Assignment 1: Distributed Systems Laboratory

Energy management system application

Dan Dumitru Ciobanu

2023

**Introduction**

The application that I have started developing is an Energy Management System that for the moment consists of three microservices that were created with the help of the Spring framework, one standing for the operations related to users, one for the devices and the last one for monitoring the energy consumption of the devices (plus an helper app, which can be instantiated multiple times, that reads values from a csv file to mimic the energy consumption for a device), as well as a React application for the frontend side. The purpose of the application is to reduce energy consumption mostly and to create a better experience for the users in their relationship with the energy provider company.

**Backend Components**

* **USER MICROSERVICE:** 
  + Responsible for managing the user related information and for providing roles to the users of the application, spitting them into: administrators and clients
  + The corresponding database contains only one table with the users with the following columns: id, firstname, lastname, password and role.
  + Having in consideration the scope of the project, there in no table assignated to the role management.
* **DEVICE MANAGEMENT:**
  + Responsible for managing the device related information and for creating a mapping between the users and the devices.
  + The synchronization mechanism for users in the two microservices was performed using an extra table for users with only one column, the id, such that when a request related the addition or deletion of users is made, the two microservices will communicate for keeping the user entries consistent and updated.
  + The device database contains two tables, as mentioned, one for users with just the id column and one for the devices having: id, description, max hourly consumption and a link to its owner.
  + There is a one to many relationship between the user and device tables
* **MONITORING:**
  + Responsible for storing information related to the energy consumption of each device and the devices.
  + The energy consumption related infotmation comes form the consumtion reader apps via RabbitMQ (the queue ‚testqueue’), while the devices are kept in sync with the ones from the device management microservice, again, via RabbitMQ (the queue ‚deviceUpdatesQueue’).
  + The monitoring database contains two tables, one for the devices and one for the devices consumption.
  + There is a one to many relationship between the device and the device consumption history tables.
* **SMART CONSUMPTION READER:**
  + Responsible for reading a .csv file which contains the values that would mimic the consumption.
  + Writes in the ‚testQueue’ the read values together with a timestamp and a device id (which comes from a config file).
  + Can be instantiated multiple times for simulating the reading for more devices.
* **COMMUNICATION:**
  + it was obtained by using the RestTemplate provided by the Spring framework by sending HTTP requests allowing the two microservices to be communicating with one another.
* **CHAT MICROSERVICE:**
  + Responsible for the chat communication of the user by using WebSockets.
  + There is a database behind all this communication, but the development in that regar was stopped, the focus being solely on the WebsSocket part.
  + There is a MessageController for each of the directives: message sending, typing and marking a message as being seen.
* All the microservices are secured with the help of JWT, the microservices sharing the same secret key.

**Frontend Components**

* The frontend component consists of four pages:
  + Sign in/up page
  + Client page: where users can see their devices and also see the consumption graph for a chosen day for a chosen device that they own (the users also get notifications via WebSocket when a device that they own has a greater hourly consumption that the maximum allowed value)
  + Admin page: where admins can perform CRUD operations on clients and devices
  + Chat page: where user can open conversations and chat with one another.
* The communication between the frontend and backend is based on a JWT that is created by the user microservice when a user signs in. This method was used for security reasons and I also integrated in it the role access to pages part (even though this is against the JWT main idea, which is to make the authorization part without needing to check the database or store anything on the server side).
* The chat part is created with WebSockets, using Stomp and SockJS.

**Conceptual Architecture**

The microservices architecture in this instance is divided into three distinct layers:

* Controller Layer: This layer contains controllers that expose endpoints. These controllers call methods from the business layer to fulfill requests and provide responses.
* Business Layer: This is where the majority of the application's logic is executed.
* Repository Layer and Database Layer: These layers manage data storage and retrieval.

A screenshot of a computer

Description automatically generated

CONCEPTUAL ARCHITECTURE DIAGRAM

**Deployment**

The deployment was made on Docker. There are 5 containers in this case, that communciate with each other inside Docker network that was created by default, not custom, in paranthesis the port mapping is stated (localhost:docker):

1. One for the frontend that runs a custom image (3000:3000)
2. One for the user microservice, running a custom image (8080:8080)
3. One for the device microservice, running a custom image (8081:8081)
4. One for the monitoring microservice, running a custom image (8082:8082)
5. One or more for the csv reader helper apps, running a custom image (8083-...:8083-...)
6. One for the chat microservice, running a custom image (8085:8085)
7. One for the user database, which runs the default mysql:8.0 image (3307:3306)
8. One for the device database, which runs the default mysql:8.0 image (3308:3306)
9. One for the monitoring database, which runs the default mysql:8.0 image (3309:3306).
10. One fot the chat database, which runs the default mysql:8.0 image (3310:3306).

A screenshot of a computer

Description automatically generated

DEPLOYMENT DIAGRAM