

# **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.

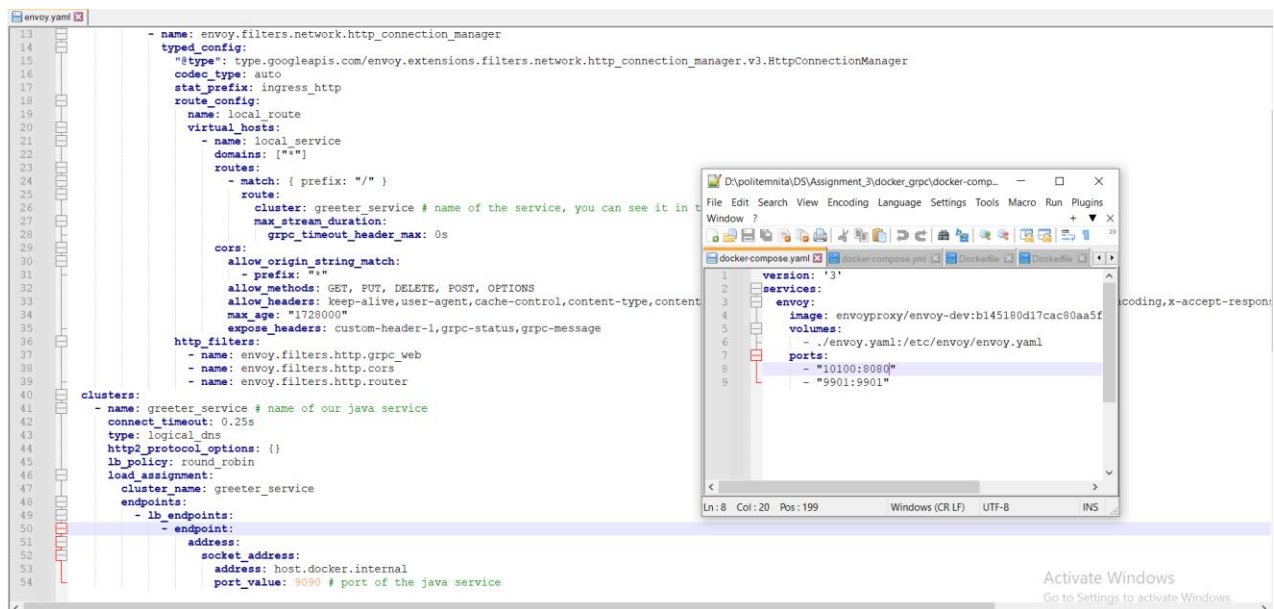
gRPC also supports bi-directional streaming, which allows for sending and receiving multiple messages at the same time, rather than in a request-response pattern. This makes it well suited for use cases such as real-time streaming data, where many messages need to be sent in quick succession.

### 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



```
envoy.yaml
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54

- name: envoy.filters.network.http_connection_manager
  typed_config:
    "@type": type.googleapis.com/envoy.extensions.filters.network.http_connection_manager.v3.HttpConnectionManager
    codec_type: auto
    stat_prefix: ingress_http
    route_config:
      name: local_route
      virtual_hosts:
        - name: local_service
          domains: ["*"]
          routes:
            - match: { prefix: "/" }
              route:
                cluster: greeter_service # name of the service, you can see it in
                max_stream_duration:
                  grpc_timeout_header_max: 0s
      cors:
        allow_origin_string_match:
          - prefix: "*"
        allow_methods: GET, PUT, DELETE, POST, OPTIONS
        allow_headers: keep-alive,user-agent,cache-control,content-type,content-transfer-encoding,x-accept-response
        max_age: "1728000"
        expose_headers: custom-header-1,grpc-status,grpc-message
    http_filters:
      - name: envoy.filters.http.grpc_web
      - name: envoy.filters.http.cors
      - name: envoy.filters.http.router

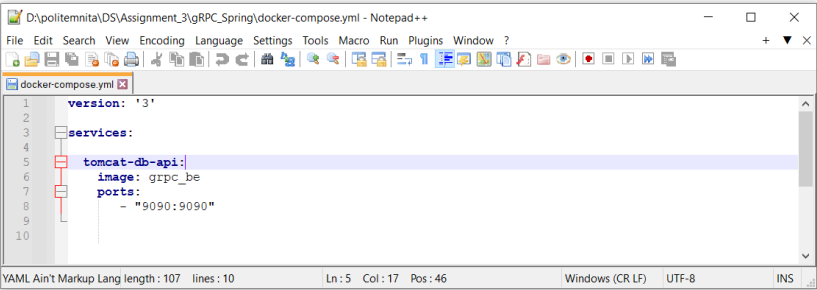
clusters:
  - name: greeter_service # name of our java service
    connect_timeout: 0.25s
    type: logical_dns
    http2_protocol_options: {}
    lb_policy: round_robin
    load_assignment:
      cluster_name: greeter_service
      endpoints:
        - lb_endpoints:
            - endpoint:
                address:
                  socket_address:
                    address: host.docker.internal
                    port_value: 9090 # port of the java service

docker-compose.yml
1
2
3
4
5
6
7
8
9

version: '3'
services:
  envoy:
    image: envoyproxy/envoy-dev:b145180d17cac80aa5f
    volumes:
      - ./envoy.yaml:/etc/envoy/envoy.yaml
    ports:
      - "10100:8080"
      - "9901:9901"
```

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

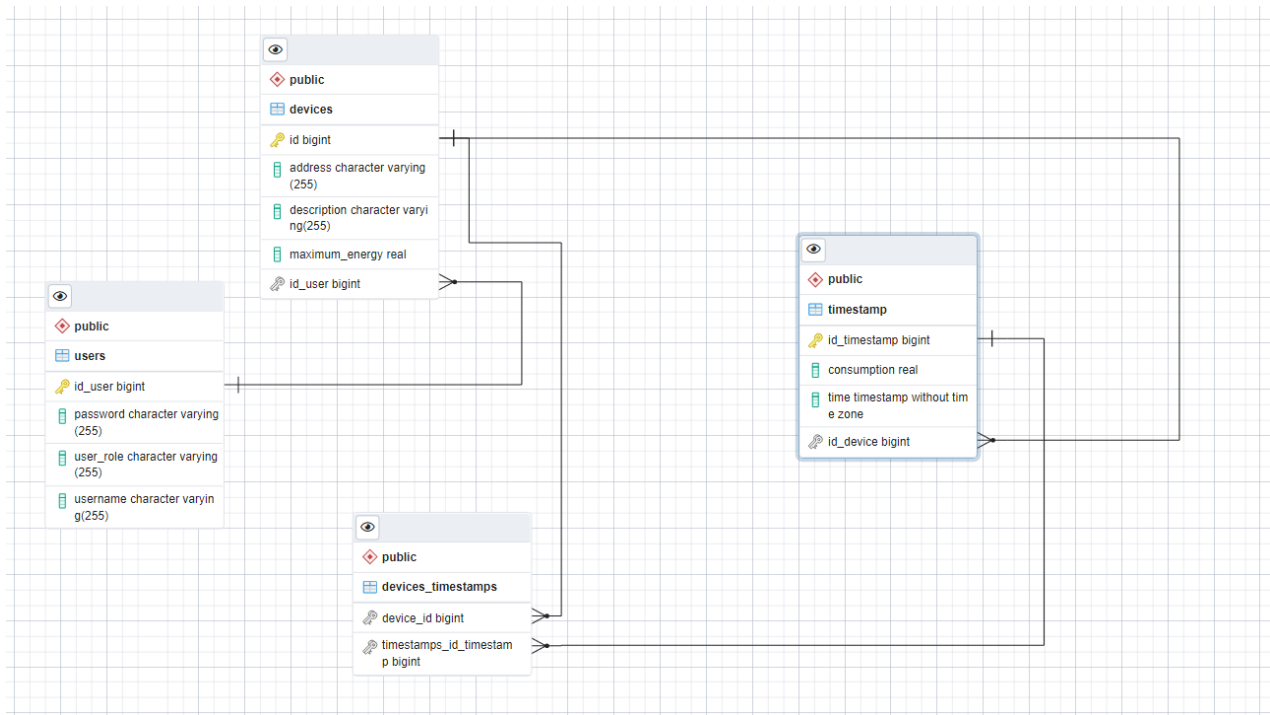
```
1 FROM maven:3.8.5-openjdk-17 AS builder
2
3 COPY ./src/ /root/src
4 COPY ./pom.xml /root/
5 COPY ./checkstyle.xml /root/
6 WORKDIR /root
7 RUN mvn package
8 RUN java -Djarmode=layertools -jar /root/target/demo-0.0.1-SNAPSHOT.jar list
9 RUN java -Djarmode=layertools -jar /root/target/demo-0.0.1-SNAPSHOT.jar extract
10 RUN ls -l /root
11
12 FROM openjdk:17.0.2-oracle
13
14 COPY --from=builder /root/dependencies/ ./
15 COPY --from=builder /root/snapshot-dependencies/ ./
16
17 RUN sleep 10
18 COPY --from=builder /root/spring-boot-loader/ ./
19 COPY --from=builder /root/application/ ./
20 ENTRYPOINT ["java", "org.springframework.boot.loader.JarLauncher", "-XX:+UseContainerSupport -XX:+UnlockExperimentalVMOptions -XX:+UseCGroupMemoryLimitForHeap -XX:MaxRAMFraction=1 ->
```



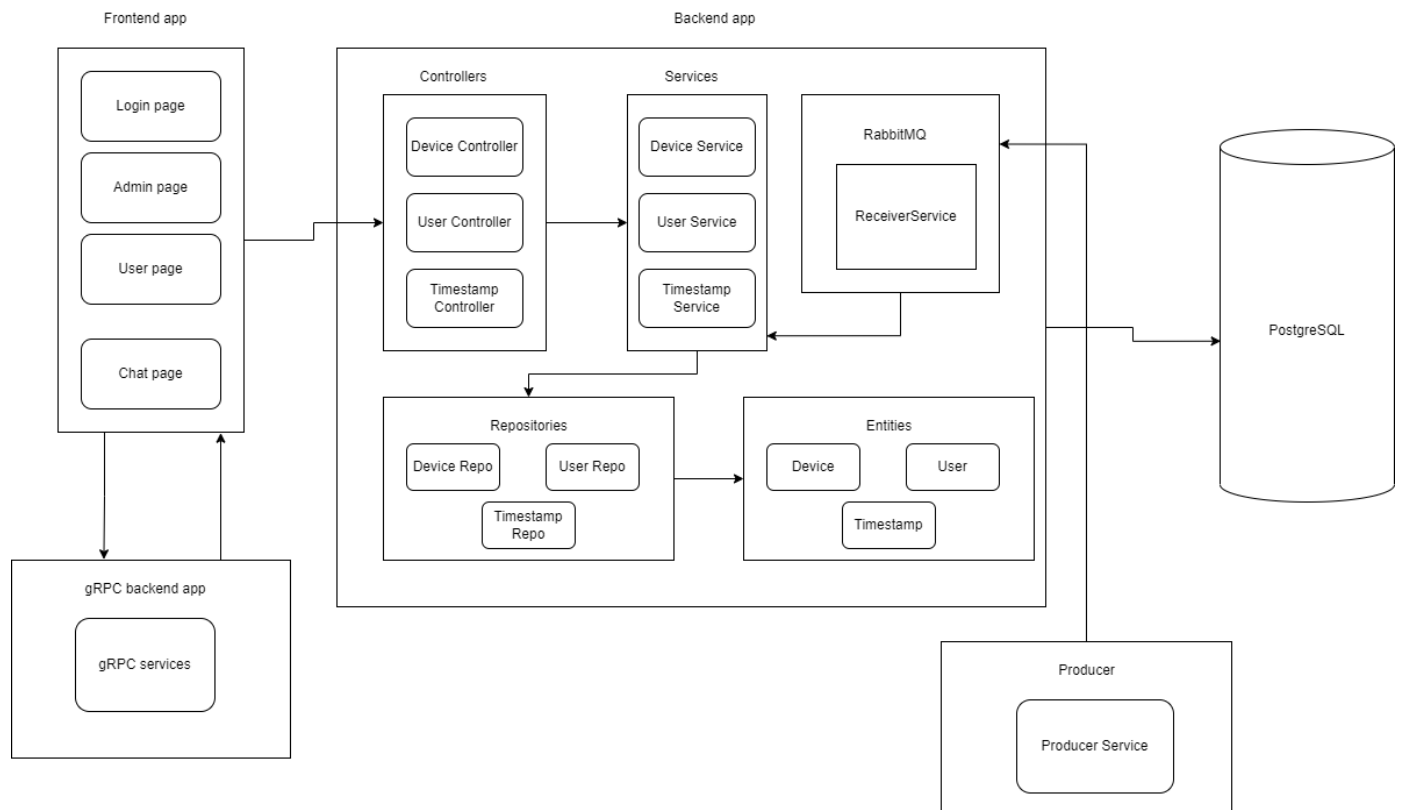
```
1 version: '3'
2
3 services:
4   tomcat-db-api:
5     image: grpc_be
6     ports:
7       - "9090:9090"
```

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:

