**Energy Utility Platform**

**Assignment 2**

**Student: Lupsa Sergiu-Rares**

**Group: 30641**

**Contents**

1. Requirements analysis ...........................................................Eroare! Marcaj în document nedefinit.

1.1 Assignment specification............................................................................................................. 3

1.2 Functional requirements............................................................................................................. 3

1.3 Non-functional requirements ..................................................................................................... 3

2.Use-case model........................................................................................................................... 4

3.System architectural design........................................................................................................ 4

3.1 Architectural pattern description and diagrams...................................................................... 4 Component diagram...............................................................

Deployment diagram..................................................................................................................... 6

4.Class design................................................................................................................................. 7

4.1 UML class diagram ................................................................................................................... 7

4.2Data model ............................................................................................................................... 8

5.Bibliography................................................................................................................................ 9

**1. Analysis of requirements**

**1.1 Assignment specification**

The second assignment implement a component for Assignment 1 application based on a message broker middleware that gathers data from the smart metering devices, pre-processes the data to compute the hourly energy consumption and stores it in the database. A Smart Metering Device Simulator module will be the Message Producer. It will simulate a sensor by reading energy data from a file (sensor.csv - one value at every 10 minutes) and sends data in the form < timestamp, device\_id, measurement\_value > to the Message Broker (i.e., the queue). The timestamp is taken from the local clock, the measurement\_value is read from the file and represents the energy measured in kWh, and the device\_id is unique to each instance of the Smart Metering Device Simulator and corresponds to the device\_id of a user from the database (as defined in Assignment 1). The sensor simulator is developed as a standalone application ( desktop application). The measurements are sent to the queue using the following JSON format:

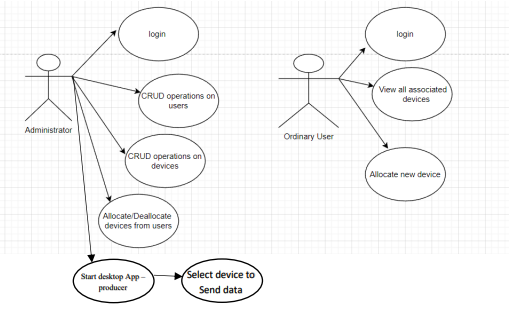
{ “timestamp": 1570654800000, “device\_id”: “5c2494a3-1140-4c7a-991a-a1a2561c6bc2”, “measurement\_value”: 0.1 }

A Message Consumer application will pre-process the data to compute the total hourly energy consumption and shows it on the console of the docker.

**1.2 Functional requirements**

The message broker allows Smart Metering Device Simulator to act as messages producer and send data tuples in a JSON format. • The message consumer component of the system processes each message from the queue and displays it on docker’s console.

**1.3 Non-functional requirements**



- Usability: user-friendly interface for the application - Security: minimal security assured by authentication and restrictions regarding access to different pages

**2. Use-case model**

Use case: Administrator login and see all devices and, by choice, all users, start desktop application by selecting 2 devices in order to send data to queue and then to consumer Primary actor: Administrator Main success scenario: Login -> See all device -> Start desktop application -> Select 2 devices to send data from -> Watch the data in the docker’s console Extensions: There are no devices yet, there are no users yet

**3. System architectural design**

**3.1 Architectural pattern description and diagrams**

Layered architectures – the most common in software development. It describes an architectural pattern which is composed by separate horizontal layers that function together as a single unit of software Start desktop App – producer Select device to Send data 5 This diagram contains all the components of the application. The main functionalities are based on 3 main components: the database (Postgres), REST API server and Angular Frontend for client visualization. The client makes a requirement which is transferred to the Server, and after that, a query is sent to the database. Then the results will be shown to the client.

Diagram

Description automatically generated

**Component diagram** Diagram

Description automatically generated

Diagram

Description automatically generated

This diagram shows the presence of three Docker containers.

These 3 containers are: the container for the database (Postgres container), the container for the server (Backend container) and the container for the client application (Frontend container).

Each container has an image built on the basis of a Dockerfile, an image that can be run locally from Docker containers and not from an IDE. For this application, it was deployed locally, running from Docker containers. The component that encapsulates the server is dependent on the database, and the client side is dependent on the database as well as the server.

The communication of the 3 components is done through a network that creates a communication bridge.

**4. Class design**

**4.1 UML class diagram**

A screenshot of a computer

Description automatically generated with medium confidence

**4.2 Database model diagram A picture containing diagram

Description automatically generated**

**5. Implementation**

The backend application was made in Spring using the Java language. The application was structured using the following packages: entities (for the application models), dtos (to transmit the first data from the frontend), repositories (for data persistence), services (for the services in which the application logic is), controllers (classes in which the endpoints of the application and the calls to the corresponding services are declared - RESTful API) and security (in which the security of the application is ensured). 9 The frontend application is written in Typescript, using the Angular framework. The structure of the application is organized by packages, so that it is as readable and easy to understand as possible. The main packages include components such as: Login component, Device-table component, User-table component etc. An ordinary user (client) has access only to his page (User-page component), and an administrator has access to all components. This is ensured by an Authorization Guard (does not allow access to pages if the user is not logged in) and a Role Guard (does not allow access to pages for which permission is not granted, depending on the role). Data saving is ensured by the PostgresSQL application, where the database is located. The database is also secured, requiring a password and a username to access the databases

**6. Bibliography**

https://dsrl.eu/courses/sd/ https://docs.docker.com/desktop/install/windows-install// https://material.angular.io/components/table/overview https://www.tutorialspoint.com/rabbitmq/rabbitmq\_producer\_application.htm https://www.tutorialspoint.com/rabbitmq/rabbitmq\_consumer\_application.htm https://www.rabbitmq.com/tutorials/tutorialonejava.html?fbclid=IwAR0xGx4MoUilDVUknAfB2EzuBmYkvtScGF8ZQwciqQPwuf9dP3fJeehrMC