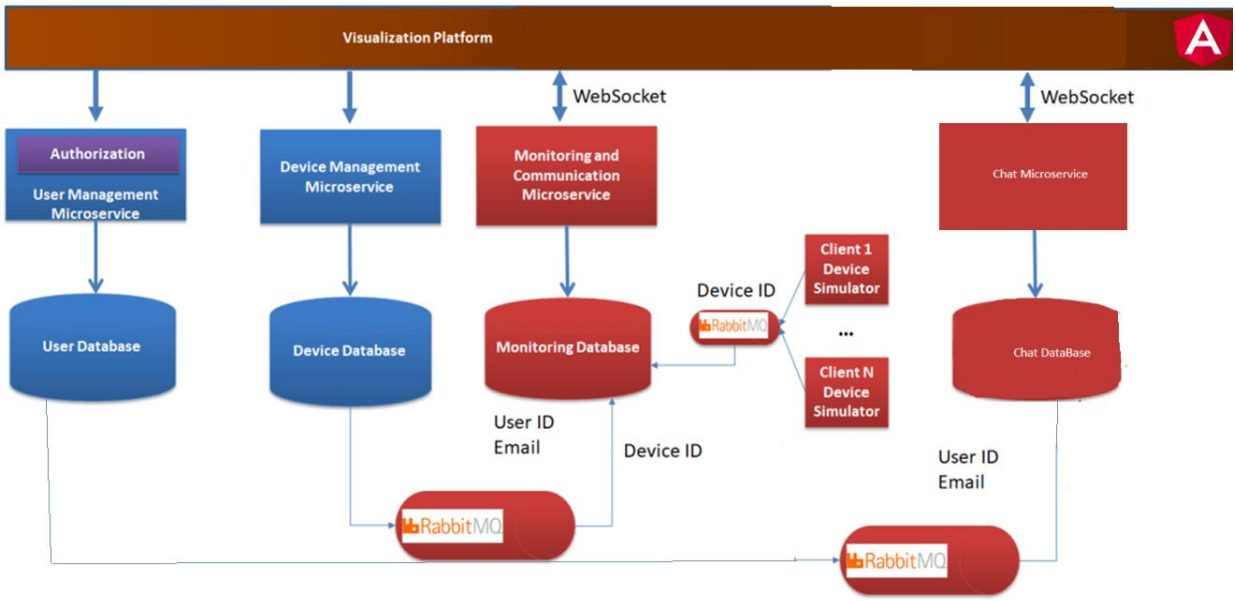
The architectural model of the distributed system is based on microservices. The microservice added in this assignment represents a chat where administrators can chat with customers. An administrator can chat with several clients simultaneously. When the user selects from the list the username of the other user with whom he wants to start a conversation, he connects to a websocket whose address has the following structure: private-chat/adminUid/clientUid. Thus, the possibility of multiple communication is ensured.

After establishing the connection to the websocket, each user automatically connects to it in order to receive the messages that are sent on that channel. Thus, when one of the users presses the send button, the message is sent through the websocket and is intercepted and displayed on the screens of the two users. The sent message is also intercepted in the backend where it is saved in a database so that users can see the messages sent over time.

The structure of the message saved in the database contains the following fields: id,senderUid, receiverUid, sentTimestamp, seen, seenTimestamp. The last two fields have the role of saving in the database the moment when the receiver of the message sees it. When the web page is rendered and the messages are displayed, it is checked whether each of them has been seen by the recipient or not. If it has not been seen yet, a notification is sent through another websocket announcing the fact that the respective message has been seen together with the timestamp when this was done.

The second type of websocket was used to notify the recipient when the sender writes a message. By capturing the input from the html in which the message is written, every time it is changed, a message is sent through another websocket that sends a "typing" type notification.

In order to ensure that the messages are sent to the appropriate users, a rabbitmq queue was added that synchronizes the user tables from the user service and chat service. Every time a user is added or deleted, a synchronization message is sent to the chat service. Thus, when receiving a message via websocket in the backend, it is checked whether the sender and the receiver are actually registered in the user table. If this is not approved, a CouldNotSendMessage exception is thrown.



The security is now extended to all microservices. When a user logs in, a jwt token is generated in the user-service and stored in the browser's local storage. When the user wants to make requests to any other microservice, the token generated at login is attached to the header. The other microservices also have the secret key used to encrypt the token, thus being able to decrypt it and check if it is a valid token, which has not expired and possibly the role of the user. The token is composed of user id, user role, username, creation timestamp.

2. Deployment

For the deployment part I used docker. For the deployment of backend microservices, first I generated the jars. Then, I created the dockerfiles.

The dockerfile starts by specifying a base Docker image that contains the minimal operating system, runtime environment required for the Java application and includes a compatible Java Development Kit (JDK-17 in this case). Then, I used COPY and ADD commands to include the source, the pom.xml, the code checker and the jar. After that I specified the working directory: This is where the JAR file will be placed and where the container will execute commands. I also used ENTRYPOINT which specifies the command that should be run when a container is created from the image (basically running the jar). I set as environment variables the information about database connection: username, password, port, etc.

After I created the dockerfile, I used the build command to create the image from it. After that, I used that image in the docker-compose file and included it in the backend container. The novelty compared to assignment 1 is the presence of a rabbitmq server whose role is to manage queues and messages. To deploy the monitoring microservice, we proceeded in the same way as in the case of user/device management microservices, as explained above. The user management microservice did not require a redeployment, unlike the device microservice and the front-end, which underwent changes.