DISTRIBUTED SYSTEMS

Assignment 2

Asynchronous Communication

Sensor Monitoring System and

Real-Time Notification

Andrei Dorofte

Group 30641

1. **Conceptual architecture of the distributed system**

**Sensor Simulator**

For this assignment, it was needed to create a new desktop app, that simulates the transmission of the power measurement from the smart meters of the devices of a user.

The input data is stored in the sensor.csv file, which contains a large number of values read by a sensor.

The sensor simulator application, reads the data from the csv file, whose name is passed as a command line argument, line by line and sends them to the RabbitMQ Service, to the desired queue, defined by the developer, along with the ID of the sensor, which is parsed as a command line argument too. The date is also passed as a string, incremented by 10 minutes at every value sent, using a local variable.

Initially, the RabbitMQ Service was hosted locally, on my computer. I installed it using the Chocolatey Package Manager. After I tested that my solution works fine, I created an account on CloudAMQP, in order to host the service there. Then, I copied the URI that they provided for my newly hosted service, and modified the connection in the Sensor Simulator application, as well as in the backend application.

Graphical user interface, text, application

Description automatically generated

**Backend**

In order to receive the messages from my Producer application, I had to integrate the RabbitMQ Consumer into my backend application. Fortunately, Spring Boot offers support for AMQP services, so I had to include the „spring-boot-starter-amqp” artifact to my dependency list in pom.xml.



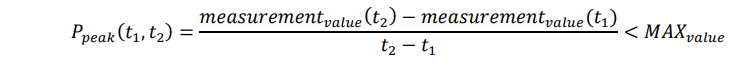
In order to connect to my CloudAMQP Hosted Service, I had to define an environment variable in my application.properties, where I pasted the connection string. Also, I defined another environment variable for the name of the queue from which I listen in my RabbitMQService class.

Text

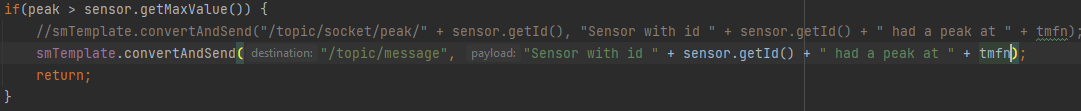
Description automatically generated

This component listens to the queue defined in the application.properties variable, and calls the method in my measurement service. There, I parse the received string to sensor ID and Value, I search for the sensor in the database, and create a new Measurement. The time passing simulation is done in the Sensor Simulator application, and the date is just parsed from the received string and assigned to the newly created measurement.

In order to calculate if the sensor max value is exceeded, I search for the last entry in the measurement table, for the desired sensor, and compute:



If the peak is not exceed, then I store the new measurement in the database. If it is exceeded, then I notify the frontend through a websocket and call „return” in order to exit the function.



**Frontend**

On the frontend application, I created the Client page, in which he can see his own devices, along with their sensors and their measurements.

Graphical user interface, application

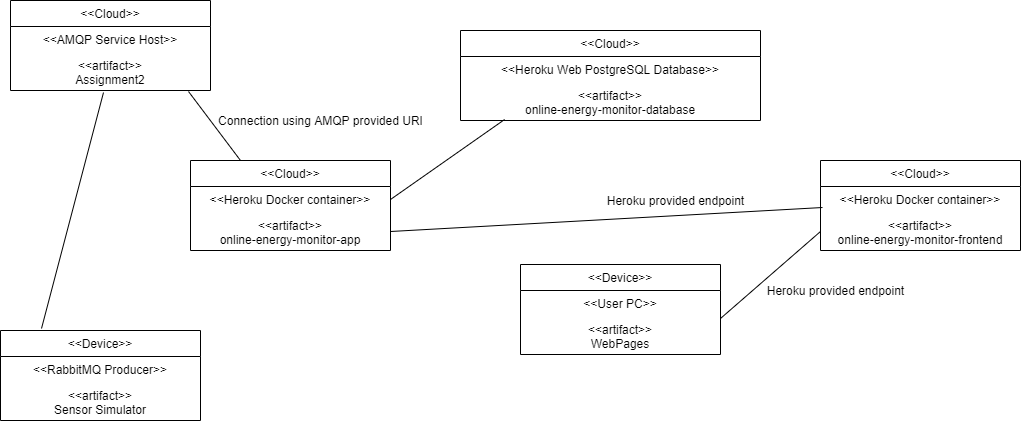
Description automatically generated

Also, here is the place where the client is notified if a sensor had a peak value, through an alert, where the id and the date are shown.

Text

Description automatically generated

1. **UML Deployment diagram**



Here is the new deployment diagram. The user accesses the link provided by Heroku, from the deployment of the frontend application. The deployed frontend app then calls the endpoints of the deployed backend application, over the Heroku provided endpoint. Then, the Heroku database of the backend container is used to store all the objects used.

The deployment is donw using Heroku’s docker containers, in which the deployment was done using the GitLab Pipeline workers. I had to create a Dockerfile for both the frontend and the backend, and also add some files needed for configuring the Nginx server (nginx.conf), along with some variables in each GitLab repository.

The backend application still uses the Heroku PostgreSQL database, whose credentials are stored in the backend’s Dockerfile.