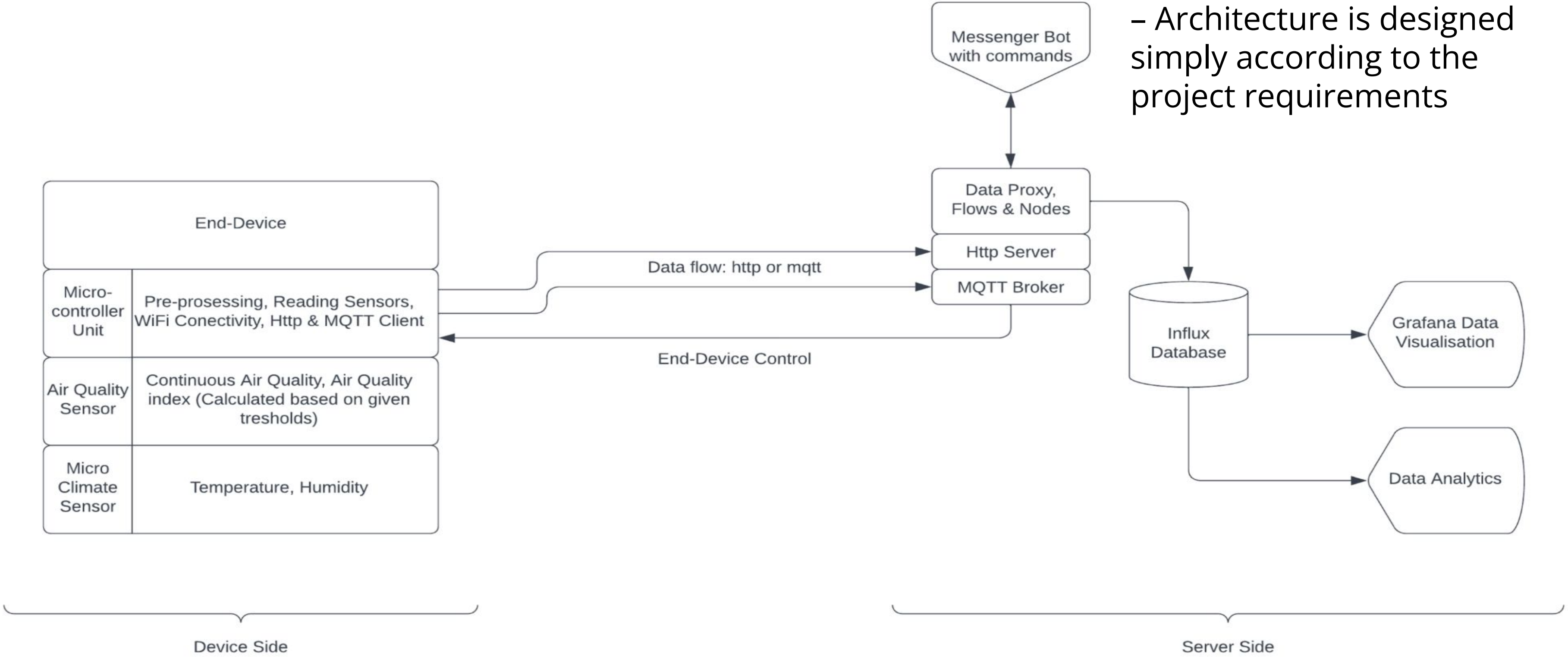
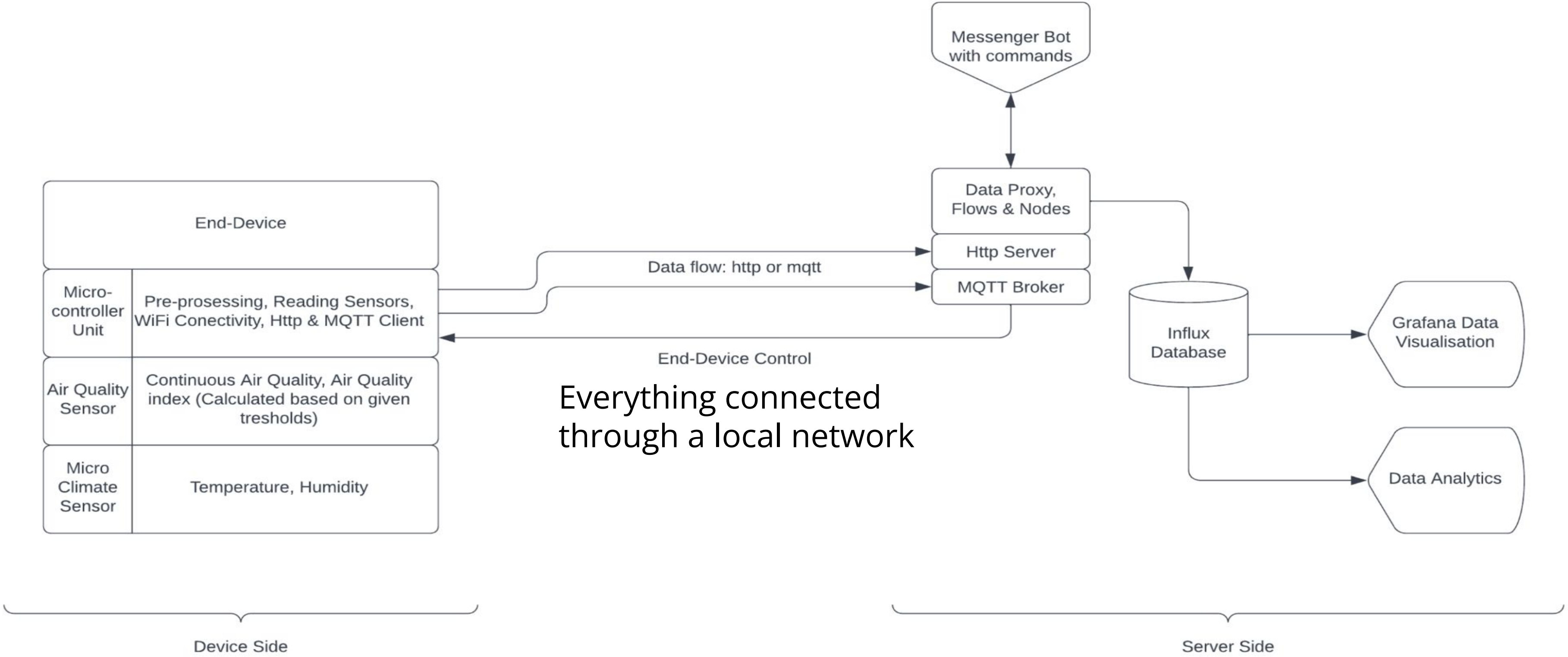
1. Project Design and Architecture

– Architecture is designed simply according to the project requirements



2. Project Implementation



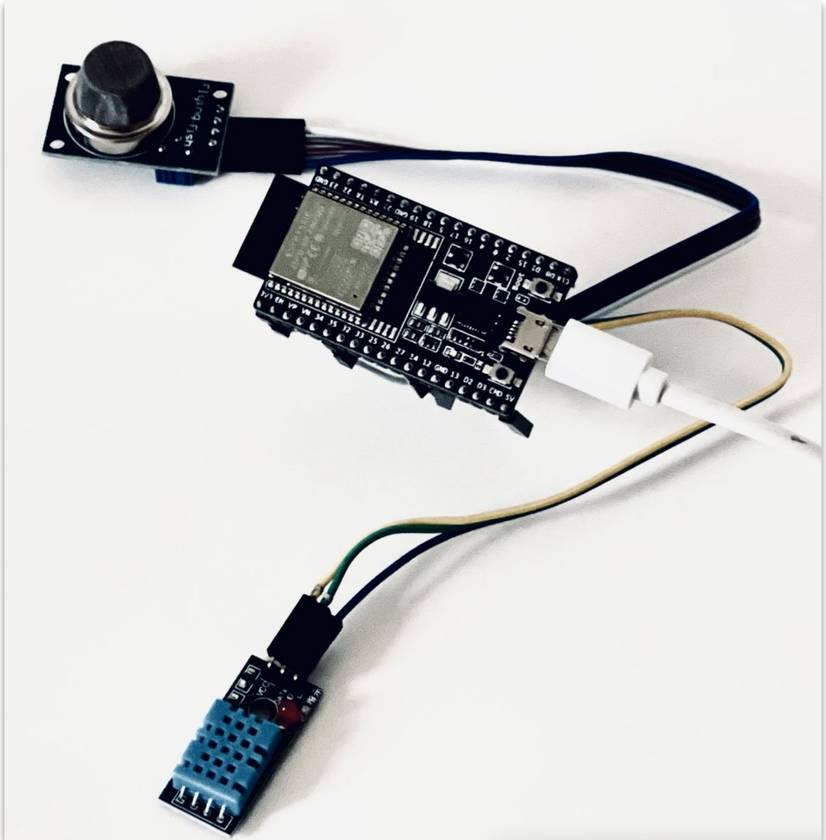
Everything connected through a local network

ESP-32 + Sensors

Effectively a laptop

2.1 End Device

– ESP-32 with WiFi antenna
– Analog: MQ2, One-Wire: DHT11
– HTTP & MQTT clients running on ESP
– Programmed using Arduino IDE & C++ with Arduino extension and using open source libraries.
– For MQTT client the values are sent directly to their topics, while for http-client the collected data is processed into a JSON.



2.2 Communication

– Indoor deployment → Local network

– MQTT broker (by MOSQUITTO) running on server (laptop) + HTTP Server

– Both MQTT and HTTP clients are constantly running on the ESP32.

2.3 Data Proxy

– Data proxy implemented with Node.js using **Node-Red**

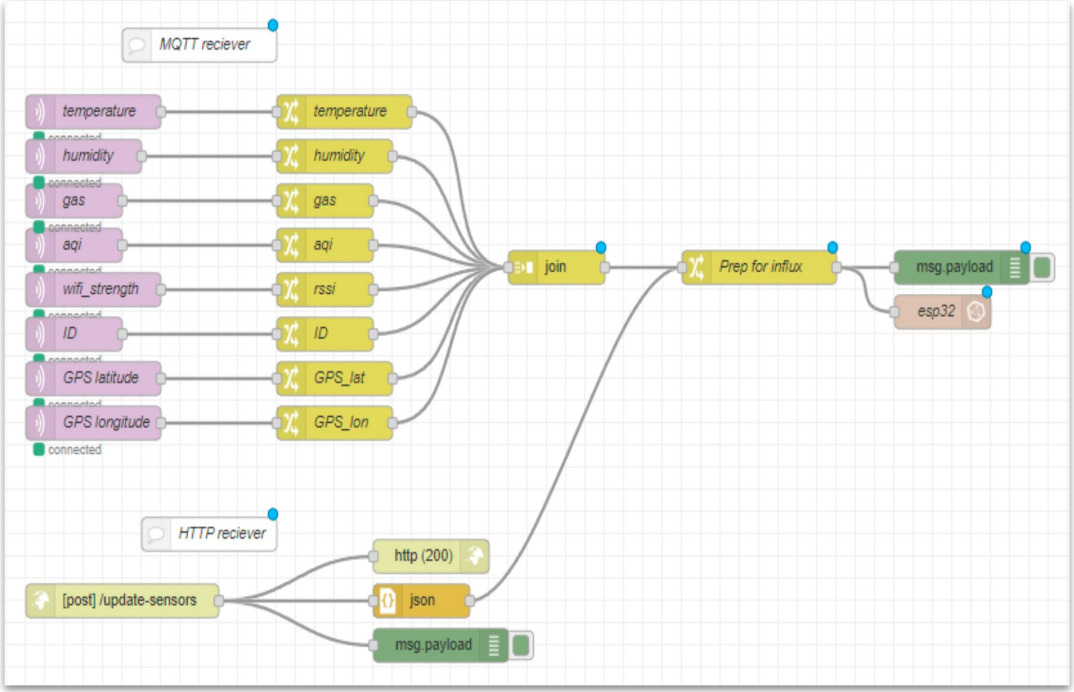
– Very visual & quick to deploy

– Drives dataflows & runs scripts: connects the nodes, e.g. raw data from MQTT scripted to JSON then sends the object to database

