Energy Management System Application – Assignment 3

Student: Ratoi Razvan

Group: 30442

1. Contents

2. Conceptual Architecture of the Distributed System

This application is a distributed system with the main functionality of managing the energy consumption of different devices owned by various users. It's main functionality is of an administrator performing CRUD operations (**C**reate-**R**ead-**U**pdate-**D**elete) on devices and users and also create mappings between the latter 2 mentioned. A basic client can only see their associated devices. The additional functionality of this assignment was to add a monitoring method such that a client can see the consumption of their devices and also be notified in real time if a device consumes an amount of energy that is greater than the maximum one established.

It is composed of 4 main components: a front-end service and 23microservices that are used as the back end. Each microservice is connected to a database, corresponding to its functionality. The detailed description of each of this service is found below:

2.1 User Database is the service that contains the database for the users. It has only one table, that being Users, having the following fields (and their corresponding types):
- Id: UUID
- Name: Varchar
- Address: Varchar
- Role: Varchar (out of the 2 options: Admin/Client)
- Username: Varchar
- Password: Varchar
- Salt: Varchar

2.2 Device Database is the service that contains the database linked to the devices. It consists of 2 tables, those being (followed by their fields and types):
- Devices:
- Id: UUID
- Name: Varchar
- Address: Varchar

- Description: Varchar
- Max Hourly Consumption: Double Precision

- UserDevice:
  - Id: UUID
  - DeviceId: UUID
  - UserId: UUID

The latter table represents a many-to-many relationship between the devices and users, meaning that an user can own multiple devices and the same devices being owned by multiple users (e.g. An user can have both a vacuum cleaner and a smart fridge, but the last being owned by another user – living in the same house).

2.3 Monitoring Database is the service that contains the database for the monitoring microservice. It is composed of the following databases:

- Monitoring:
  - Id: UUID
  - Timestamp: Bigint
  - DeviceId: UUID
  - Consumption: Double Precision
- Limits:
  - Id: UUID
  - DeviceID: UUID
  - Limit: Double Precision

The first table represents the readings of energy consumption from the devices and the latter one represents the maximum consumption of energy allowed of each device.

2.4 User Microservice is the microservice serving to perform all needed operations on users, such as CRUD on users, authentication and synchronization with the device microservice such that deleting an user also handles deletion for mappings related to said user.

2.5 Device Microservice is the microservice that is responsible for handling the needed operations related to the devices and the mappings between them and users. Its main functionalities lie in the following: CRUD on devices and mappings and alerting the monitoring microservice such that a new entry for the "Limits" table is created for synchronization. (this is done by sending a message on a RabbitMQ queue)

2.6 Monitoring Microservice is the microservice entrusted with handling all the monitoring of each device's consumption. Besides being able to create, update, get and delete records of Limits and Monitorings, it also has a Consumer component that consumes messages from 2 RabbitMQ queue.

The web socket hub is found in the NotificatinoHub component, which implements the Hub given by Microsoft's SignalR library created in this sense.

2.7 Device Simulator is a Desktop Application which simulates the consumption of a device (whose ID is given) by reading from a .csv file and preparing the read data to be sent to the RabbitMQ queue.

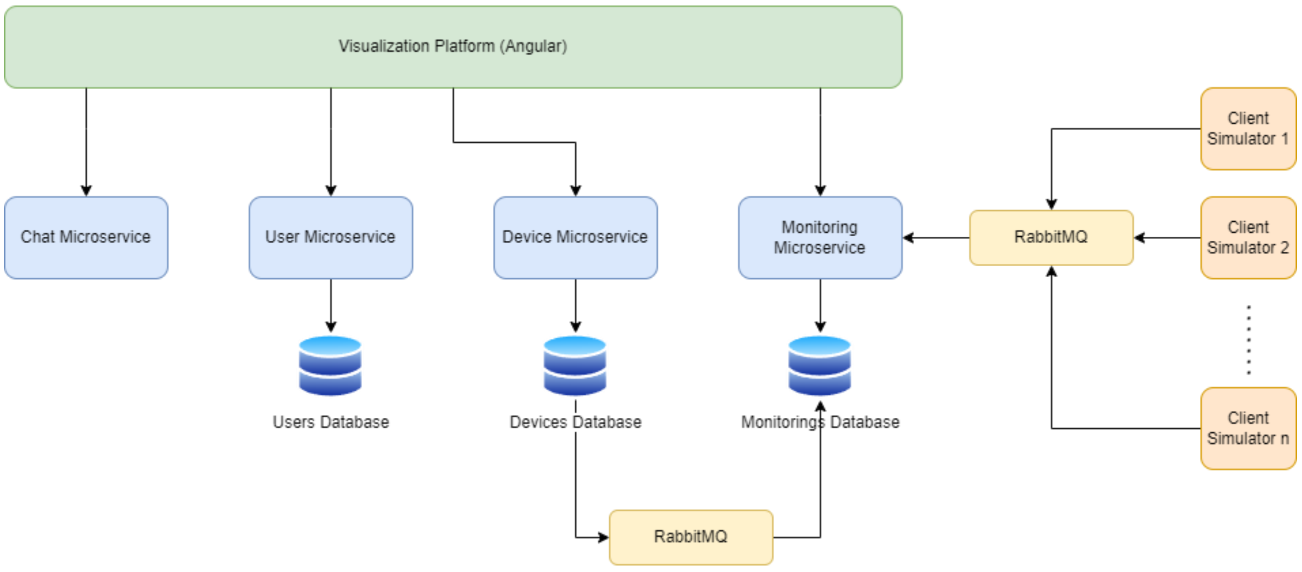
2.8 Chat Microservice is the microservice responsible for handling communication through a chat between clients and the administrator. Like the **Monitoring Microservice**, it makes use of the Microsoft's SignalR library for handling web socket communication. It contains of a **Message** class (which has the same fields as the interface defined in the client side: sender, receiver, content and timestamp). This class is used as a wrapper for the messages sent and received by web sockets, including the relevant information about that message. The controller-like class is the **ChatHub**, which has the necessary methods to support the chat:

- **SendMessage:** it receives a message and sends it back invoking the **ReceiveMessage** method of the client side.
- **SeenMessage:** similar to the previous one, but invokes **ReceiveSeenMessage**.
- **AlertTyping:** has the same structure as the first 2.

2.9 The Front End is the service that provides the interface for the user for easily accessing the data needed and to manipulate it in a friendly manner. The main pages that can be met by the user are:

