Gruppe 13: Julius Ostermann, Max Hiort

Fog Computing (Summer Term 2024) - Prototyping Assignment

Concept

For the virtual sensors of the client we have chosen two basic sensors where one collects the temperature and the other one collects the humidity. For simplicity purposes, we created a function which simulates realistic but random values every 1 to 3 seconds using AI. The client sends the sensors' data to a cloud server every 15 seconds. The cloud node receives the data, calculates the averages of the last 10 events from each sensor, and returns them to the client. The main idea is to have multiple sensors in different places in a city for example and to calculate the average temperature and humidity of the town. The concept could be expanded to use more sensors such as air quality.

Components

The prototype consists of a local client and a cloud server hosted on Google Cloud. Both the client and the cloud server are implemented using Python as the coding language and have a user interface, being a pyplot for the client (Figure 1) and a website for the server (Figure 2). The server uses FatsAPI to handle HTTP requests as well as WebSocket connections for the ui. The client uses the client generated from the server’s OpenAPI YAML file for HTTP communication. Both the client and the server use SQLAlchemy for the database interactions to store the sensor data and averages calculated. The data structure of the databases can be seen in Figure 3. Both the server and the client can either run inside a docker container or natively on the system, depending on the use case.

Communication

As communication in fog environments is not reliable, we developed a protocol which is resistant against the loss of messages. In order to prevent wasting resources we refrain from a consistent connection and use periodic requests and responses initiated by the client. This also ensures that clients can join and leave flexibly. The clients store all events locally in a database with an UUID and an untransmitted flag. On the periodical sync, all untransmitted events are collected from the database and sent to the server. The server responds with all event UUIDs it received which the client then marks as transmitted in its database. If the call gets lost (either request or response) the events will be retransmitted on the next sync until they get acknowledged by the server. By using UUIDs, the server can prevent inserting the same event twice. When the server receives the events it automatically calculates the averages for each sensor as described above which must not only contain values transmitted in this request. The results get assigned a UUID each and get stored in the server database with an untransmitted flag. The responses to the sensor data then contain not only the acknowledgements for the events received but also the calculated averages from the server. After the client acknowledges the reception of the average UUIDs, they are marked as transmitted in the server's database, ensuring robustness in both directions. The communication process is illustrated in Figure 4.

Reliability

To ensure reliability in case the client or the server crashes all the data is stored in a local database. This way no data is lost when a crash occurs and both components can operate independently. They will synchronize whenever both are available and connected.