– Hosts HTTP server

– Manages the Telegram bot

– Harvests data from openweathermap.com

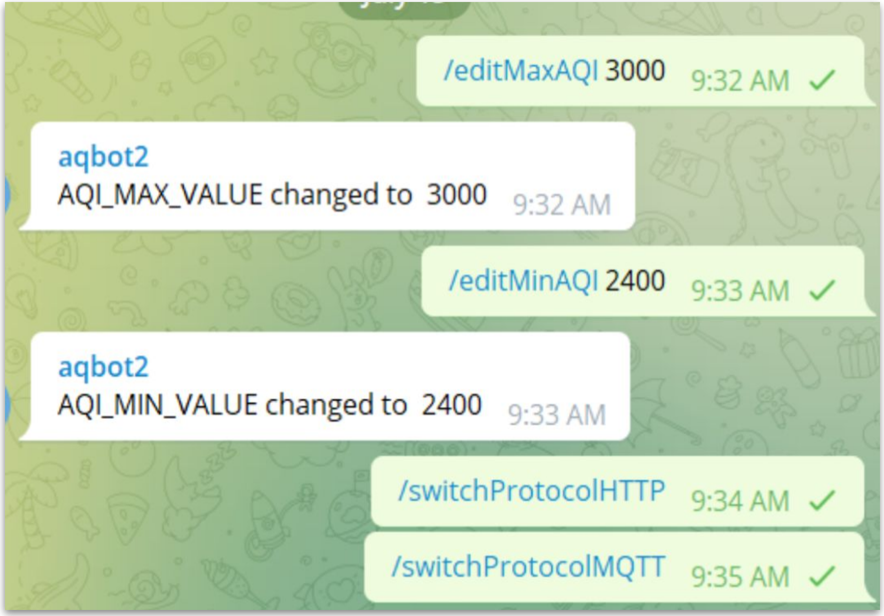


2.4 Service management

– IoT service is fully managed through a messenger service Telegram bot

– The Bot receives commands and reacts to them.

– User can for example edit how the end-device behaves (e.g. read-freq), what protocol it uses to communicate or even request to show mean sensor-data values over 4-hour period



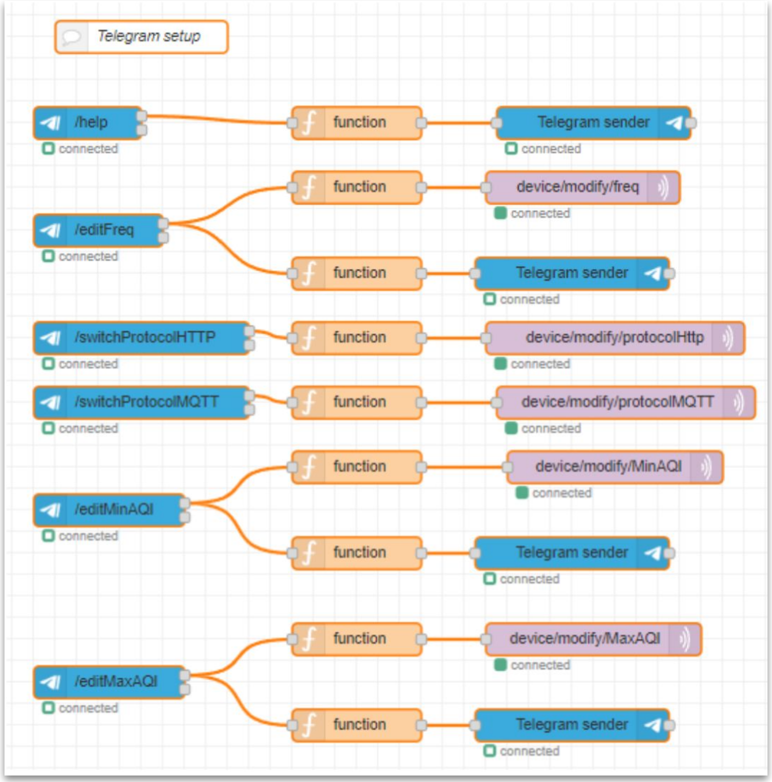
2.4 Service management – Bot

– Bot operates according to Node-red scripts

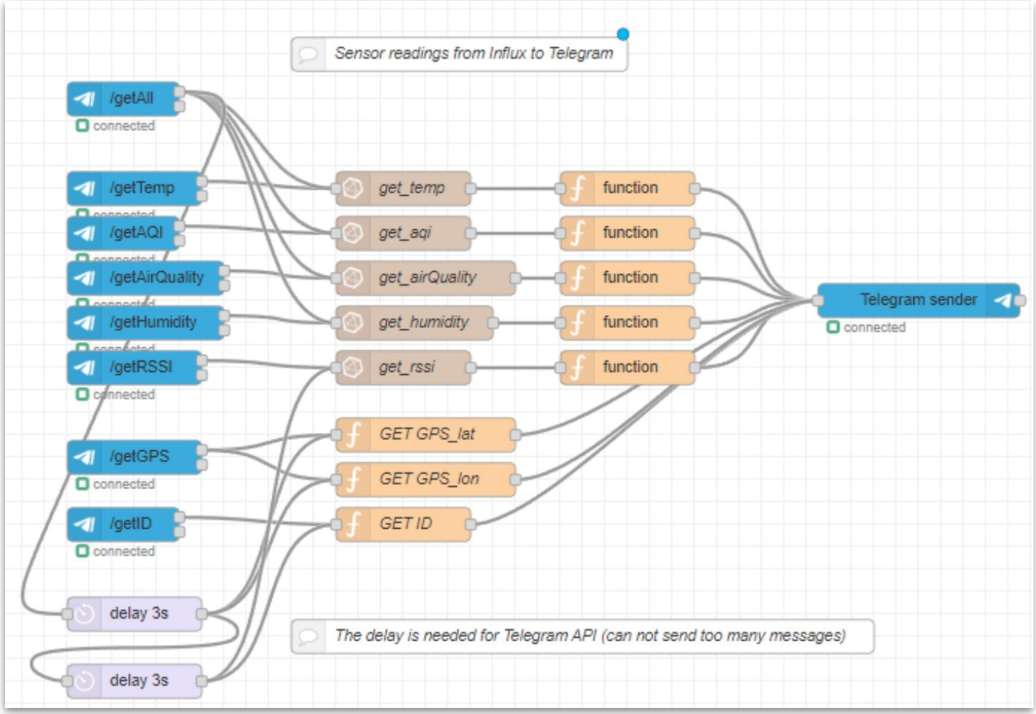
aqbot2
/help - get help.
/editFreq __ - edit test frequency (s).
/switchProtocolMQTT - switch data communication protocol to MQTT
/switchProtocolHTTP - switch data communication protocol to HTTP
/editMinAQI ___ - Change MIN_AIR_QUALITY treshold for AQI index.
/editMaxAQI ___ - Change MAX_AIR_QUALITY treshold for AQI index.

In order to get mean values from the database use:
/getTemp
/getHumidity
/getAQI
/getAirQuality
/getRSSI
/getGPS
/getID

Or simply use /getAll to get all values.



2.4 Service management - Bot



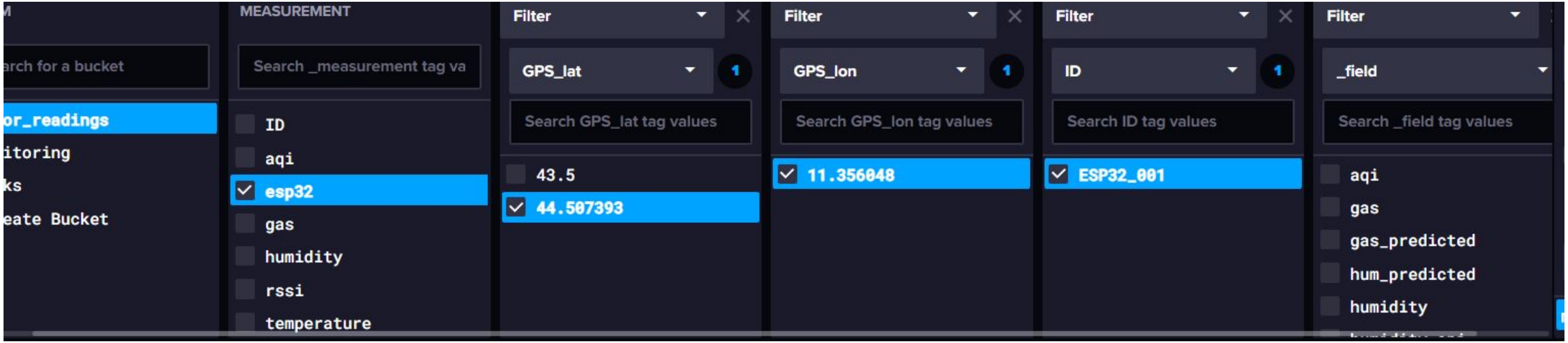
2.5 Database

– Node-red routes data to InfluxDB Cloud database – good for time-related data (*a Node-red library by InfluxDB is used*)

– Data is being received by the database as a JSON object

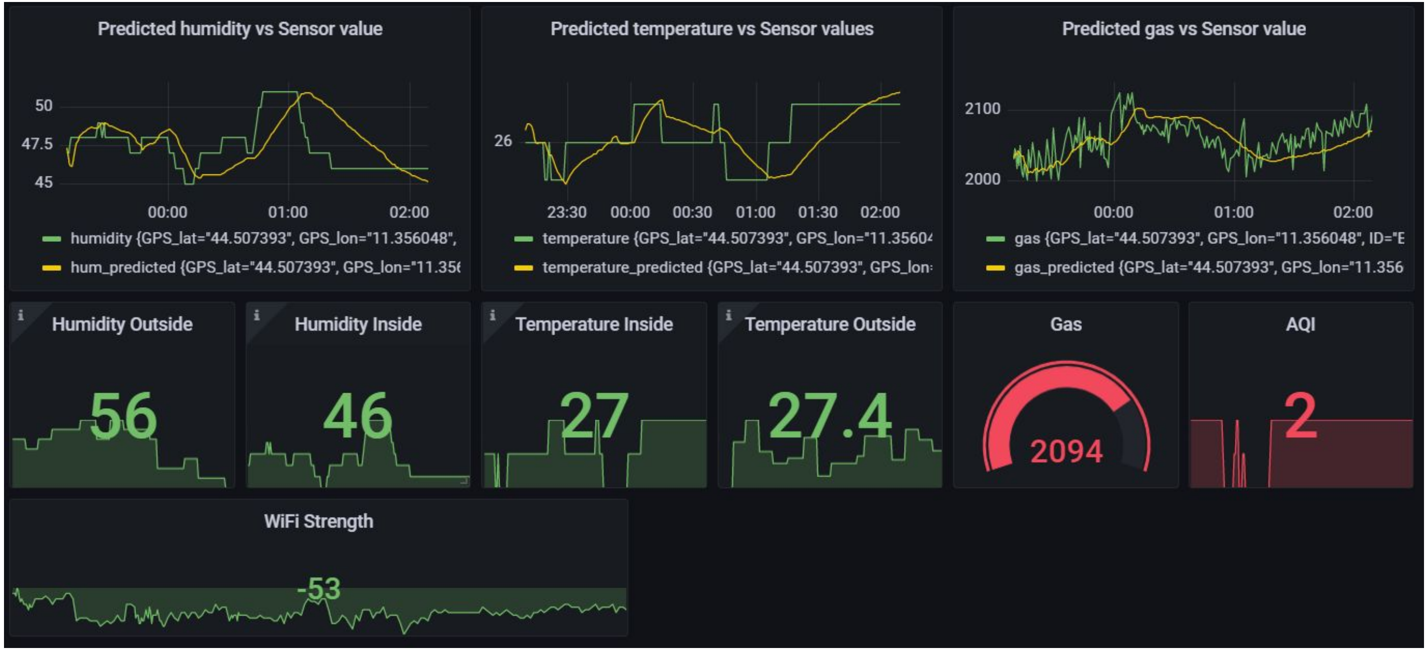
– The InfluxDB library for Node-red also enables writing queries as scripts and routing cached data (*effectively used in Telegram-Bot's mean value requests*)

– Stored data is managed with queries and sent to data visualisation tool locally.



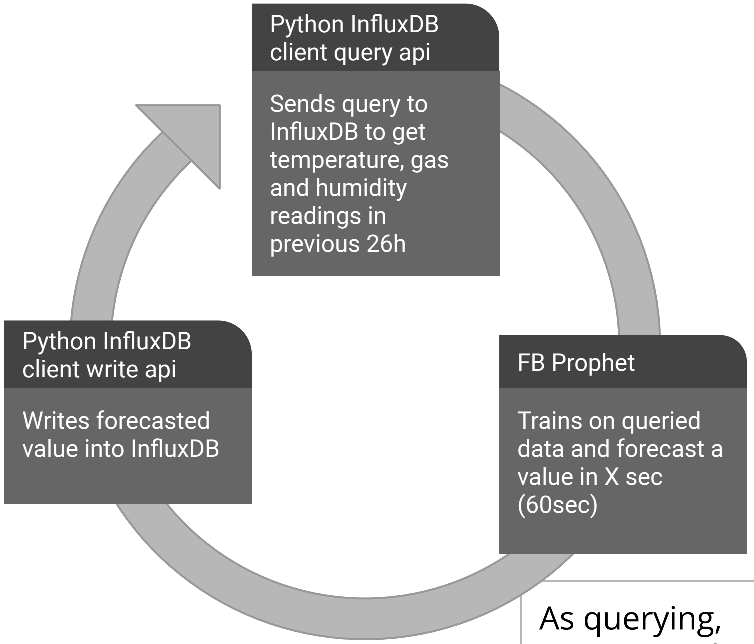
2.6 Data Visualization

Data is queried from InfluxDB cloud into Grafana dashboard panels.



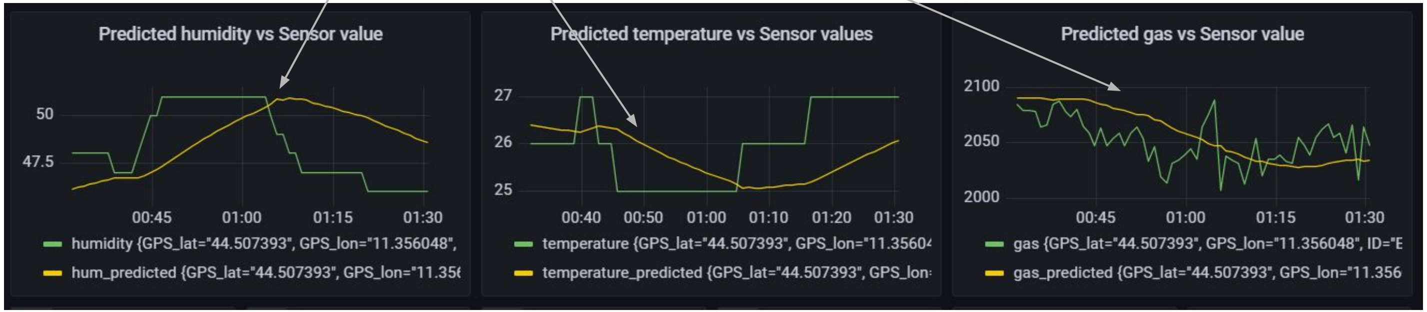
2.7 Data Forecasting

- Visualised by panels in Grafana.
- Facebook Prophet running as a python script provides forecasting.



Both predicted and measured values are being shown

As querying, training and predicting takes around 30 sec, X is set >30sec.



3 Results

We assessed Network qualities and Forecasting algorithm’s performance.

3.1 Network

Delay & PLR for send-receive trip from end-device to the database was calculated experimentally:

1) Packet sending frequency is known and is compared against packets received into the database.

2) Time is calculated simply by adding timestamp field at end-device level. Another timestamp is added when received by the database → time difference = time packet needs

3) Bad RSSI conditions were achieved by simply putting the device outside the building

Values are obviously rough estimates, as the tests were not conducted in a proper lab. Variation is around 4-6 %

Protocol	MQTT	Http
Time (avg)	26 ms	31 ms
PLR (RSSI -50 – -60 dBm)	0 %	0 %
PLR (RSSI -80 – -90 dBm)	90 %	90 %

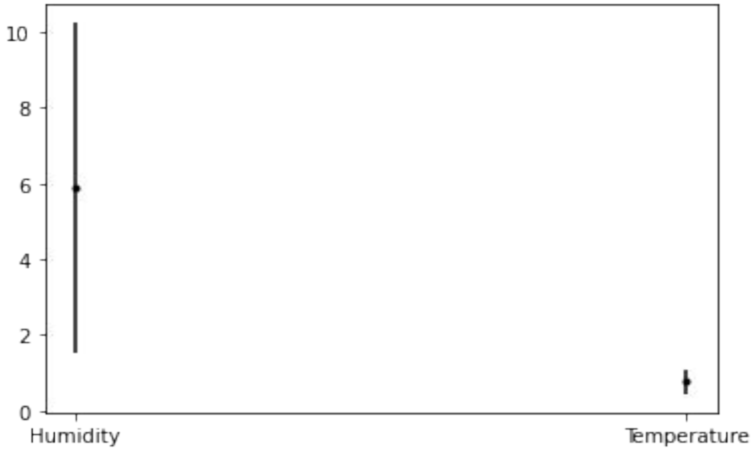
3.2 Data Analysis: Forecasting 1/2

- Model: Facebook Prophet
- Data for analysis: 1h (60 data points)

	MSE	STD
Humidity	5.87	4.36
Temperature	0.76	0.32
Gas	3912.64	3207.90

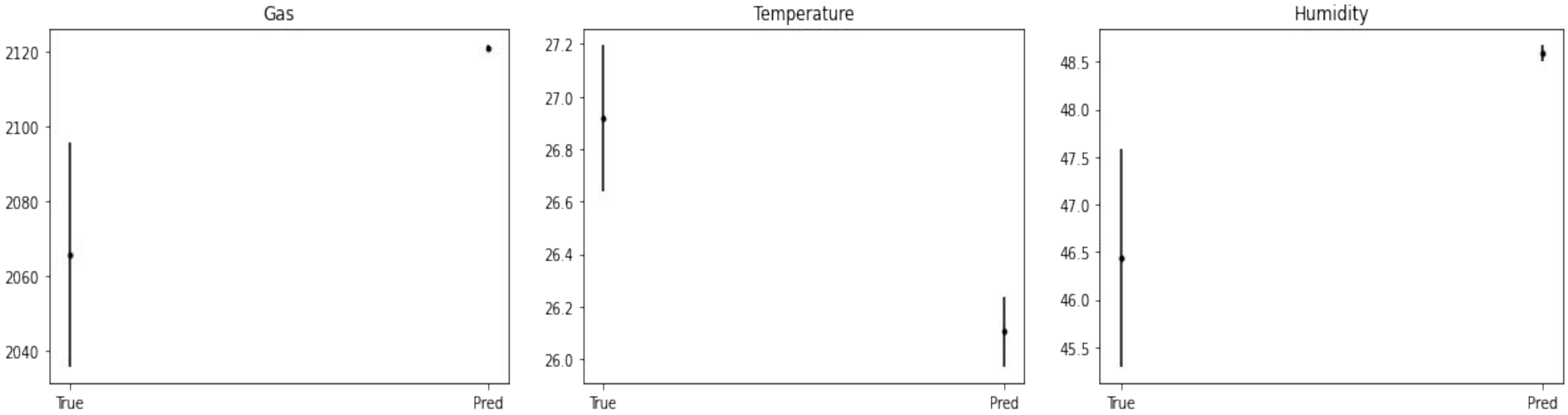
Table 1: MSE and STD of predicted and actual values

Mean Square Error and CI of predicted sensor values



3.2 Data Analysis: Forecasting 2/2

Average and Standard deviation of True and Predicted sensor values



Conclusion

– We successfully deployed a functional & expandable IoT service with a single end-device.

– Though working well, the implementation feels a bit *DIY*: (Programming with hobbyist Arduino tools, using Node-red instead of taking advantage of versatile self-built server, not ending up with an embedded end-device on its own PCB and casing & no possible customer usage was ever considered.)

– As a state of art project we learned how easy it can be to quickly deploy a simple custom IoT service e.g. for personal use