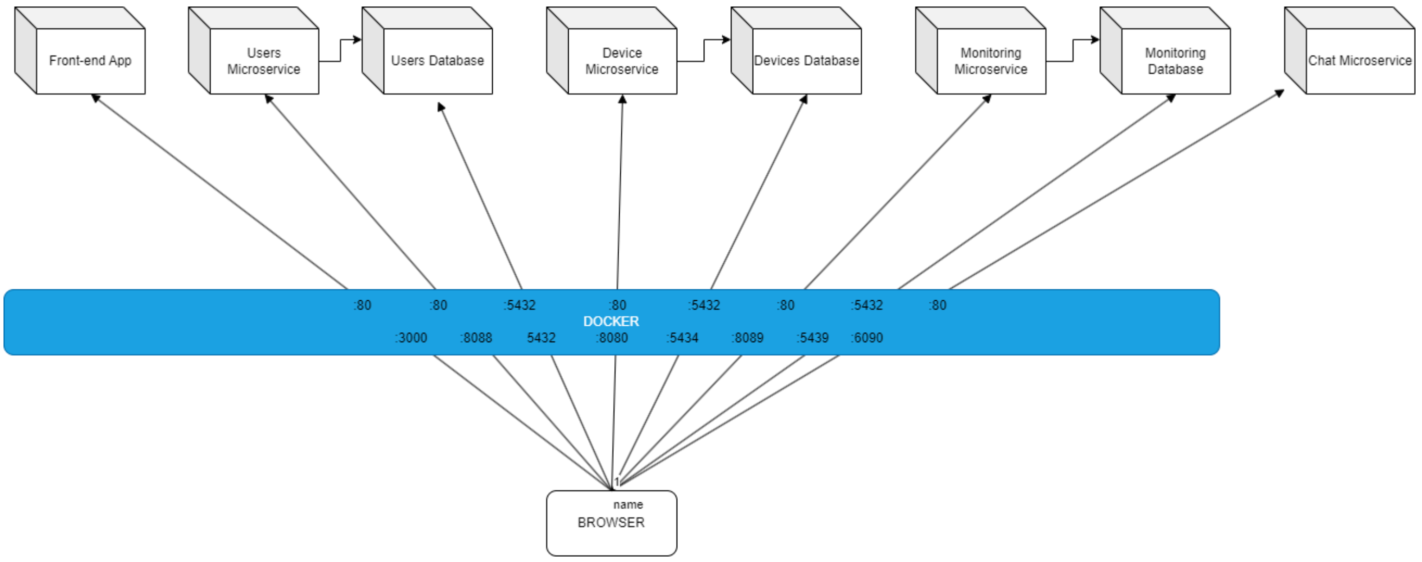
- Login Page;
- Dashboard Page + warning of energy consumption via a pop up;
- Access-Denied Page;
- Chart Page;
- Chat Page: this page contains 2 chat pages actually, but displays the correct one based on the role of the user:
    - The **Client Chat Page** renders a simple chat GUI, having the name of the other person in contact (the administrator) in the upper bar, a text input and a "send" button in the bottom bar and the rest of space is used to display messages. Messages received as seen by the admin are marked with 2 checks (✓✓) – the ones considered not yet seen are marked with just 1(✓). Also, besides the name of the other contact, a text can appear displaying "is typing".
    - The **Admin Chat Page** displays in the majority the same GUI as the client's, but additionally it contains a list of users that have written to them. The admin can select any client from the list and their chat will be displayed. Opening a chat is considered as seeing the new messages.
    - The chats are stored only session-wise (in the session storage), meaning that on exit from the browser, the chats will be deleted.
    - The frontend sends message via web sockets when 3 of the following events occur:
        - The user types something in the input (invokes **AlertTyping**)
        - The user opens/clicks on the conversation (invokes **SeenMessage**)
        - The user sends a message (invokes **SendMessage**)
    - The frontend also reacts to received web socket messages:
        - When receiving **ReceiveAlertTyping**, the status of the other person in contact is set to "is typing"
        - When receiving **ReceiveSeenMessage**, the messages are labeled by double checks.
        - When receiving **ReceiveMessage**, the new message is stored into the session storage and also displayed.

The overall architecture of the system can be seen in the figure below:



3. <u>UML Deployment Diagram</u>

The deployment diagram of this application in shown in the figure below, alongside the associated ports for both the host and docker:



4. <u>Building and Execution of the project</u>

To get the project working, the following steps should be done:

4.1 Create a folder where you want to keep the app, and inside this one, 2 more folders: ds-front and ds-back-users.

4.2 Open a terminal and insert the following commands:

- ng new <project-name>
- ng add @angular/material
    - these 2 are needed for the front-end project (make sure you are in the ds-front directory)
- dotnet new webapi <UsersMicroservice>
- dotnet new webapi <DeviceMicroservice>
- dotnet new webapi <MonitoringMicroserviceNET>
- dotnet new webapi <ChatMicroservice>
    - to create the 4 .NET projects (in the ds-back-users directories)
- git clone https://gitlab.com/ds2023-ratoi-razvan/ds2023_30442_ratoi_razvan_assignment_3.git
    - this is to pull the source files from the git repository

4.3 Replace the src folder in the Angular project with the one pulled from the repository.

4.4 Add to each of the .NET projects the corresponding directories

4.5 Open a terminal in the folder you firstly created

4.6 Run the command: docker-compose up –build (if no changes were done to the code, forget the –build)

4.7 Access localhost:3000 in your browser and you will be shown the Login Page

4.8 Log in using one of the following credentials:

- Administrator:
    - Username: admin
    - Password: admin

- Client:
    - Username: client
    - Password: client

4.9 Enjoy the application!