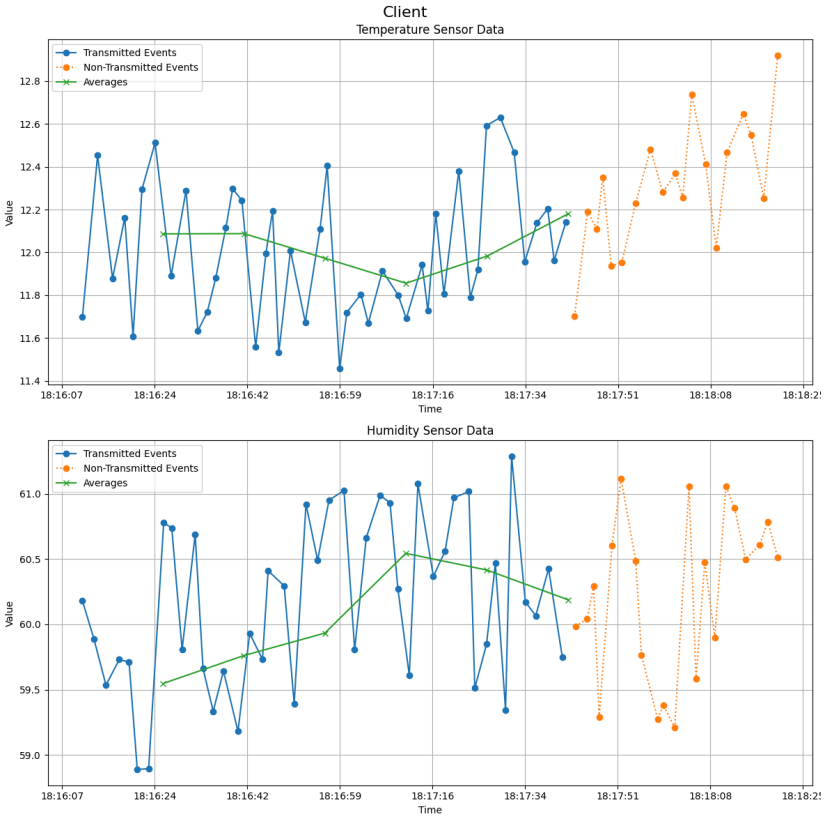

Figure 1 - UI of the client

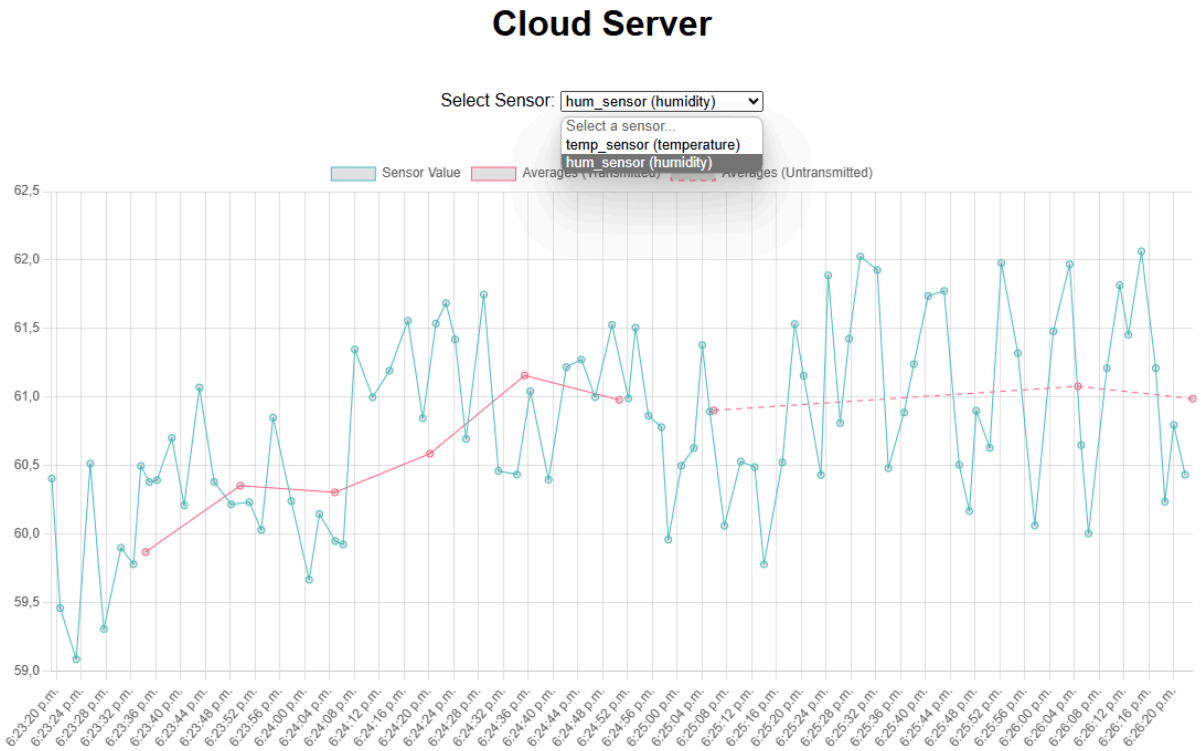

Figure 2 - Webinterface of the server

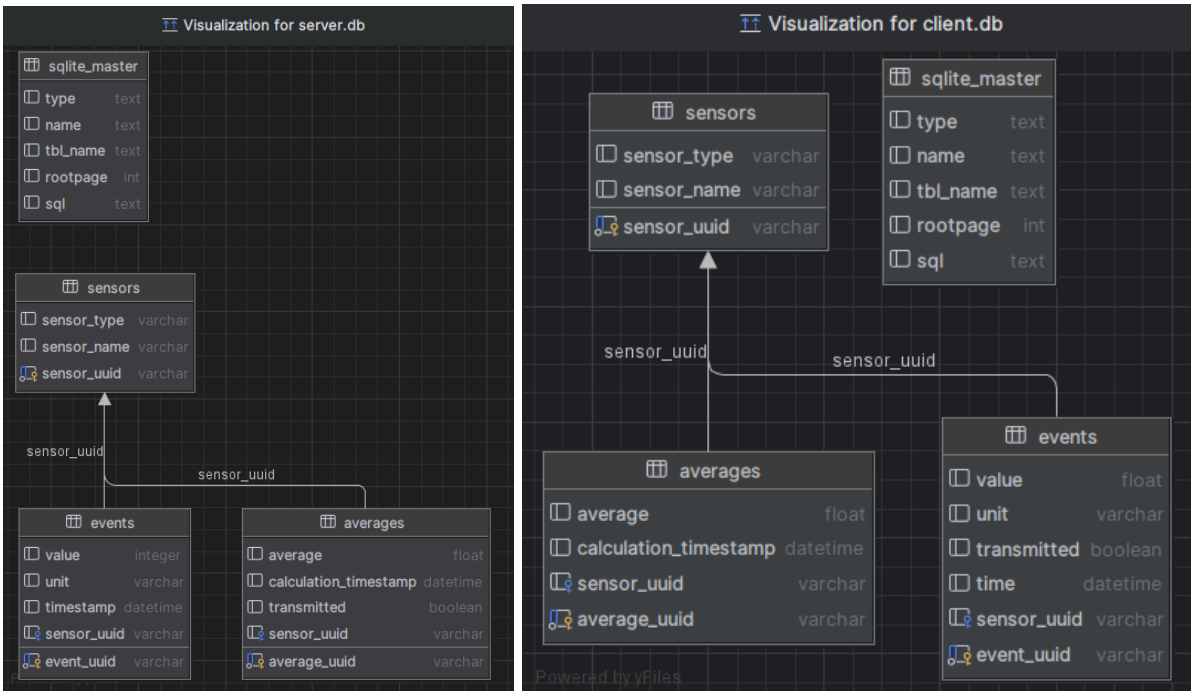

Figure 3 - Database structure

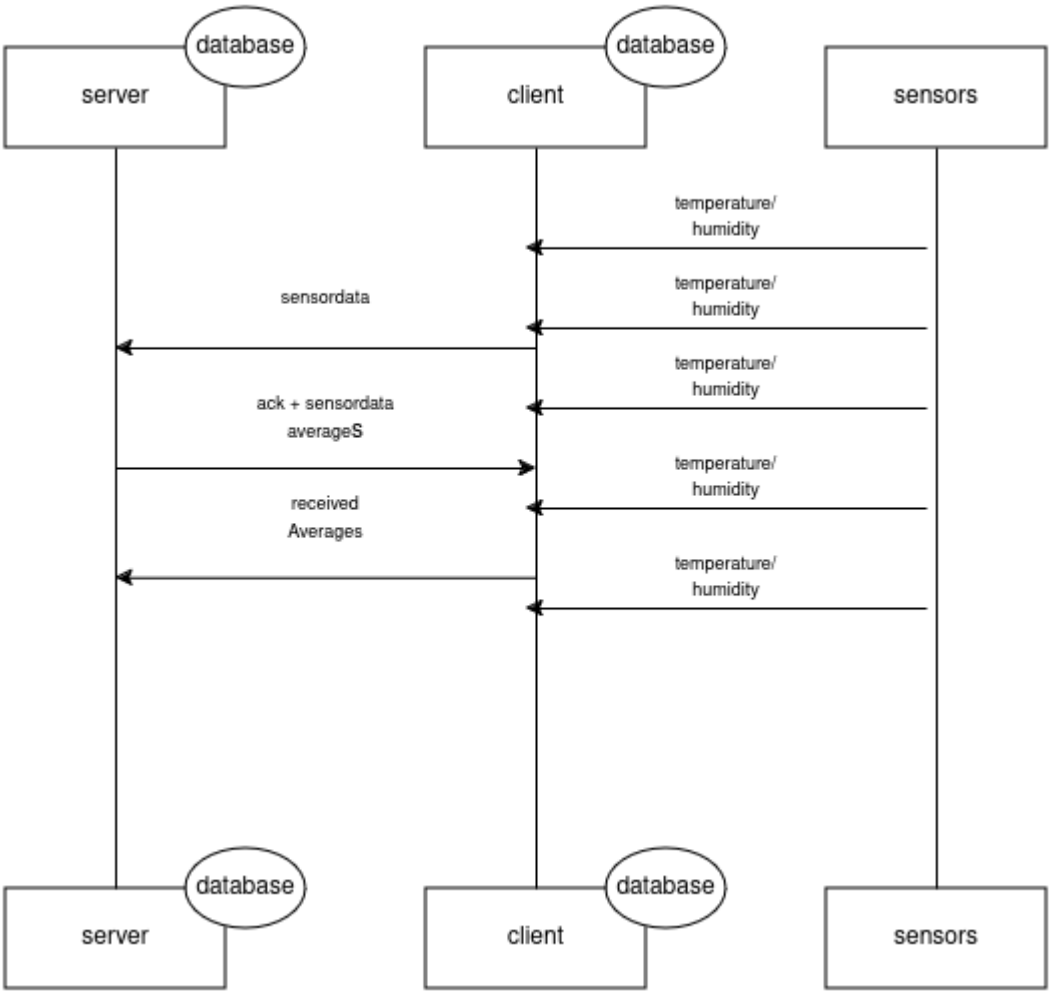

Figure 4 - Communication process