Distributed Systems

Assignment 3

Remote Procedure Call

Chat System for Client Support

Ghise Nicolae-Mihai, Group 30443

1. Problem description

Develop a chat system to offer support for the clients of the energy platform if they have questions related to their energy consumption. The chat system should allow communication between the clients and the administrator of the system.

The client application displays a chat box where clients can type messages. The message is sent asynchronously to the administrator, that receives the message together with the client identifier, being able to start a chat with the client. Messages can be sent back and forth between the client and the administrator during a chat session. The administrator can chat with multiple clients at once. A notification is displayed for the user when the other user reads the message and when the other user is typing.

2. Technologies and solutions

gRPC is a high-performance, open-source framework for building remote procedure calls (RPC) APIs. It is based on the Protocol Buffers data serialization format and the HTTP/2 network protocol.

One of the key features of gRPC is its ability to use a contract-first approach to API development, where the API is defined using a protocol buffer file, and the framework generates the necessary code for the client and server implementations. This makes it easy to create APIs that are efficient, strongly typed, and easy to evolve.

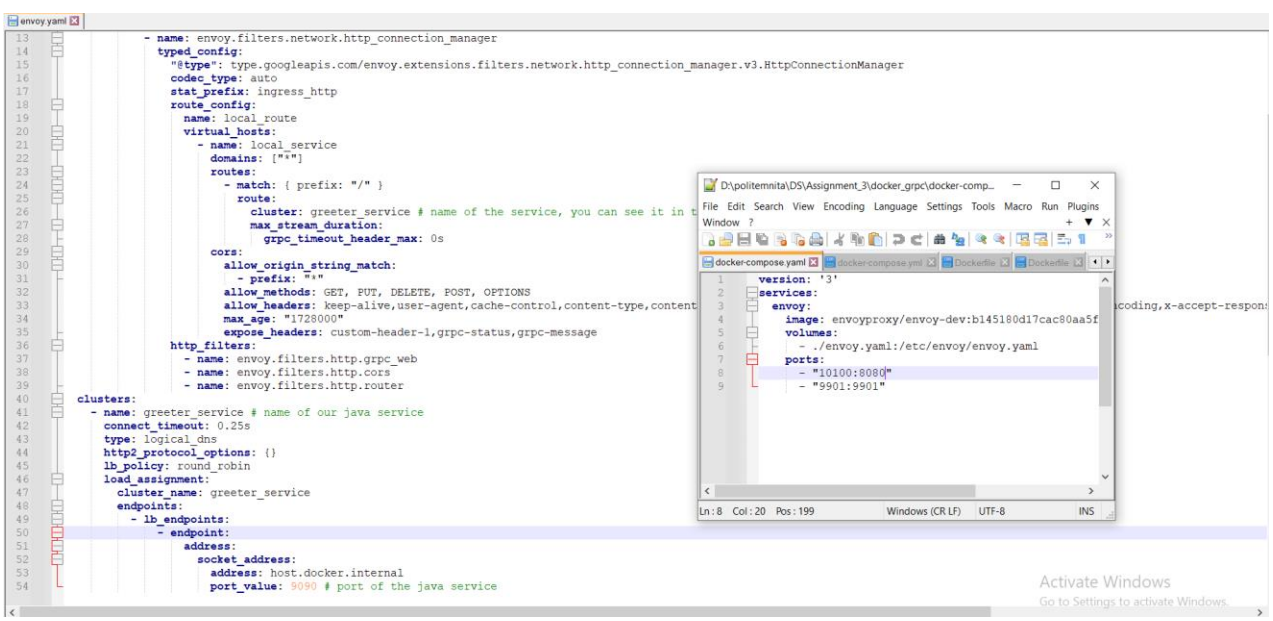
gRPC also supports bi-directional streaming, which allows for sending and receiving multiple messages at the same time, rather than in a request-response pattern. This makes it well suited for use cases such as real-time streaming data, where many messages need to be sent in quick succession.

3. Functional requirements

In order to achieve the goals of the assignment, I used gRPC, a server application based on remote procedure call with one distributed object supporting message send and receive primitives.

