Air quality sensor

Henrik Lechte, Florian Finkel, Julia Grabinski, Cara Damm
University of Mannheim
Pervasive Computing: Smart City Air Quality Sensors - FSS 2019
Master of Businessinformatics
Email: abc@xyz.com

***Abstract*—**

I. Technical implementation

The basic idea is, to build an air quality sensor out of an Arduino and multiple other sensors, distribute multiple of those air quality sensors throughout Perheim, gather the measured data into a central database and display this data into an easy to use and visually appealing application. The following chapters will cover each of these steps in detail.

A. Building the air quality sensor

First of all, it has to be determined what the final air quality sensor should measure. The European Union defined the European Air Quality Index (EAQI) in 2017 and it is similar to most of Air Quality Index from other governments around the world, like the U.S. Environmental Protection Agency. It provides a set of air quality index levels: good, fair, moderate, poor and very poor. Furthermore, it defines multiple components, such as Ozone, particulate matter, Sulfur dioxide, Carbon monoxide and Nitrous oxide, that are getting measured and it provides a table to calculate the EAQI with the measured data of these components. Since the EAQI was created by the EU it is used throughout Europe and will also be used as a basis for the air quality sensor in Perheim(Germany). As the goal is to distribute a lot of air quality sensors in Perheim, the first generation of air quality sensors will be built with cheaper Sensors to make them more affordable. In a later stage, when the developed system got accepted and gets used frequently, sensors can be more expensive and of higher quality standards. With the focus on cheaper products, the following sensors will be used for the first-generation air quality sensors:

- Ozone: SainSmart MQ131
- Particulate matter: Grove Dust Sensor
- Sulfur dioxide: Mq2 Gas Sensor
- Carbon monoxide: Mq9 Gas Sensor
- Nitrous oxide: MICS-2714

With these sensors all the components of the EAQI can be measured. Other important factors that play a crucial role to the severeness of the EAQI are the current temperature and humidity. That's why a temperature and humidity sensor will also be integrated into the air quality sensor to improve the calculations. In the first iteration the SODIAL - Temperature and Relative Humidity Sensor will be used. Another point is, that the final application has to use the location of each sensor to display the measured data at the corresponding site. Therefore, each air quality sensor will be equipped with a NEO-6M GPS sensor to send the Gps coordinates of its location with each measured data set. At last the internet module FONA Mini Cell GSM Breakout SMA will be utilized to connect the Arduino to the internet and send the measured sensor data to a central system for further processing. All these hardware components will be soldered to a soldering board and thereby connected to an Arduino. The final construct will be placed in a weatherproof box that has three cutouts. A 5-Volt fan will be placed in the first cutout to blow the air through the box, along all the sensors and finally out of the second cutout. The third cutout is for a power supply for the Arduino. With that procedure many air quality sensors will be manufactured. The estimated cost of one sensor without the box is around 150. The needed Arduino coding to read all the measurements is not part of this document.

B. Creating a backend to process incoming data and manage air quality sensors

The backend system that provides an application programming interface (API) for the sent measurements of the sensors described previously will be covered in this section. This API can be implemented in many ways, e.g. as a Representational State Transfer (REST) API or as a create, read, update, and delete (CRUD) API, but the detailed implementation of this API won't be discussed in this document. The API from the backend to receive the sent measurements of the sensors will provide the following functionalities:

- Receive a set of data containing the measurements, a timestamp and location from a sensor in JavaScript Object Notation (JSON) format
- Save received data into a persistence as one tuple of the corresponding database table
- Error handling in the event of wrong message formats
- Propagate crucial information if sensor measurements could be hazardous to health
- Authorization and authentication checks for each request

Moreover, the backend will also provide another API that sensors can poll from to retrieve the current interval time for sensor measurements. In case of numerous distributed sensors this API will make it easy to adjust said interval and avoid unnecessary manual labour. It will provide the features:

- Accept GET requests and return the currently set measurement interval

- Accept PUT and POST request to set a new measurement interval
- Error handling in case of wrong request format
- Authorization and authentication checks for each request

With the two APIs defined before all the required interactions between each air quality sensor and the backend can be achieved.

Since these two backend endpoints should provide authorization and authentication checks, a corresponding security system has to be implemented. For the first generation sensors the widely established open standard OAuth 2.0 will be utilized. Therefore, the backend system will also provide an authorization server that returns access tokens for the sensors. When a sensor is build the oauth client credentials flow has to be performed manually to retrieve an access token for the new sensor. After receiving the token it will be hardcoded into the sensor which will in return send this token in each of its measurement requests as authorization header. Figure 1 illustrates this concept in a simplified representation.

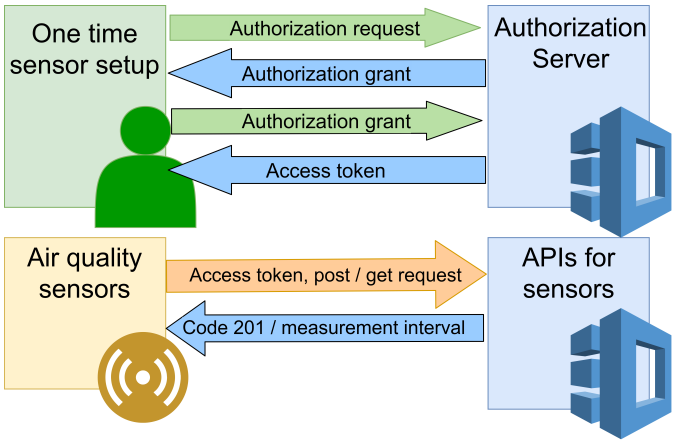

Fig. 1. Air quality sensor oauth credentials flow

Usually oauth access tokens have an expiration date but for the first generation sensors each token can be used for an unlimited time period. In a later implementation phase the security concept should be reworked since the manual set up is tedious and using an access token for a prolonged time may result in security vulnerabilities.

C. *Creating the frontend application*

A map overview of Perheim will be provided:

Detailed measurements of each sensor will be provided:

Notifications will be provided:

II. Conclusion

The conclusion goes here [**?**].

Acknowledgment

The authors would like to thank...

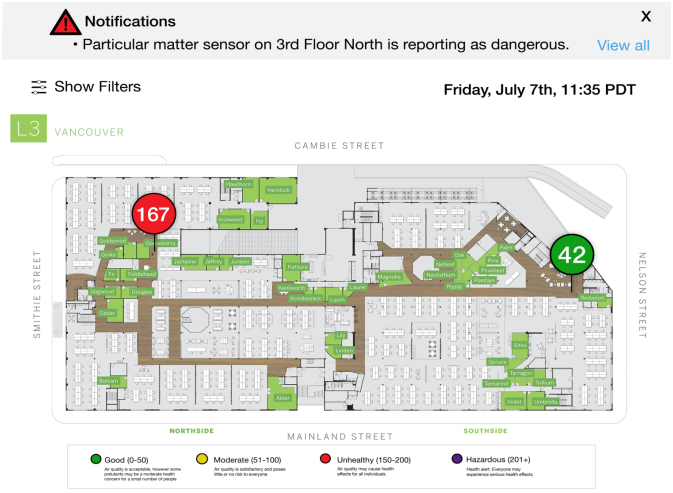

Fig. 2. perheimMapMockup

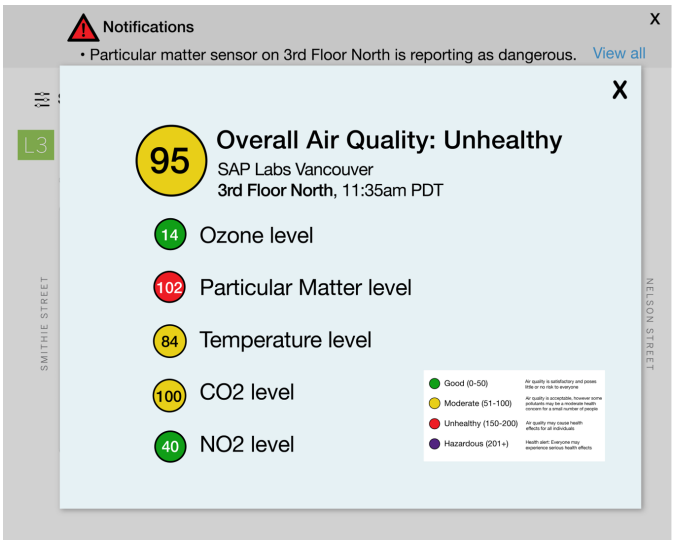

Fig. 3. DetailedSensorMeasurementsMockup

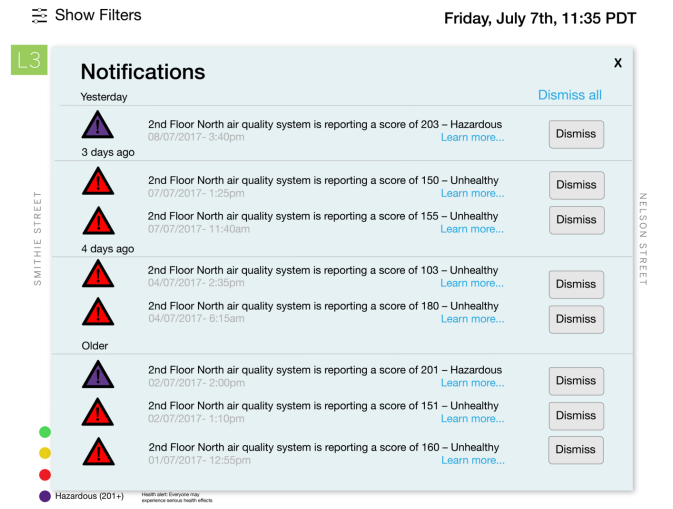

Fig. 4. NotificationsMockup