ENERGY UTILITY PLATFORM

**& SENSOR SIMULATOR**

Analysis and Design Document

Student: Florin Petrean

Table of Contents

1. Requirements Analysis 3

1.1 Assignment Specification 3

1.2 Functional Requirements 3

1.3 Non-functional Requirements 3

2. Use-Case Model 3

3. System Architectural Design 3

4. UML Sequence Diagrams 3

5. Class Design 3

6. Data Model 3

7. System Testing 3

8. Bibliography 3

1. Requirements Analysis

# Assignment Specification

The clients of the energy distributor have installed smart meters for each device registered to measure its energy consumption. Each sensor sends data to a server periodically, in the form *(timestamp, sensor\_id, measurement\_value)*, where *timestamp* is the time instance when the measurement was made and *measurement\_value* is the value of the energy counter measuring the total energy consumed by the device in kWh since the sensor was installed.

Implement a system based on a message broker middleware that gathers data from the sensors and pre-processes them before storing them in the database. If the queue consumer application that preprocesses the data detects a measurement power peak that exceeds the sensor maximum threshold (i.e. sensor *maximum value* measure in kW defined in Assignment 1) it notifies asynchronously the client on its web interface. To compute a power peak, the instantaneous power in a measurement interval is computed by averaging the energy consumption and dividing the value to the time interval.

𝑃𝑝𝑒𝑎𝑘(𝑡1,𝑡2)=(𝑚𝑒𝑎𝑠𝑢𝑟𝑒𝑚𝑒𝑛𝑡𝑣𝑎𝑙𝑢𝑒(𝑡2)−𝑚𝑒𝑎𝑠𝑢𝑟𝑒𝑚𝑒𝑛𝑡𝑣𝑎𝑙𝑢𝑒(𝑡1))/(𝑡2−𝑡1)<𝑀𝐴𝑋𝑣𝑎𝑙𝑢𝑒

**A Sensor Simulator** will simulate a sensor that reads data from files (sensor.csv), one value at every 10 minutes. The module will contain a timer synchronized with the local clock. The module sends data in the form < *timestamp, sensor\_id, measurement\_value* > to the message broker. The timestamp is taken from the local timer, the *measurement\_value* is read from the file at the corresponding index, representing the energy measured in kWh, and the *sensor\_id* is unique to each instance of the Sensor Simulator and corresponds to the sensor ID associated to a device of a client from the Energy Database.

The sensor simulator should be developed as a standalone application (i.e. desktop application) to read the sensor monitored activities from the file *sensor.csv,* configured as a message producer and send the monitored sample data to the queue defined. The file *sensor.csv* can be downloaded from https://dsrl.eu/courses/sd/materials/sensor.csv. The measurements are sent to the queue using the following JSON format:

{

“timestamp”: 1570654800000,

“sensor\_id”: “5c2494a3-1140-4c7a-991a-a1a2561c6bc2”

“measurement\_value” : 0.1,

}

# Functional Requirements

* The message-oriented middleware allows the sensor system to send data tuples in a JSON format
* The message consumer component of the system processes each message and notifies asynchronously using WebSockets the client application

# Non-functional Requirements

The sensor simulator runs using the Scheduler from SpringBoot and can be configured from within the application.properties (using arguments on the .jar file) to change the sensor id and the rate at which the messages are sent to RabbitMQ queue. Also, a different server port has to be specified for different sensor simulators instances, so there is no conflict.

The notifications in frontend will only appear to the user to which the device that has the sensor installed belongs to. We configured the websockets on the backend in order to achieve this functionality.

The RabbitMQ Listener has been implemented in RecordService backend, as it fits the description of a service that processes data received through the queue.

2. System Architectural Design

**2.1 Architectural Pattern Description**

The application is using a layered architecture on the server and each layer of the layered architecture pattern has a specific role and responsibility within the application as suggested by the name of the layer.

Between the presentation and business layer we have controllers which puts the needed data from business layer onto a specific port in order for the presentation layer to access it.

Diagram

Description automatically generated with medium confidence

The presentation layer is separated “physically” by the other layers and deployed as a separate application in our case. The frontend part of the application was developed in React.

The other layers represent the backend part of the application which is also deployed and developed with Spring Boot. The database layer resides in Heroku.

Both parts of the application are deployed using Heroku Servers as docker containers.

The sensor simulator architecture is a simple one, being a spring boot application that simply parses data from the sensor.csv file and schedules the sending of data at fixed intervals of time. We do that through the custom classes RabbitMQ config and RabbitMQSender in order to connect to our CloudAMQP server. All information about the connection to this service resides in application.properties. This applies to the backend part of the project as well.

**2.2 Diagrams**

**Deployment Diagram**

*Diagram

Description automatically generated*

3. Class Design

**For Backend**

User

* Long id
* String username
* String firstName
* String lastName
* String address
* LocalDate dateOfBirth
* List<Device> devices

Device

* Long id
* String description
* String address
* Long maxEnergyConsumption
* Float avgEnergyConsumption
* Sensor sensor

Sensor

* Long id
* String description
* Double maxValue
* Device device
* List<Record> records

Record

* Long id
* Long timestamp
* Double energyConsumption

**For Sensor Simulator**

Record

* Long id
* Long timestamp
* Double energyConsumption
* Long sensorId

4. Data Model

The data model of this application contains 4 tables: user, device, sensor, record.

*Diagram

Description automatically generated*

8. Bibliography

<https://www.springboottutorial.com/spring-boot-react-full-stack-crud-maven-application?fbclid=IwAR1_EEVsMkGDDM1qEVtTXEC2puVLmiWAatgQzOl1kDD6OiZeIUfLvvaydus>

<https://hellokoding.com/deleting-data-with-jpa-hibernate/>

<https://github.com/RatneshChauhan/springboot-react-chatroom>

<https://www.youtube.com/watch?v=o4qCdBR4gUM>

<https://github.com/zcmgyu/websocket-spring-react>

<https://www.tutorialspoint.com/spring_boot/spring_boot_scheduling.htm>

A lot of stackoverflow

Course materials