4. Deployment on docker

Deployment of gRPC proxy, because gRPC uses HTTP/2 and our base application uses normal HTTP



Docker file and docker compose file for the backend regarding the gRPC setup.

```
FROM maven:3.8.5-openjdk-17 AS builder

COPY ./src/ /root/src
COPY ./pom.xml /root/
COPY ./checkstyle.xml /root/
WORKDIR /root
RUN mvn package
RUN java -Djarmode=layertools -jar /root/target/demo-0.0.1-SNAPSHOT.jar list
RUN java -Djarmode=layertools -jar /root/target/demo-0.0.1-SNAPSHOT.jar extract
RUN ls -l /root

FROM openjdk:17.0.2-oracle

COPY --from=builder /root/dependencies/ ./
COPY --from=builder /root/snapshot-dependencies/ ./

RUN sleep 10
COPY --from=builder /root/spring-boot-loader/ ./
COPY --from=builder /root/application/ ./
ENTRYPOINT ["java", "org.springframework.boot.loader.JarLauncher","-XX:+UseContainerSupport -XX:+UnlockExperimentalVMOptions -XX:+UseCGroupMemoryLimitForHeap -XX:MaxRAMFraction=1 -X
```

D:\politemnita\DS\Assignment_3\gRPC_Spring\docker-compose.yml - Notepad++

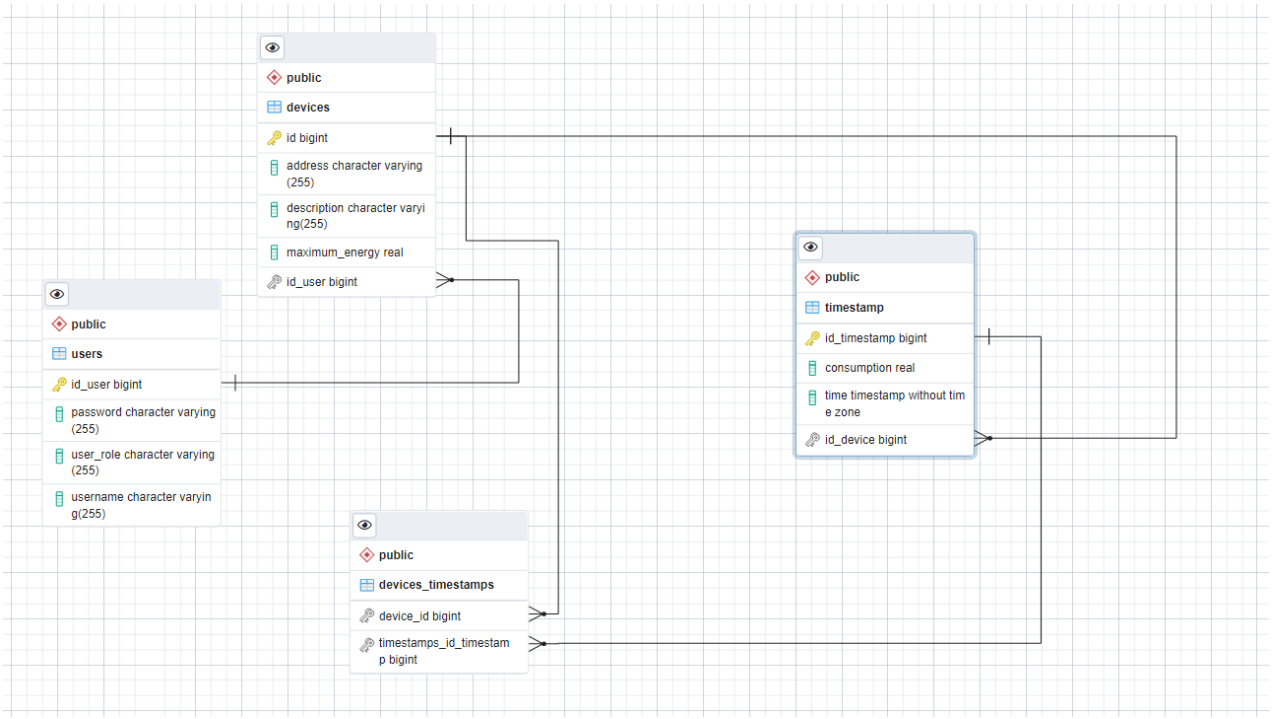
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

docker-compose.yml

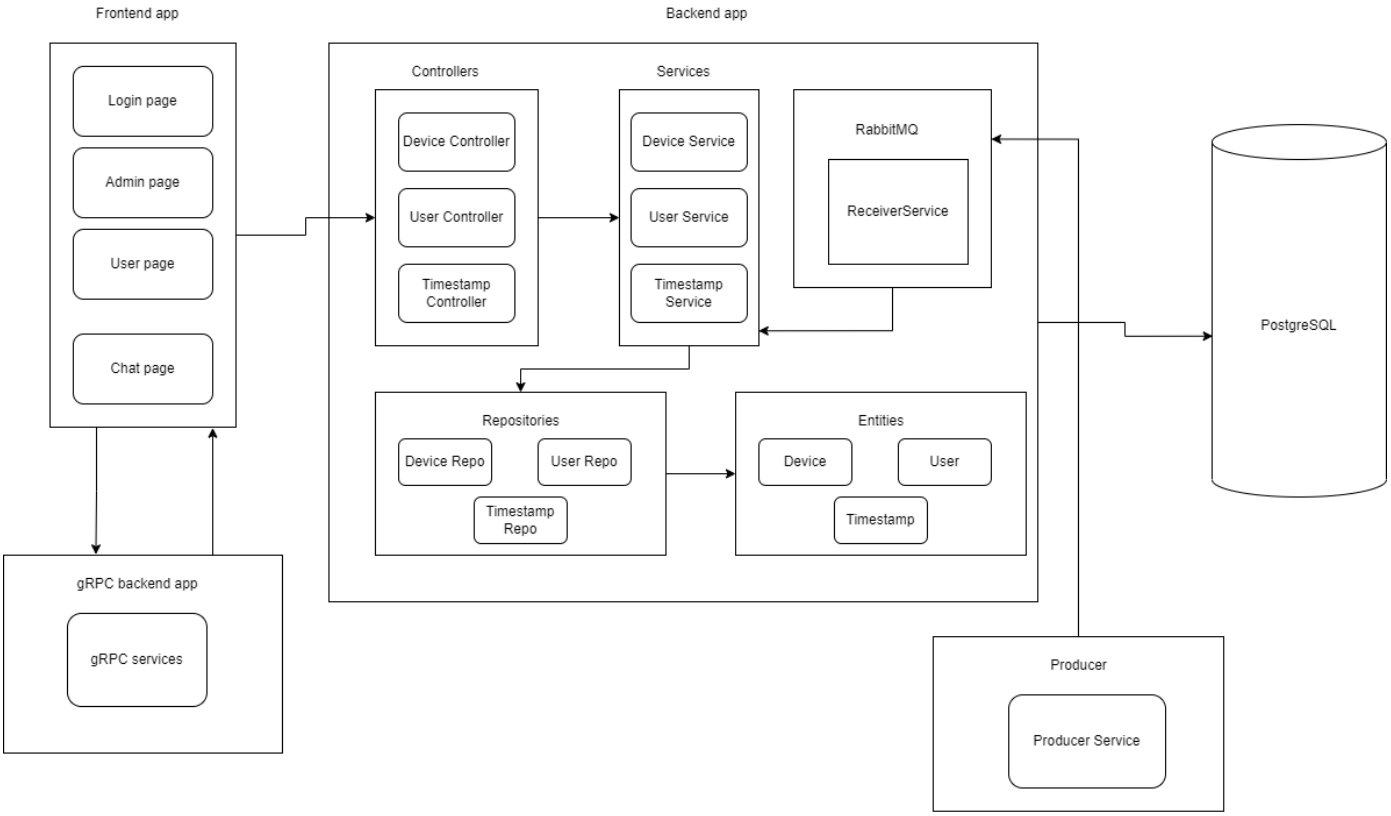
```
version: '3'

services:

  tomcat-db-api:
    image: grpc_be
    ports:
      - "9090:9090"
```

YAML Ain't Markup Lang length : 107 lines : 10 Ln : 5 Col : 17 Pos : 46 Windows (CR LF) UTF-8 INS

Database diagram:



Conceptual diagram:



Deployment diagram: