Distributed Systems

Assignment 1

Request-Reply
Communication Paradigm

Online Energy Utility Platform

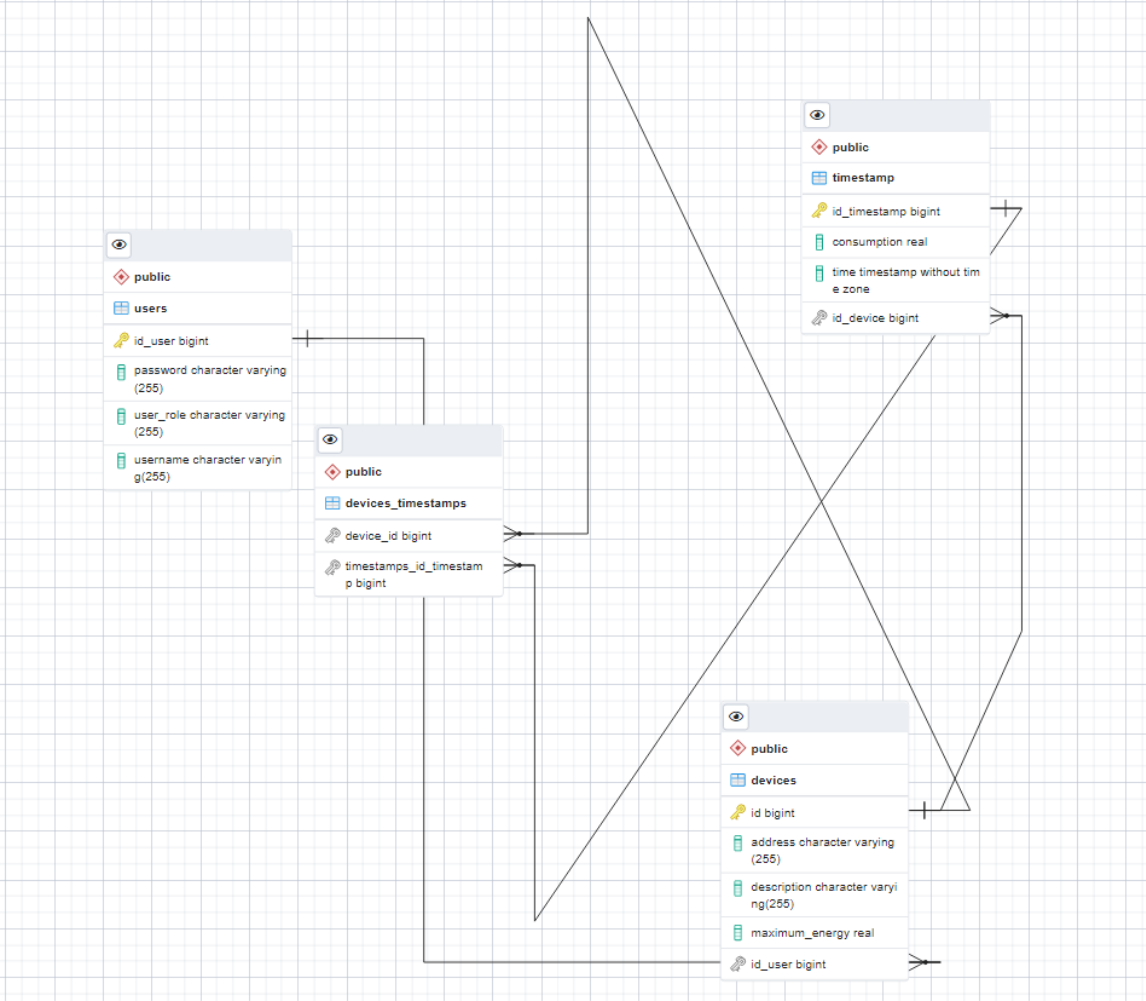
Ghise Nicolae-Mihai, group 30443

1. Problem description

An online platform should be designed and implemented to manage users, their associated devices and the monitored data received from each user. The system will be accessible by two types of users: a simple user and a more privileged user called an administrator. The former can view his devices and can visualize a chart for each device on which day he wants which will show the energy consumption for that specific day. The latter can view all devices, all users, can add/update/remove both categories and can associate devices to users.

2. Implementation technologies and solutions

I will be using REST services for backend application, more specifically Java Spring and for the front-end part I will be using Angular. First, I started by creating the database and the entities using hibernate from Java. A user has a username, a password, a role and a list of devices. The device has a description, an address and a maximum energy consumption. The mappings are rather simple, a user can have more devices, but one device can be associated with only one user at a time. This is a One-To-Many relationship from users table to devices table. Then, the timestamp entity will have a consumption and a timestamp, this pair of attributes illustrates how much energy that device consumed from the beginning to the time pointed out by the timestamp. The relationship between timestamp and devices has been done through a join table.



3. Functional requirements

To implement the function requirements for this assignment I will be using Spring REST API and my architecture will be Controller-Service-Repository one. For the front end, the only special thing I will be using will be a guard for the admin so that when a user tries to go to an admin page, he will not let be to.

4. Deployment on docker

My backend docker file looks like this:

```
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/assignment1-0.0.1-SNAPSHOT.jar list
RUN java -Djarmode=layertools -jar /root/target/assignment1-0.0.1-SNAPSHOT.jar extract
RUN ls -l /root

FROM openjdk:17.0.2-oracle

ENV TZ=UTC
ENV DB_IP=host.docker.internal
ENV DB_PORT=5433
ENV DB_USER=postgres
ENV DB_PASSWORD=12345
ENV DB_DBNAME=assignment1


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:MaxRAM
```

Starting from this, I ran the command: “docker build –t assignment1 .” in order to create the specific docker image for the backend and then “docker-compose up –d" to generate its container.

For the front end I have the following file:

```
FROM node:16 as builder
WORKDIR /app
COPY package*.json /app/
RUN npm install
COPY ./ /app/
RUN npm run build


FROM nginx:1.17-alpine
RUN apk --no-cache add curl
RUN curl -L https://github.com/a8m/envsubst/releases/download/v1.2.0/envsubst-`uname -s`-`uname -m` -o envsubst && \
    chmod +x envsubst && \
    mv envsubst /usr/local/bin


COPY --from=builder /app/nginx.conf /etc/nginx/nginx.template
CMD ["/bin/sh", "-c", "envsubst < /etc/nginx/nginx.template > /etc/nginx/conf.d/default.conf && nginx -g 'daemon off;'"]
COPY --from=builder /app/dist/frontend/ /usr/share/nginx/html
```

From this, I ran the same commands, but I renamed the image. These created the image and the container for the frontend.

I did the same for the database but I used the following docker file:

```yaml
# Use postgres/example user/password credentials
version: '3.1'

services:

  db:
    image: postgres
    restart: always
    environment:
      POSTGRES_PASSWORD: 12345
    ports:
      - 5433:5432

  adminer:
    image: adminer
    restart: always
    ports:
      - 9080:8080
```

This all creates the following docker architecture illustrated in the schema below: