DATENKRAKEN
Dokumentation

Software-Engineering Project 4th Semester

DATENKRAKEN

MIT

Table of contents

1. DATENKRAKEN

This documentation describes the software-engineering project of semester 4.

1.1 AI Usage

Within the documentation we collectively use foundation models to generate blueprints of html tables. **Foundation models weren't instructed to generate the actual informative content of each table! Instead it's only used to reduce the workload of repetitive tasks (e.g. enumerations, html syntax, etc.).**

2. 2. arc42

2.1 Introduction and Goals

2.1.1 Introduction

DATENKRAKEN collects data regarding room climate (temperature, humidity, co2, sound level) of different rooms within the digital campus. The collected (and persisted) data is used to propose recommendations (and identify causes) to improve the climate within the rooms via a dashboard in order to achieve optimal studying atmosphere.

A indepth train of thought of our requirements engineering can be found here

Goals

Therefore the goals of this project are:

1. Improve the learning environment
2. Cause identification => Continuous improvement of the environment

2.1.2 Requirements

ID	Category	Requirement
FR.1.1	Monitoring	The system must collect room quality data (temperature, CO2 content of the air, humidity, noise level) at regular intervals and make it retrievable.
FR.2.1	Analysis / Interpretability	The system must analyze the collected data and provide current recommendations for action to improve the learning environment.
FR.2.2	Analysis / Interpretability	The system must visually prepare the data for better readability and interpretability.
FR.3.1	Operation	The system must provide the interpretable data and analyses within the DHBW Digital Campus.
NFR.1.1	Monitoring	The system must make the cyclically collected room quality data (and recommendations) retrievable within 5 minutes of collection.
NFR.1.2	Monitoring	The system must persistently store the collected room quality data.
NFR.1.3	Monitoring	The system must provide data of high quality regarding comprehensibility (unit), timeliness (see NFR 1), appropriateness, and correctness.
NFR.2.1	Analysis / Interpretability	The system may only issue truly beneficial (with regard to room climate) recommendations for action.
NFR.2.2	Analysis / Interpretability	The system must clearly distinguish current data from past data.
NFR.3.1	Operation	The system must be fully operable without the use of a keyboard.

The train of thought can be found here

2.1.3 Quality Goals

This chapters references the top 3 quality goals that are chosen to be the most beneficial for the stakeholders. Those are used for implementation based on the SFMEA analysis.

Quality Category	Quality	Description	Scenario	Referenced Failure Mode by RPN (Corrective Actions)
Monitoring	Data Quality (NFR.1.1)	The system must make the cyclically collected room quality data (and recommendations) retrievable within 5 minutes of collection.	All sensor data must reach the gold layer within 5 minutes starting from the moment when the arduino sends the data.	FA7, FA9, FA5, FA6, FA12
Monitoring	Data Quality (NFR.1.2)	The database shall have a 95% availability on minutewise status check within 1 week in production.	In production the database shall be checked regarding its availability (r/w) each minute and achieve a 95% availability.	FA9, FA12
Operation	Accessibility (NFR.3.1)	The system must be fully operable without the use of a keyboard.	This holds true to the UI. Therefore a user should be able to use the dashboard without the use of a keyboard.	None

In order to ensure each NFR, corrective measures are described within each FA in Chapter: SFMEA. Only exception is NFR.3.1 since it's not dependent on a system failure. NFR.3.1 just ensures a discussed UX guideline regarding the coming UI.

2.1.4 Stakeholder

The following (most important) stakeholders have been identified:

Stakeholder	Interest
Students	Have an interest in room quality (and thus in improvement through the system to be developed)
Developers	Have an interest in project success
Lecturers	Have an interest in room quality to ensure optimal teaching
Study Program Organization Team	Has an interest in improving room quality to enable optimal teaching
Partner Company (Student)	Has an interest in the best possible education for its students

2.2 Architecture Constraints

Technical Constraints

Topic	Constraint	Reason
Programming language for microcontroller	C++	Provides best compatibility and is the standard for Arduino programming
Programming language	Python	High readability and fast development for scripting and tooling
Operating systems	Must be executable on all major operating systems	Ensures maximum compatibility
Hardware requirements	Sensor data is collected using an Arduino	Chosen for simplicity, availability, and compatibility with sensors
Communication protocols	Sensor data is published to an MQTT broker	Enables other teams to access and use our sensor data
Testing requirements	Code coverage ≥ 75%	Helps identify bugs early and improves code quality

Organizational / Political Constraints

Topic	Constraint	Reason
Project methodology	Scrum	Central project management tool for task distribution
Decision-making processes	Architecture decisions must be made together with the team	Encourages discussion to avoid oversight and mistakes

Conventions / Standards

Topic	Constraint	Reason
Python programming	Follow the PEP8 style guide (e.g. snake_case for variables, PascalCase for classes)	Improves code readability and consistency
Arduino programming	Follow Google C++ Style Guide (e.g. camelCase for variables, PascalCase for classes	Ensures consistent, readable, and maintainable code across the Arduino codebase
Pull requests	Must be reviewed and approved by at least one other developer	To catch errors and gather suggestions before merging into main
Documentation	mkdocs	Standardized and easy-to-use tool for project documentation
Commit messages	Follow Conventional Commit	Clear and understandable commit history
Branch naming conventions	Branches are named similar to the Conventional Commit style *(feat/docs)/(feature name)*, e.g. `docs/constraints`	Clear differentiation between branches

2.3 Solution Strategy

2.3.1 Technological Architecture

The system follows a hybrid architecture consisting of hardware components (Arduino) and a containerized software infrastructure, locally executed via Docker Compose. The core components are:

- **Sensor Layer**: Multiple Arduinos collect environmental data (e.g., temperature, humidity, VOC, sound) and generate timestamps using an external NTP server.
- **Communication**: The Arduinos transmit timestamped data over Wi-Fi using the MQTT protocol to a centralized MQTT broker.
- **Data Persistence**: A Python-based subscription script (in the "Subscription Container") subscribes to MQTT topics and writes data into the "bronze" layer of a TimescaleDB database.
- **Data Architecture**: The database follows the Medallion architecture with a raw bronze layer and periodically updated materialized views for the silver and gold layers.
- **Visualization**: A separate UI container runs a Streamlit-based web application that visualizes gold-layer data.

2.3.2 Technology Choices and Rationale

- **Arduino (C++)**: Proven microcontroller platform with broad sensor support and simplicity.
- **Python**: Chosen for its rapid development capabilities and extensive libraries for scripting and data handling.
- **MQTT**: Lightweight protocol ideal for IoT communication.
- **TimescaleDB**: PostgreSQL-compatible time-series database offering strong performance and full SQL support.
- **Docker Compose**: Orchestrates modular containers, enabling fast and repeatable local development.
- **Streamlit**: Rapid frontend development framework tailored for data applications in Python.

2.3.3 Development Strategy

- **Local Execution**: All containers are started locally via `docker-compose up`.
- **Versioning**: Git is used with conventional branching (`feat/`, `docs/`, etc.) and commit standards following "Conventional Commits".
- **Testing**: A code coverage goal of ≥ 75 % is defined. Integration of automated testing via GitHub Actions is **planned as an improvement**.
- **Documentation**: `mkdocs` is used to structure technical documentation in a standardized way.

2.3.4 Architectural Decisions

Key architectural decisions include:

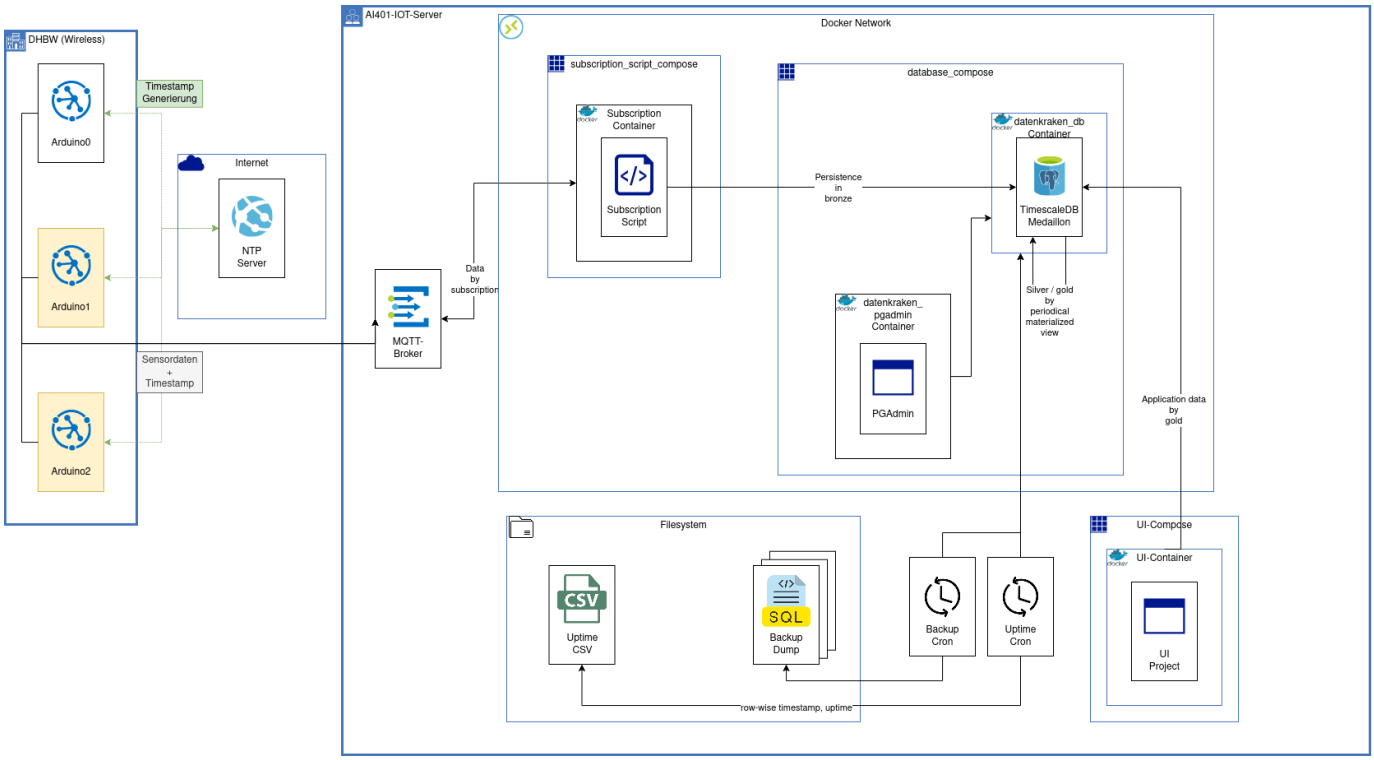
- **NTP synchronization is handled on the Arduino side** rather than edge devices to ensure accurate timestamping (ADR 1.1).
- **Subscription script is colocated with the database** within the same Docker Compose network to reduce complexity (ADR 1.2).
- **TimescaleDB + Medallion architecture** enable performant querying and clean data separation (ADR 1.3).
- **Transmission and sampling rates** are defined based on data volume and outlier mitigation (ADR 2.1 & 2.2).

2.3.5 Summary

This solution strategy is built on pragmatic technology choices, modern data architecture principles, and simple local orchestration using Docker Compose. The system is optimized for clarity, maintainability, and performance while remaining accessible to the development team. Future improvements include setting up GitHub Actions for automated testing.

2.4 Building Block View

The following picture shows the top-level decomposition of the DATENKRAKEN system. The picture mainly shows the infrastructure and data paths and their purpose. While all components are relevant for the final solution their position may change, based on their color within the picture. By default the shown components aren't supposed to move within the next iterations, except for yellow components. Those components are either not part of the bare mvp (arduino1, arduino2), or may move within the shown containers (subscription container). The decision making is primarly illustrated within chapter 4 and 9.



2.4.1 DHBW (Wireless)

This container shows all physical devices, that communicate via a wireless connection (e.g. wifi) within the digitalcampus. Therefore it primarily shows arduinos (and later edge devices like esp32). Arduinos do communicate with a remote ntp server in order to being able to create timescape for their collected data points that are sent to the proper mqtt topic. For further information regarding the arduino setup take a look at Chapter: Arduino Components.

2.4.2 AI401-IOT-Server

This container holds all components that are run within the IoT Server within the digitalcampus. Except for the mqtt broker those components are either a docker compose or (within a compose) docker containers. Containers within the compose are suppossed to communicate via the docker container network. Composes communicate via the internal linux network.

Subscription Container

This container holds a script that primarly fetches new data from the mqtt broker in order to persist this data into the bronze layer of our used database.

This is held within a seperate docker compose in order to seperate its configuration and its resetability. It's also part of the central docker network datenkraken-db-net, in order to ensure that the database port doesnt have to be exposed to the host.

DB-Container

This container holds "TimescaleDB", a postgres database service with plugins for optimized timeseries handling. It's architecture is based on the medaillon architecture. Therefore it consists of a table for the bronze layer (raw data), and either a materialized view (periodically fetched) for the silver layer (cleaned, augmented) and gold layer (production ready data).

This is held within a seperate docker compose in order to seperate its configuration and its resetability. It's also part of the central docker network datenkraken-db-net, in order to ensure that the communication with the subscription container doesnt have to be via the docker host.

PGAdmin

This container holds "PGAdmin", a web interface to interact with the postgres database.

Cron Jobs

We suppose two cron jobs. One that fetches the database and retrieves a backup in the form of a pgdump each day (00:00), and one the retrieves the uptime of the database each minute. The output of the uptime is saved into a csv file of the form postgres_start_time,recorded_time.

The backup is then persisted on cloud periodically.

UI-Container

This container holds the UI logic by using the streamlit framework. The UI Container is ran **within the datenkraken-network**.

2.5 Design Decisions

The following section shows our architectural decisions by sprints (initially introduced -> edits below), and their implication:

2.5.1 Sprint 1

ADR 1.1: Usage of NTP Server

Section	Description
Date	17.07.2025
Context	The usage of the NTP server was first thought to be made on a edge device (f.e. ESP32). This was cancelled because of the following reason.
Decision	The decision was made since a NTP server call on the side of ESP32 would result in a loss of information (complete loss or inaccuracy of timestamp for collection time of a data point)
Status	Accepted
Consequences	It moves the implementation of a edge device to later phases of the project => This means that we may run out of time if we still want to use one.

ADR 1.2: Subscription script in Persistence-Compose

Section	Description
Date	17.07.2025
Context	A script is needed to forward the data from the mqtt broker towards the database. This step is done via a script within the persistence compose (the same as the database).
Decision	The decision a strong connectivity between database and the script since its held within the same network. Other than that the structure is more clear than holding it in two different composes.
Status	Accepted (04.08.2025)
Consequences	This means that if components of the compose must be replaced the script and/or the database must be stopped to. For now this should not be a problem since the mqtt broker can hold data until fetched (QOS). It's not 100% clear whether this setting is set, so this decision might change. => Accepted: Since a loss of a small portion of datapoints is not critical for our use case. The stability of the script is ensured by tests and a double auto reconnect logic of docker and paho-mqtt. Collisions with a duplicate broker client id were also tested, and do not result in a loss of data points, but only in a slight instability and flooding of the logs, which can be treated fairly well.

ADR 1.3: TimescaleDB / Medaillon architecture of database

Section	Description
Date	17.07.2025
Context	TimescaleDB is used for storing the time series sensor data. It's architecture is based on the medaillon architecture. Bronze layer is suppossed to be a table, but silver / gold layer is suppossed to be a materialized view.
Decision	The decision to use timescaledb is made since, all group members are familiar with sql, whilst still having sota timeseries performance. The medaillon architecture ensures NFR 1.2 (bronze layer) and NFR 1.3 (silver layer). A materialized view is used since it ensures that less overhead is needed (tables would propose replication), by that we can ensure NFR 1.2 by backing up the bronze layer table => Silver and Gold layer can be restored on compute time.
Status	Accepted
Consequences	This means that all of data cleaning must be done within the database. No external scripts can be used since of the usage of materialized views.

2.5.2 Sprint 2

ADR 2.1: Transmission Rate

Section	Description
Date	22.07.2025
Context	We set the transmission rate of each message (by sensor) to a rate 30s / message
Decision	The decision was made by calculating the final data volume for a time period of 60 days (project run time)
Status	Accepted
Consequences	It implicates a aggregation of datapoints on periods of 30s intervalls. And a data volume of roughly 172k lines per table.

ADR 2.2: Sampling Rate

Section	Description
Date	22.07.2025
Context	We set the sampling rate of each sensor to: Sound Sensor: 1s -> Vector[30], Humidity: 30s -> Vector[1], Temperature: 30s -> Vector[1] VOC: 5s -> Vector[6]
Decision	The decision was made by calculating the estimating the probability of outliers for each sensor (in order to have enough data to aggregate data to reduce outliers)
Status	Accepted
Consequences	A loss of information in the time between the samples.

ADR 2.3: Multi Table Timescale Setup

Section	Description
Date	24.07.2025
Context	We use a hypertable in our database for each sensor.
Decision	The decision was made since otherwise we introduce many null values since our sampling rate of the sensors are different. This is a problem since timescale interprets null as a actual value.
Status	Accepted
Consequences	Slighly more difficult joining strategies may be needed.

ADR 2.4: Composite index

Section	Description
Date	24.07.2025
Context	We create a composite index on arduino_id and time.
Decision	Since our use case will need both columns for filtering often we introduce composite index on all layers => Allows fast tracing of values from gold to bronze layer and vice versa.
Status	Accepted
Consequences	Slightly less insert performance (negligible with our transmission rate), higher memory usage

2.5.3 Sprint 3

ADR 3.1: Composite primary key

Section	Description
Date	31.07.2025
Context	We create a primary key (+composite index) on id and time.
Decision	Since sqlalchemy requires a primary key in order to work we introduced a primary key in our table. This is no anti pattern when using timescale if a composite primary key (which includes the time) is used.
Status	Accepted
Consequences	Slightly more memory usage (esp. since additional index).

ADR 3.2: Cron Job Backup

Section	Description
Date	04.08.2025
Context	We use a cron job to backup our database.
Decision	Since the backup has to be created periodically and be saved by hand regularly a cron job was chosen as our backup strategy. It creates a pgdump which is archived and persisted on a private cloud storage. The size was estimated on a full backup with around 600mb.
Status	Accepted
Consequences	If a backup is missing, the whole data is lost since it's not a incremental backup. But the chance is pretty low since it lays on a private cloud storage.

ADR 3.3: Cron Job Uptime

Section	Description
Date	04.08.2025
Context	We use a cron job to track our uptime.
Decision	Since we just have to track our uptime for up to one week we decided to use a cron job to fetch the database uptime and store it within a csv file.
Status	Accepted
Consequences	At first glance we do not have any dashboard (like in more complicated containerized methods). But we do have control over the full configuration and what we do with it later on.

ADR 3.4: Architectural Decisions

Section	Description
Date	04.08.2025
Context	We introduce a deleted_at column in bronze layer.
Decision	We use soft deletes in our bronze layer in order to not lose any data.
Status	Accepted
Consequences	Higher memory usage => But our memory usage is overall pretty low therefore we do not want to lose data if not necessary.

2.5.4 Sprint 4:

ADR 4.1: Sensor data aggreggation intervalls (silver)

Section	Description
Date	13.08.2025
Context	In order to reduce memory layerwise we propose a aggregation via average in certain time intervalls in a way we can quantify information loss. => Variational Coefficient (see Chap: Database)
Decision	Temperature & Humidity: 15 Minutes, Noise: 30 Sec, Voc: 5 Minutes
Status	Accepted
Consequences	We may lose information of local trends for our machine learning use case (which could potentially need further information for f.e. moving avgs or other rolling features) but since the time seems not to allow our ml use case this is doesn't propose a risk. And even if we need this kind of information of local trends we can still choose a smaller intervall since all layers besides bronze is constructed as a view.

2.5.5 Sprint 5:

ADR 5.1: Passthrough view for each sensor in gold layer

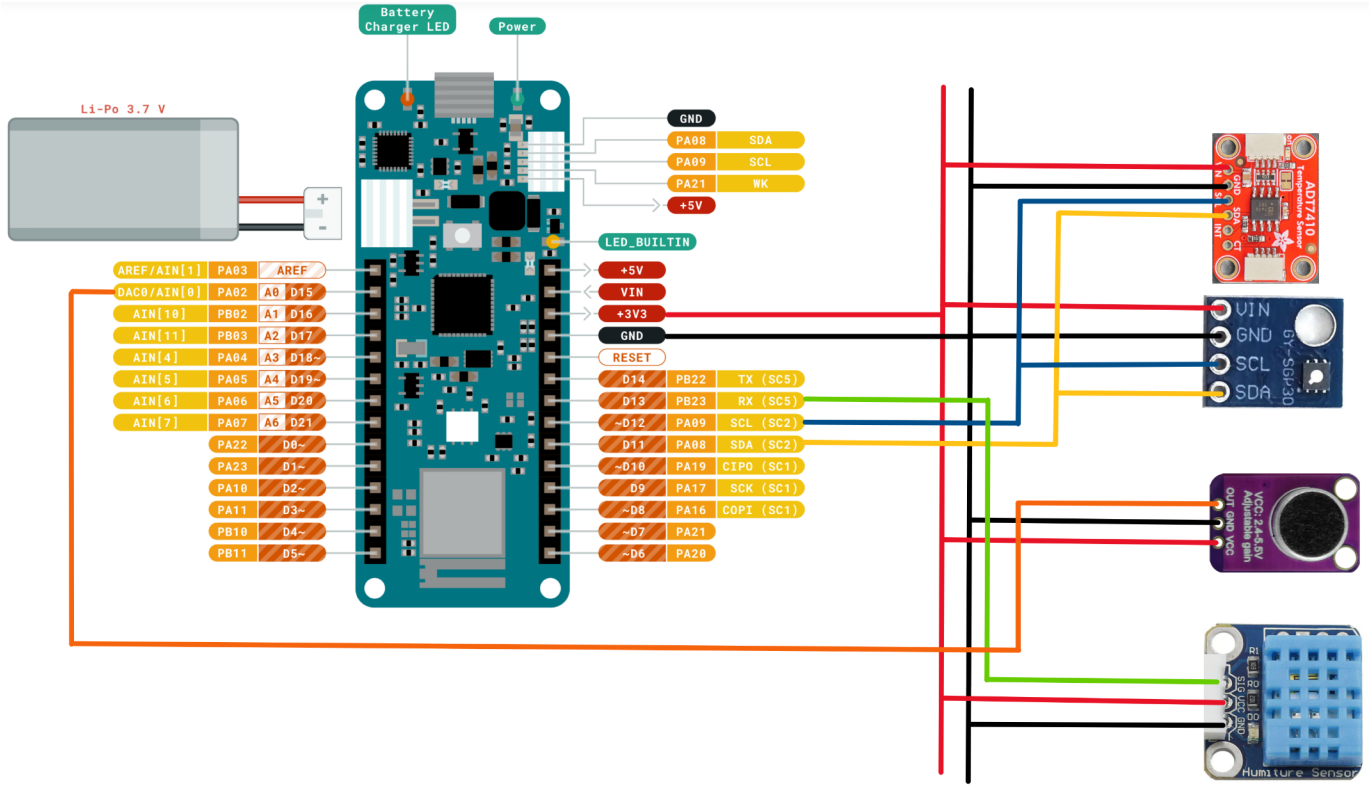
Section	Description
Date	18.08.2025
Context	Our gold and silver layer would be almost identical in terms of data, since our recommendation algorithm relies on basic interval checks. Therefore we have to reduce memory consumption, since copying would introduce a lot of redundance.
Decision	We define passthrough views in our gold layer with a unified naming.
Status	Accepted
Consequences	A passthrough view would reduce the memory consumpion to 0 whilst remaining the performance of the materialized view, since we do not have to aggregate any further data. Due to our naming convetion within the gold layer we can later extend the view with a materialized view whenever we need a more complex business logic and data.

3. 3. Arduino

3.1 Hardware Components

- Arduino MKR Wifi 1010
- Temperature sensor (ADT7410)
- Humidity sensor (sunfounder Humiture Sensor)
- CO2 sensor (GY-SGP40)
- Noise level sensor (GY-MAX4466)

3.2 Circuit Plan



3.2.1 Temperature & CO2 Sensor:

Arduino	Temperature
VIN	VCC
GND	GND
SCL	SCL
SDA	SDA

3.2.2 Noise Level Sensor:

Arduino	Temperature
A0	OUT
GND	GND
VCC	VCC

3.2.3 Noise Level Sensor:

Arduino	Temperature
D13	SIG
VCC	VCC
GND	GND

3.3 Output format

Each sensor has a different unit of measurement. Therefore different datatypes are needed.

1. Temperature
The temperature is measured in celcius (°C), the values are floats and can be negative.
Unit: °C

2. Humidity Sensor
The Humidity sensor returns values ranging from 0% to 100%, anything other is an invalid value.
Unit: relative Humidity in %

3. CO2 Sensor
The CO2 sensor is dependant on temperature and humidity. These values are needed to calculate the VOC-Index. The VOC-Index is ranging value from 0 to 500, any other value is invalid.
Unit: VOC-Index

4. Noise Level
The Noise sensor returns a analog value ranging from 0 to 1023. This value needs to be used to calculate a db value.
Unit: integer

3.4 Required Libraries

- Platform.ini holds all used Libraries

3.5 Deployment of new Arduinos

3.5.1 Prerequisite

To upload the code to an arduino you need a software like pio or the arduino IDE.

3.5.2 Clone the Repository

For the deployment you need to download the code from our repository.

3.5.3 Config

For the configuration of your needs you rename the `arduino_secrets_template.h` file to `arduino_secrets.h` in the arduino folder and adjust the following information in the file accordingly.

```
#define ROOM_ID 1
#define TOPIC "topic/subtopic" //Sensortype is automatically added
#define SECRET_SSID "wifi ssid"
#define SECRET_PASS "wifi password"
#define BROKER "broker address"
#define MQTT_USER "username"
#define MQTT_PASS "userpassword"
```

The topic is then combined with the sensor and the room id to make it unique. In this example the topic to publish would be `topic/subtopic/sensortype/1`.

3.5.4 Upload

After changing the filename and the information in the file you upload this to the arduino with pio or the arduino ide.
When you plug the arduino in it connects automatically to the wifi and the broker you provided and will start publishing the data on the configured topic.

3.6 MQTT

3.6.1 Topicformat

`dhbw/ai/si2023/<GROUP-NUMBER>/<SENSOR-TYPE>/SensorID`
(Group 6)

Sensortypes

- temp (temperature)
- mic (microphone / noise level)
- hum (humidity)
- co2 (CO2 Sensor)

3.6.2 Messageformat

The messages are sent in the json format. This json object contains

- unix timestamp (examine when the data was gathered)
- value array (the gathered data)
- sequence number (to be able to check if a message was lost)
- meta data is used to transfer the arduino id (which arduino sent the message)

A example json object looks like this

```
{
  "timestamp": "1753098733",
  "value": [23.45],
  "sequence" : 123,
  "meta": {
    "device_id": 303
  }
}
```

3.7 Effects of Room Parameters on Learning Performance

A healthy and comfortable indoor climate is a crucial foundation for successful teaching and effective learning. In particular, in classrooms and smaller lecture halls with high occupant density, factors such as room temperature, relative humidity, and the concentration of airborne pollutants directly affect the well-being, health and cognitive performance of those present. Numerous scientific studies show that an optimally set indoor climate not only improves subjective comfort but also enables measurable gains in concentration and learning performance.

3.7.1 Room Temperature and Cognitive Performance

The temperature in indoor spaces plays a central role in students performance. Studies in European schools have shown that pupils' cognitive performance improves significantly when room temperature is lowered from 30 °C to 20 °C (Pawel Wargocki, 2012, p. 582). Likewise, very low relative humidity can irritate the respiratory tract and increase the risk of infections.
At temperatures between 20 and 22 °C, most people feel comfortable and can concentrate optimally.
However, when the temperature rises above 25 °C, productivity subconsciously decreases by about 1 to 2 % per additional degree Celsius (Pawel Wargocki, 2012, p. 586). At extreme temperatures above 30 °C, physical complaints such as heat stress, accelerated pulse and rapid fatigue also occur. Children are particularly sensitive to high temperatures, leading to observed fatigue, declining attention and an increased error rate in tasks in overheated classrooms.

3.7.2 Relative Humidity and Health

Humidity also significantly influences comfort and health indoors. A range of 40 to 60 % relative humidity is considered ideal (Peder Wolkoff, 2007, p. 854). At humidity levels below 30 %, mucous membranes dry out, leading to irritated eyes, dry throat and increased infection risk. At the same time, the respiratory tract's ability to defend against pollutants and pathogens is reduced.
Conversely, high humidity above 60 % promotes mold growth, which in turn increases the risk of respiratory diseases and allergies. High humidity also impairs the body's temperature regulation, as sweat evaporates less effectively.

3.7.3 VOC Index and Indoor Air Quality

The VOC (Volatile Organic Compounds) index is an important indicator of indoor air quality and ventilation needs. VOCs are emitted from building materials, furniture, cleaning products, and human activity. Elevated VOC levels can lead to symptoms such as headaches, dizziness, fatigue, irritation of the eyes and airways, and reduced cognitive performance. A VOC index of 100 or less is generally considered good air quality, while values above 200 suggest poor air quality that requires immediate action. Prolonged exposure to elevated VOC levels can also aggravate asthma and allergies. Regular ventilation approximately every 20 to 45 minutes or the use of mechanical ventilation systems is necessary to keep VOC levels at a healthy level. Good ventilation not only lowers VOCs but also reduces sick days and improves learning performance. The use of low-emission building materials and cleaning products also significantly improves indoor air quality.

3.7.4 Noise Level and Concentration in Class

In addition to temperature, humidity and air quality, noise level also has a significant effect on students' learning and working ability. Acoustic disturbances from conversations, street noise or technical equipment cause students attention to wane and impair their cognitive performance. Even at a continuous sound level of about 50 dB(A), comparable to normal conversational noise, many students begin to work slower and make more errors (Daniel Connolly, 2019). Background noises that overlap speech are particularly detrimental to understanding instructional content. This is especially problematic for younger children and non-native speakers, as they rely more on clear acoustic perception. Studies show that speech intelligibility in the classroom decreases once the signal-to-noise ratio (SNR) falls below +15 dB (Maria Klatte, 2010). Prolonged high noise levels can also trigger stress reactions: increased blood pressure, elevated pulse and the release of stress hormones impair not only learning ability but also students' long-term health.

3.7.5 Conclusion

In summary, an optimal indoor climate is essential for students’ health, well-being and learning ability. Based on the researched optimal values, a set of guidelines has been developed:

Temperature Management

- Optimal air temperature between 20 and 22 °C
- Implement cooling or shading strategies if indoor temperature exceeds 25 °C
- Avoid direct sunlight by using blinds or shades in warm/sunny weather
- Use continuous monitoring to detect overheating early

Humidity Control

- Optimal relative humidity between 40 and 60 %
- Use humidifiers if RH < 40 %
- Ventilate regularly or use dehumidifiers if RH > 60 %
- Watch for signs of mold growth at high humidity levels to prevent health risks

VOC Reduction through Ventilation

- VOC index should be < 100 (optimal)
- Ventilate immediately if VOC index > 100
- Identify and remove sources if index remains high

Noise Reduction

- Keep ambient noise at a maximum of 50 dB(A)
- Signal-to-noise ratio of at least +15 dB
- Keep windows facing busy streets closed during lessons
- Minimize background noise from technical equipment

3.7.6 References

D. G. Shendell, R. P. (2004). Associations between classroom CO2 concentrations and student attendance in Washington and Idaho.
Daniel Connolly, J. D. (2019). The effects of classroom noise on the reading comprehension of adolescents.
Maria Klatte, J. H. (2010). Effects of Classroom Acoustics on Performance and Well-Being in Elementary School Children: A Field Study.
Pawel Wargocki, D. P. (2012). Providing better thermal and air quality conditions in school classrooms would be cost-effective.
Peder Wolkoff, S. K. (2007). The dichotomy of relative humidity on indoor air quality.
Wargocki Pawel, W. D. (2017). Ten questions concerning thermal and indoor air quality effects on the performance of office work and schoolwork.

4. 4. Components

4.1 Frontend Documentation

This Document describes how the frontend works and what role every necessary file plays. It will also be explained how to start it.

4.1.1 General Information

The Frontend is coded using a variety of Librarys like for example Streamlit. Instead of installing every Dependency individually, the user can run the command 'uv sync' inside the folder 'frontend' (if uv is installed). After that the command 'uv run streamlit run app.py' starts the frontend (Only if ran from inside the frontend folder). If everything was done correctly, a new browser window will automatically pop up, showing the content.

4.1.2 app.py

App.py represents the main/landing page. It functions as a generel and quick overview over the rooms which are being meassured. It shows the current value ov every meassured parameter and the current status of the room which is defined by the 'worst' current value. That means that if one Parameter is in the critical range, the room status is displayed as critical. Indepth pages are also initiated in here via page_definition package.

4.1.3 page_definition

The package 'page_definition' withholds the detailed views of the corresponding rooms. The general overview page is defined via the overview module, whilst the indepth pages are defined via the submodule generic_analytics page.

4.1.4 Utils / Widgets / Database

Those packages primarily provide functionality to gather and augment data (Utils / Database) and show them on the ui (widgets). **For a deeper understanding, we HIGHLY recommend taking a look in the code. Every function is documented. Don't forget to take a look at the test cases in order to find out how functions should be used!**

4.1.5 Resilience & Error Handling

The frontend contains several mechanisms to remain usable and informative even when underlying services (database, external APIs) degrade. Below is an overview of the implemented strategies.

1. Database Query Behavior

All low-level selection helpers (`commit_select`, `commit_select_scalar`) return an empty list `[]` on execution errors while logging the underlying exception. This keeps callers simple (`if not rows:` works for both *no data* and *error* states) and prevents unhandled exceptions from crashing pages.

2. Database Health Check

`is_db_healthy()` performs a lightweight `SELECT 1` to distinguish between: - Database unreachable / credentials invalid → show a clear error panel - Database reachable but empty → show an empty-state message instead of a technical error

3. Unified Error Panels

`render_error_panel(title, details)` provides a consistent visual block for: missing data, connectivity failures, or unavailable subsystems. This improves scanability and reduces ad-hoc `st.error()` scattering.

4. Staleness & Freshness Logic

The staleness helpers (e.g. `temperature_below_five_minutes`) now: - Normalize timestamps to UTC - Tolerate small clock skews by using the absolute delta - Treat absence of data as stale (False) with a logged hint

5. Time Series Robustness

The history widget guards: - Empty DataFrames (shows a warning instead of calling `.min()` / `.max()` on empty data) - Exceptions during fetch (shows an inline error and aborts the widget render) - Uses timezone-aware timestamps to avoid inconsistent range calculations.

6. External API (Weather) Degradation

Weather fetch failures surface as `st.info("Weather data temporarily unavailable.")` rather than a console print. Downstream recommendation logic is skipped when data is absent, preventing secondary errors.

7. Progressive User Feedback

Toasts (success notifications) are only emitted when: - At least one valid sensor value exists - Database health check passes - A per-session / per-room toast flag is not yet set (prevents notification spam)

8. Severity Color Coding

- `st.error` (Red): Hard failure (DB unreachable, fatal query error)
- `st.warning` (Yellow): Staleness or non-critical data quality issues
- `st.info` (Blue): Temporary unavailability (e.g. weather API) or suppressed diagnostics
- `st.toast` (Green): Successful load confirmation

9. Rationale

The layered fallback approach reduces alarm fatigue: true infrastructure problems are visually distinct, while normal early-phase empty data or transient external API hiccups remain low-noise. Returning empty lists (instead of raising) keeps caller code linear and prevents fragmented error handling.

10. Future Enhancements

- Export metrics (health counters, last failure timestamps)
- Distinguish "no data yet" vs. "filtered range empty" in analytics views
- Progressive backoff / circuit breaker on repeated DB failures
- Optional observability integration (Sentry / OpenTelemetry)
- Configurable strict mode to differentiate error vs. empty result explicitly

4.2 Frontend Error Handling (Detailed Reference)

This page provides the extended version of the frontend error handling & resilience concept. A condensed summary lives inside `frontend.md`.

4.2.1 1. Database Access Layer Resilience

Problem: Unhandled exceptions from SQL queries caused page crashes. **Solution:** `commit_select` / `commit_select_scalar` return `[]` on failure (error is logged). Callers use simple falsy checks. **Impact:** No propagation of transient DB errors into UI; consistent iterable type.

4.2.2 2. Database Health Monitoring

`is_db_healthy()` performs a lightweight `SELECT 1`. Distinguishes: - DB unreachable -> explicit error panel - DB reachable but empty -> empty state (not an error)

4.2.3 3. Unified Error Panels

Reusable `render_error_panel(title, details)` ensures consistent styling for connectivity, data absence, and subsystem failures.

4.2.4 4. Time Series & Staleness Robustness

- UTC normalization for timestamps
- Absolute time difference to absorb small clock skew
- Guard empty DataFrames before min/max
- Exception wrapper around history fetch prevents cascading failures

4.2.5 5. External API Resilience

Weather API failures yield informational user feedback instead of stack traces; downstream logic checks for `None` gracefully.

4.2.6 6. Toast & Feedback Strategy

- Toast only after successful real data load (has at least one non-None value)
- Per-session & per-room flags to avoid spam
- Distinguish load success from mere page render

4.2.7 7. Severity Color Coding

Severity	Component	Usage Examples
Red (`st.error`)	Hard failures	DB unreachable, fatal query errors
Yellow (`st.warning`)	Degradation	Stale sensor data (>5m)
Blue (`st.info`)	Informative transient	Weather API unavailable, staleness check skipped
Green (`st.toast`)	Success	Data load completed

4.2.8 8. Rationale

Color & fallback hierarchy reduces alarm fatigue: treat true outages distinctly, while soft failures (external APIs) remain unobtrusive. Empty data is common early in deployments and must not be conflated with infrastructure faults.

4.2.9 9. Testing Coverage Snapshot

- Selection helpers: success + simulated exception (empty list fallback)
- Staleness helpers: fresh vs stale vs missing data
- DataFetcher: partial sensor availability & empty states
- History widget: empty dataset & exception guard

4.2.10 10. Future Enhancements

- Metrics endpoint exporting health counters & last error timestamps
- Differentiation: "No data yet" vs. "Filtered range has no rows"
- Incremental circuit breaker for repeated DB failures (progressive backoff)
- Optional Sentry/OpenTelemetry integration for richer diagnostics
- Configurable toggle: return `None` vs `[]` for error signaling in advanced use-cases

4.2.11 11. Design Principles

1. Fail soft, not silent (log errors, show user-friendly panel)
2. Preserve UX continuity over strict error purity
3. Make severity visually scannable
4. Prefer idempotent, side-effect free health probes
5. Keep caller contracts simple (iterables not `Optional[Iterable]`)

4.2.12 12. Quick Reference (Cheat Sheet)

Aspect	Mechanism	Location
DB health	`is_db_healthy()`	`database/sql/engine.py`
Safe selection	empty list fallback	`database/sql/engine.py`
Error panel	`render_error_panel`	`frontend/utils.py`
Staleness check	absolute delta <= 5 min	`utility/currentness.py`
History safety	try/except + DataFrame guards	`generic_analytics/widgets/history_widget.py`
Weather resilience	info message on failure	`generic_analytics/widgets/utils.py`
Toast gating	session state flags	`overview.py`, `generic_analytics.py`

4.3 Database

The following chapter describes our decision regarding our database. The database name used within the project is: datenkraken.

There are three users within the database: **ui** -> read-only on gold schema **dev** -> all privileges on tables within bronze, silver, gold except for delte **datenkraken_admin** -> all

Configuration via .env:

```
POSTGRES_USER=
POSTGRES_PASSWORD=
UI_PASSWORD=
DEV_PASSWORD=
```

4.3.1 Bronze

The bronze table has to store roughly 172k messages in the 60 days of our project period as discussed here . As specified the sensor data must be stored in the following fields (with additional fields of the messageformat).

Since we collect sensordata using different sample rates in order to be more memory efficient, we propose 4 tables in the bronze layer. Each stores the raw data of each sensor.

Fields that are part of all tables:

1. id: BIGSERIAL => used for composite primary key with the time field. => Composite primary key since this is the proposed way of timescaledb without inheriting any bottlenocks due to the index. (See down below: Best practices)
2. time: TIMESTAMPTZ
3. arduino_id: text -> denormalized since timescaledb does a dictionary compression / + enums do not allow to delete individual values within enum 8.7. Enumerated Types
4. deleted_at: TIMESTAMPZ -> soft delete

Field per table:

1. temperature: float4
2. humidity: smallint
3. voc: smallint
4. noise: smallint

This type of bronze layer definition is chosen due to the public recommendation by timescaledb. First we thought about storing all values for each 30 seconds intervall within an array. By that we can ensure to persist all data within one table. But we could not find any references that ensure, that this wouldn't lead to performance issues on data aggregation as it's common with the array type on relational databases.

Also if we would persist it within a single table (without arrays) we would introduce many null values. That would lead to a growth of memory space since timescaledb interprets a "Null" as actual value. Best Practices for Picking PostgreSQL Data Types.

Therefore we chose a multiple table design as proposed in Best Practices for Time-Series Data Modeling: Single or Multiple Partitioned Table(s) a.k.a. Hypertables

The tables are named in the following schema:

bronze.SENSORTYPE

Bronze table size

In order to estimate the final size of the database we inserted 172k rows of realistic dummy data into the temperature table. And 5m rows in the noise table. By examining its relation and index size we wanted to make sure we dont run into future trouble. We found out that the biggest table within the 60 day period of our project would be the noise table with 341mb of memory usage.

On ~172k temperature entries => 60min x 24h x 60days

```
datenkraken=# SELECT pg_size_pretty(hypertable_size('bronze.temperature'));
 pg_size_pretty
----------------
 106 MB
(1 row)
```

On ~5m noise entries => 60min x 24h x 60days x 60 entries per minute

```
datenkraken=# SELECT pg_size_pretty(hypertable_size('bronze.noise'));
 pg_size_pretty
----------------
 341 MB (1 row)
```

Although this table size isn't very high we decided to create a composite index on the columns time and arduino_id in order to speed up read processes. We only need a composite index out of three reasons:

1. Our use case primarily will only need two filters within the dashboard -> in general by time (to scale diagrams), by time and arduino_id (to view each room). Therefore a composite index should be created.
2. Even if we filter only for time (for development purposes) timescale has a index on time by default.
3. Even if we filter only for arduino_id, we defined segments (partitions) for arduino_id on our hypertable definition.

When we want to detect bugs later on this could therefore help a lot, since in the worst case we had to trace it from gold to bronze layer and vice versa.

4.3.2 Silver

The purpose of this layer is to create filtered and cleaned layer of data. We develop the layer definition by the typical data quality criterias:

1. **Accuracy** - How correct is the data values?
2. **Completeness** - Is all information present?
3. **Consistency** - Does the data match other trusted data sources? (Not checkable => Ignored)
4. **Validity** - Does the data conform to the predetermined format & constraints?
5. **Timeliness** - How up-to-date is the data?
6. **Intergrity** - Is the data maintained & updates over time?
7. **Uniqueness** - How little duplication is there in the records?

Accuracy & Validity

Since we don't have direct access to other data sources we check the datas accuracy and validity based on plausibility rule set. Therefore we examine the columns as following

1. values -> sorted by min, max see if range is plausible + singular values -> see whether values are plausible
2. time -> sorted by min, max + singular values -> see whether formatting works right / if it needs to be cleaned / pruned

We used the following query to check those:

VALUES MIN, MAX CHECK

```
SELECT * FROM bronze.temperature ORDER BY values DESC LIMIT 50;
```

-> Check for min max values. (via desc, asc)

On first sight we have seen, that we have extreme outliers for bronze.voc. Where the common voc should be around 50 according to research we found many points lying in the region of around 100 - 300. When examining the data points via interval nesting of the ids we've seen that the outliers aren't actually "outliers" but the voc index is just pretty high in the morning. Therefore we wrote the following query, which looks for the min and max in a one hour interval per day:

```
SELECT time::date as Voc_Date, time_bucket('1 hours', time) as bucket, min(voc) as Voc_Minimum, max(voc) as Voc_Maximum
FROM bronze.voc
WHERE time::date = '2025-08-06'
GROUP BY Voc_Date, bucket
ORDER BY bucket ASC;
```

This produces the following output:

```
  voc_date  |         bucket         | voc_minimum | voc_maximum
------------+------------------------+-------------+-------------
 2025-08-06 | 2025-08-06 00:00:00+02 |         212 |         263
 2025-08-06 | 2025-08-06 01:00:00+02 |         261 |         281
 2025-08-06 | 2025-08-06 02:00:00+02 |         279 |         296
 2025-08-06 | 2025-08-06 03:00:00+02 |         296 |         313
 2025-08-06 | 2025-08-06 04:00:00+02 |         313 |         325
 2025-08-06 | 2025-08-06 05:00:00+02 |         325 |         335
 2025-08-06 | 2025-08-06 06:00:00+02 |         118 |         337
 2025-08-06 | 2025-08-06 07:00:00+02 |          51 |         118
 2025-08-06 | 2025-08-06 08:00:00+02 |          44 |         265
 2025-08-06 | 2025-08-06 09:00:00+02 |          15 |         162
 2025-08-06 | 2025-08-06 10:00:00+02 |          21 |         110
 2025-08-06 | 2025-08-06 11:00:00+02 |          14 |          21
 2025-08-06 | 2025-08-06 12:00:00+02 |           3 |          40
 2025-08-06 | 2025-08-06 13:00:00+02 |           4 |          34
 2025-08-06 | 2025-08-06 14:00:00+02 |          15 |          34
 2025-08-06 | 2025-08-06 15:00:00+02 |          20 |          23
 2025-08-06 | 2025-08-06 16:00:00+02 |          18 |          30
 2025-08-06 | 2025-08-06 17:00:00+02 |          22 |          30
 2025-08-06 | 2025-08-06 18:00:00+02 |          29 |          35
 2025-08-06 | 2025-08-06 19:00:00+02 |          28 |          33
 2025-08-06 | 2025-08-06 20:00:00+02 |          27 |          46
 2025-08-06 | 2025-08-06 21:00:00+02 |          46 |          73
 2025-08-06 | 2025-08-06 22:00:00+02 |          73 |         100
 2025-08-06 | 2025-08-06 23:00:00+02 |         100 |         127
```

The following days show a similar output. BUT on weeknds the voc index stays permanently high, heres a sample of it:

```
  voc_date  |         bucket         | voc_minimum | voc_maximum
------------+------------------------+-------------+-------------
 2025-08-09 | 2025-08-09 00:00:00+02 |         140 |         166
 2025-08-09 | 2025-08-09 01:00:00+02 |         166 |         199
 2025-08-09 | 2025-08-09 02:00:00+02 |         200 |         212
 2025-08-09 | 2025-08-09 03:00:00+02 |         212 |         226
 2025-08-09 | 2025-08-09 04:00:00+02 |         227 |         241
 2025-08-09 | 2025-08-09 05:00:00+02 |         241 |         248
 2025-08-09 | 2025-08-09 06:00:00+02 |         248 |         260
 2025-08-09 | 2025-08-09 07:00:00+02 |         260 |         266
 2025-08-09 | 2025-08-09 08:00:00+02 |         266 |         271
 2025-08-09 | 2025-08-09 09:00:00+02 |         260 |         271
 2025-08-09 | 2025-08-09 10:00:00+02 |         251 |         260
 2025-08-09 | 2025-08-09 11:00:00+02 |         251 |         251
 2025-08-09 | 2025-08-09 12:00:00+02 |         250 |         251
 2025-08-09 | 2025-08-09 13:00:00+02 |         251 |         251
 2025-08-09 | 2025-08-09 14:00:00+02 |         251 |         253
 2025-08-09 | 2025-08-09 15:00:00+02 |         253 |         253
 2025-08-09 | 2025-08-09 16:00:00+02 |         253 |         254
 2025-08-09 | 2025-08-09 17:00:00+02 |         253 |         254
 2025-08-09 | 2025-08-09 18:00:00+02 |         253 |         253
 2025-08-09 | 2025-08-09 19:00:00+02 |         253 |         254
 2025-08-09 | 2025-08-09 20:00:00+02 |         252 |         254
 2025-08-09 | 2025-08-09 21:00:00+02 |         252 |         252
 2025-08-09 | 2025-08-09 22:00:00+02 |         251 |         252
 2025-08-09 | 2025-08-09 23:00:00+02 |         250 |         251
```

We therefore conclude it doesn't have to do with bad data quality (f.e. in a lack of sensor quality), but it seems like the air quality just turns bad, due to reasons like (disabled vents, etc.) => We could talk about that with the facility management.

Except for that no other flaws could be found depending the values.

TIMESTAMP

When examining the timestamp the same way as in the Values check, we see that there multiple timestamp that differ largely in its id of the datetime around 2036-02-07 07:28:46+01. It seems like there is a bug in the timeline which we could take a look at later. It shouldn't be a hurry since those points are really rare and not concurrent.

But still a plausibility check should be introduced, especially since problems are already occuring.

SOLUTION STRATEGY

We introduce plausibility checks for the technical valid ranges that the sensors produce output for. Other than that we filter timestamp for the range of the start date around 05. August and the todays date (since the bug occurs only for the year 2036 this should be filtered out easily)

Our checks would therefore filter as following: - Temperature: 0 - 50 (plausible with some offset) - Humidity: 0 - 100 - Noise: 0 - 1023 - Voc: 0 - 500 - Time: 05. August - NOW()

Completeness & Timeliness & Integrity

Here we check whether there are datapoints missing, and whether that happens often or not.

For that we used the following sql, which calculates the difference of concuring timestamps and filters whether the difference is lower than 45 seconds (15 seconds difference due to timestamp inaccuracies by the arduino)

```
SELECT *
FROM (
    SELECT
        id,
        time,
        LAG(time) OVER (ORDER BY time) AS previous_time,
        EXTRACT(EPOCH FROM time - LAG(time) OVER (ORDER BY time)) AS diff_seconds
    FROM bronze.noise
    WHERE time::date < '2026-01-01'
) AS sub
WHERE diff_seconds > 45;
```

By examining the tables we see that there are time periods in which data points are missing. The time on which the datapoints are missing are equal upon all bronze tables. Since we want to know whether this can expose a risk to our project we count the differences that exceed a threshhold of 7.5 Minutes (since our current plan as of 12.08.25 is to give a user suggestion in 15 min intervalls => small offset such that there may be enough values to aggregate from as fallback)

We conclude that there is only one datapoint with a higher difference, therefore this isn't much of a risk. Due to that we do not plan to propose consequences for the silver layer.

Uniqueness

In order to persist a good data quality it's also necessary to remove duplicates (and unnecessary data). For our use case any information is considered use less, that doesn't add "much" validity to our guidance -> This bases on the insight of a trend (f.e. temperature rises soon, or threshold was reached).

Our final goal is to generate a table in which enough datapoints (with timestamp) for each sensor temperature, humidity, voc and noise for given time intervalls T in order to provide guidance for the next T intervall. Therefore we proposed to start with a sampling rate in the bronze layer, that is as small as reasonable concerning memory, and not as high as that we could not detect when trends start to happen when aggregating data. => Those intervalls can be found in the Arduino part of our project.

Therefore we now have to find out which time intervall T can be aggreggated in order to:

1. Not lose too much information in the data as, that we could not detect trends for the next intervall T
2. Not to choose a time intervall T as that the provided guidance can not be considered useful for the user

Finding the optimal time intervall T

In order to find the optimal time intervall T in which we can aggregate data via it's we propose the following solution:

1. Define multiple time intervalls T, which a user considers useful to get guidance of
2. Set a reasonable threshhold r for VC
3. For each time intervall T calculate it's VC value within => VC will measure how much information will still be contained when calculating the mean for the time intervall T
4. For each time intervall T additionally calculate the average of Cv scores
5. For each time intervall T Calculate the ratio: Count of time buckets of T where VC >= r / Total count of time buckets of T => In order to see whether the average of VC considers also local trends (if this ratio is low, then the average of VC for T was dominated by global trends)

Based on the metrics generated by 3 and 4 we now are able to choose a reasonable time intervall T for each sensor based on evidence found in the data.

This may still remove local trends, since it doesn't consider correlations (like R^2 or r), but since it seems like we don't have enough time for our timeseries forecasting idea, we will just choose this procedure.

This is done via the following sql query:

```
\echo TemperatureVC
WITH
params(win) AS (
  -- Bucket intervals
  SELECT unnest(ARRAY[
    interval '1 min',
    interval '2 min',
    interval '4 min',
    interval '5 min',
    interval '15 min',
    interval '30 min',
    interval '1 hour',
    interval '2 hours'
  ])
),
-- 1) Threshhold r of cv
threshold AS (
  SELECT 0.05::double precision AS r
),
-- Temperature for each bucket of day => Cross Joined to combine all params
base AS (
  SELECT
    p.win,
    date_trunc('day', t.time)               AS day_id, time_bucket(p.win, t.time)              AS bucket,
    t.temperature
  FROM bronze.temperature t
  CROSS JOIN params p
),
-- Mean and Sample std for each bucket
stats AS (
  SELECT
    win,
    day_id,
    bucket,
    AVG(temperature)             AS mean,
    STDDEV_SAMP(temperature)     AS sd
  FROM base
  GROUP BY win, day_id, bucket
),
-- Variational Coefficient for each bcket
vc AS (
  SELECT
    win,
    day_id,
    bucket,
    -- Just to be safe and not divide by 0 (although it should not be possible on our data)
    CASE WHEN mean = 0 THEN NULL ELSE sd / mean END AS vc
  FROM stats
),
per_win_day AS (
  -- Per Bucket and day get avg vc
  SELECT
    v.win,
    v.day_id,
    AVG(v.vc)                                  AS avg_vc,
    COUNT(*) FILTER (WHERE v.vc IS NOT NULL)   AS bucket_count,
    COUNT(*) FILTER (WHERE v.vc >= th.r)       AS high_vc_count,
    (SELECT r FROM threshold)                   AS r_used
  FROM vc v
  CROSS JOIN threshold th
```

```
  GROUP BY v.win, v.day_id
),
per_win_summary AS (
  -- Summary across days per interval (final aggregation level)
  SELECT
    win                                         AS interval_T,
    AVG(avg_vc)                                 AS avg_vc_over_days,
    STDDEV_SAMP(avg_vc)                         AS sd_vc_over_days,
    SUM(bucket_count)                           AS total_bucket_count,
    SUM(high_vc_count)                          AS total_high_vc_count,
    (SUM(high_vc_count)::numeric / NULLIF(SUM(bucket_count), 0))
                                                AS overall_ratio_cv_ge_r,
    MIN(r_used)                                 AS r_used
  FROM per_win_day
  GROUP BY win
)
-- Final output per interval T (summarized over days)
SELECT
  interval_T,
  avg_vc_over_days,
  sd_vc_over_days,
  total_bucket_count,
  total_high_vc_count,
  overall_ratio_cv_ge_r,
  r_used
FROM per_win_summary
ORDER BY avg_vc_over_days ASC, overall_ratio_cv_ge_r ASC;
```

This results in:

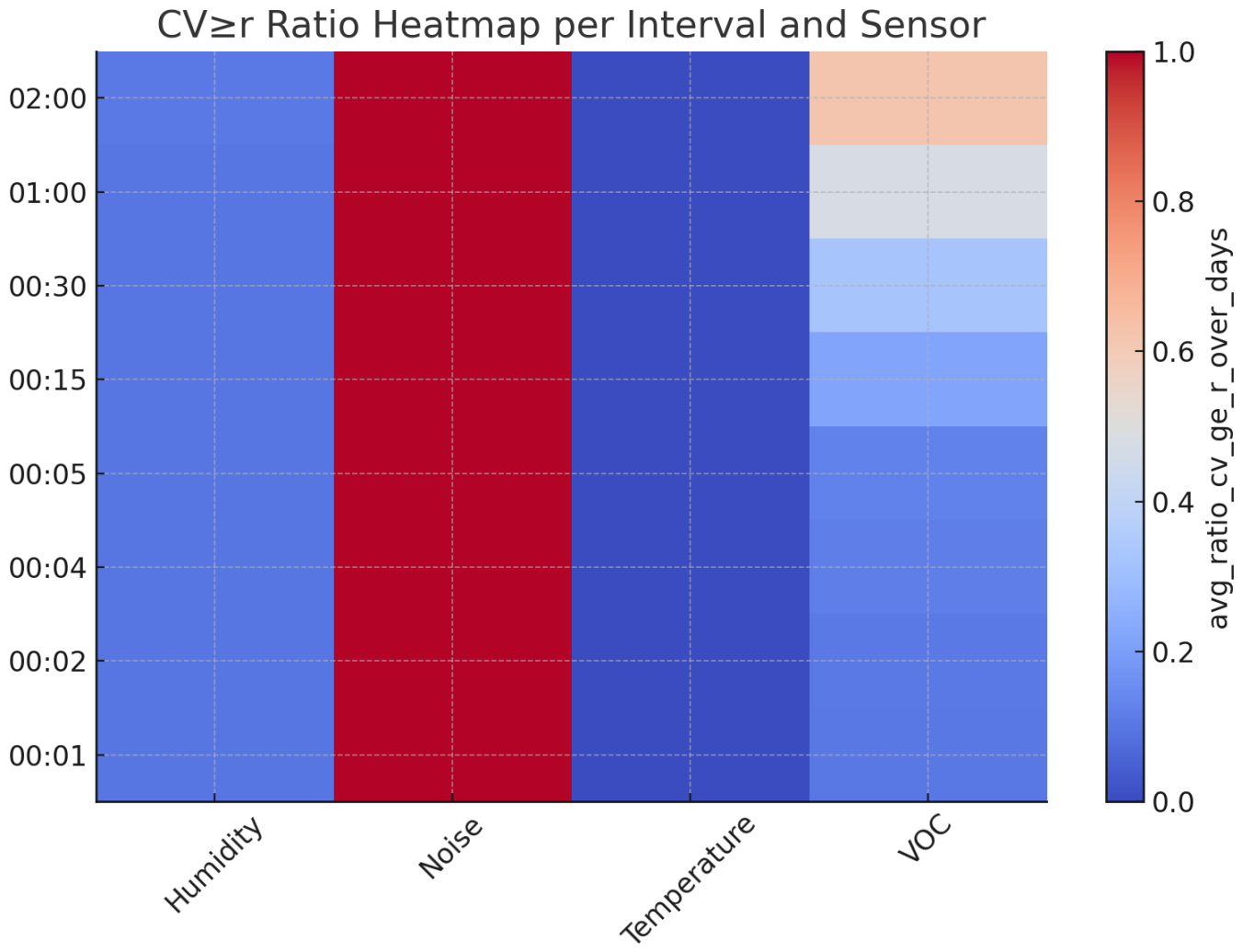
```
TemperatureVC
 interval_t |   avg_vc_over_days    |   sd_vc_over_days    | total_bucket_count | total_high_vc_count |   overall_ratio_cv_ge_r    | r_used
------------+-----------------------+----------------------+--------------------+---------------------+----------------------------+--------
 00:01:00   |  0.004096811605428777 | 0.010881863662589643 |              11136 |                   0 | 0.000000000000000000000000 |   0.05
 00:02:00   |  0.004228549158144857 |  0.01083560662429167 |               5629 |                   0 |     0.0000000000000000000000 |   0.05
 00:04:00   | 0.0043425645574160605 |  0.01075978469604812 |               2821 |                   0 |     0.0000000000000000000000 |   0.05
 00:05:00   |  0.004396210697793138 | 0.010777239148532404 |               2257 |                   0 |     0.0000000000000000000000 |   0.05
 00:15:00   |  0.004845213975577472 | 0.010624895373986249 |                755 |                   0 |     0.0000000000000000000000 |   0.05
 00:30:00   |  0.005486358700811078 |  0.01041348639338682 |                378 |                   0 |     0.0000000000000000000000 |   0.05
 01:00:00   |  0.006830516070406559 | 0.010012541164512388 |                190 |                   0 |     0.0000000000000000000000 |   0.05
 02:00:00   |  0.009348634338048636 | 0.009355182752916011 |                 97 |                   0 |     0.0000000000000000000000 |   0.05
(8 rows)

HumidityVC
 interval_t |    avg_vc_over_days    |              sd_vc_over_days               | total_bucket_count | total_high_vc_count |   overall_ratio_cv_ge_r    |
r_used
------------+------------------------+--------------------------------------------+--------------------+---------------------+----------------------------
+--------
 00:01:00   | 0.01003824050420096598 | 0.027250581851291520023889325253984089320 |              11134 |                   1 | 0.00008981498113885396084 1 |
0.05
 00:02:00   | 0.01063182454735112260 | 0.027058575035657998140239085085590122190 9 |               5629 |                   2 |     0.0003553028957186001 1 |
0.05
 00:04:00   | 0.01116062576020292930 | 0.026890924632994928904419525250817452387 6 |               2821 |                   1 |     0.0003544842254519673 9 |
0.05
 00:05:00   | 0.01130525134983010206 | 0.026842861612627171690205010070283213170 7 |               2257 |                   1 |     0.0004430660168365086 4 |
0.05
 00:15:00   | 0.01231362855079484884 | 0.026509143371882527279186393646274882674 8 |                755 |                   1 |     0.0013245033112582781 5 |
0.05
 00:30:00   | 0.01334250936588425399 | 0.026174683709815454484981870446741770747 4 |                378 |                   1 |     0.0026455026455026455 0 |
0.05
 01:00:00   | 0.01489419972417857723 | 0.025674983083817397075592878219352765096 6 |                190 |                   1 |     0.0052631578947368421 1 |
0.05
 02:00:00   | 0.01748013206572656803 | 0.024943903023237311114441639739825464343 6 |                 97 |                   2 |     0.0206185567010309278 4 |
0.05
(8 rows)

NoiseVC
 interval_t |    avg_vc_over_days    |              sd_vc_over_days               | total_bucket_count | total_high_vc_count | overall_ratio_cv_ge_r  | r_used
------------+------------------------+--------------------------------------------+--------------------+---------------------+------------------------+--------
 00:01:00   | 0.6761409305344279482 6 | 0.039567744015412620947165789017579530383 6 |              11257 |               11257 | 1.0000000000000000000 |   0.05
 00:02:00   | 0.6938922318228556610 1 | 0.041079303911066502025607821787271640245 5 |               5638 |                5638 | 1.0000000000000000000 |   0.05
 00:04:00   | 0.7114979420733563904 5 | 0.048865914678102555952233533025078373552 8 |               2821 |                2821 | 1.0000000000000000000 |   0.05
 00:05:00   | 0.7183535893120495550 | 0.052878620508308018686522539415257635346 91 |               2258 |                2258 | 1.0000000000000000000 |   0.05
 00:15:00   | 0.7514709588422430876 6 | 0.080031754276526100468652237876950020768 6 |                755 |                 755 | 1.0000000000000000000 |   0.05
 00:30:00   | 0.7672446102382251503 6 | 0.091263195122068269290643127762113794764 6 |                378 |                 378 | 1.0000000000000000000 |   0.05
 01:00:00   | 0.7911582525352824028 8 | 0.120845767528698279742961692780310569989 8 |                190 |                 190 | 1.0000000000000000000 |   0.05
 02:00:00   | 0.8226182316649990638 7 | 0.177671862992381204980233656064628911672 8 |                 97 |                  97 | 1.0000000000000000000 |   0.05
(8 rows)

VocVC
 interval_t |    avg_vc_over_days    |              sd_vc_over_days               | total_bucket_count | total_high_vc_count | overall_ratio_cv_ge_r  | r_used
------------+------------------------+--------------------------------------------+--------------------+---------------------+------------------------+--------
 00:01:00   | 0.0654980167247811407 1 | 0.199828065773891726485676637234172640755 8 |              11257 |                  34 | 0.0030203428977525095 5 |   0.05
 00:02:00   | 0.0672386231499717865 4 | 0.199235118647176027671652592656011596698 7 |               5638 |                  50 | 0.0088683930471798510 1 |   0.05
 00:04:00   | 0.0704827168047930104 2 | 0.198155947434339151720145225318139571992 6 |               2821 |                  66 | 0.0233959588798298475 7 |   0.05
 00:05:00   | 0.0721791050484297116 8 | 0.197613240683165834106587963923170077387 5 |               2258 |                  71 | 0.0314437555358724535 0 |   0.05
 00:15:00   | 0.0871663884909880339 3 | 0.193272682125997731272261172033311973956 0 |                755 |                  99 | 0.1311258278145695364 2 |   0.05
 00:30:00   | 0.1070689749508484247 1 | 0.188932354266726189319578849418910627624 8 |                378 |                  89 | 0.2354497354497354497 4 |   0.05
 01:00:00   | 0.1424128424350533347 9 | 0.182581632028338292888271696450977433734 3 |                190 |                  76 | 0.4000000000000000000 0 |   0.05
 02:00:00   | 0.1931183368445866948 5 | 0.178871347252653211312353363908407658407 7 |                 97 |                  55 | 0.5670103092783505154 6 |   0.05
(8 rows)
```

INTERPRETATION



The heatmap shows a unified view across all intervalls and their corresponding varational coefficient (generated by ChatGPT based on the above values). The following interpretation also considers the other columns shown before, but for a better visibility the heatmap was added.

Our conclusions regarding the final intervall size is as follows:

1. **Temperature, Humidity** - The analysis shows, that in the current room the temperature and humidity seems to stay constant across the whole day => due to AC. Therefore we cannot propose a reasonable intervall. In order to still implement the use case we propose a intervall of **15 minutes** based on our experiences in other buildings, just in case we gather sensor data from rooms without ac (old DHBW building f.e.)
2. **Noise** - Due to a high variational coefficient and low std it seems like we should take a small intervall in order to not remove local trends across the intervalls (this gets worse with bigger intervalls) -> Therefore we should take a intervall of **30 seconds**. Therefore we still have enough data to give guidance for bigger (or smaller) intervalls => f.e. we could penalize noise even on small intervalls during exams in comparison to times when no exam is written.
3. **Voc** - Since the voc sensor can reach peaks pretty quick due to its physical properties (also seen by std) we even consider a r > 0.05 as acceptable. Therefore we propose a interval of **5 minutes** due to high statistical noise as proposed through std on higher intervalls, and its bucket cv ratio that increases significantly on a **15 minute ratio**

Definition

We define the silver layer for each sensor as following:

1. **A Continuous Aggregate that shows realtime data (via materialized_only = false)** - Refresh Policy: Update last 30 days till now() - 15 minutes all 15 minutes
2. **Columns: time, arduino_id, avg_temperature_at_15m, stddev_temperature_at_15m** - differs based on each bucket interval chosen and for each sensor

4.3.3 Gold

The purpose of this layer is to create a layer with data that is solely used by our business logic within the UI. Since our UI only needs numerical representation of each sensors taken data point by time, we can define the gold layer as a passthrough view of our silver layer. Therefore we do not duplicate any data regarding memory, and we have a unified structure (naming conventions of each column) of each view. Therefore we stay with a good extensibility in the case that we do want to add more complex business logic / data strucure => f.e. replacing the views with materialized views.

Since our recommendation algorithm is based on checking intervals of values per timebucket we only need the following columns of each sensor.

1. **bucket_time** - Starting time of each time interval
2. **arduino_id** - Arduino from which the data point comes from
3. **avg_value_in_bucket** - Average of data points within the sensors time bucket

Don't forget: The first time_bucket describes the ongoing time intervall that is updated each time you query the view

For example if it's currently 10:46 and we query the gold.temperature view, then the first data point averages the ongoing time interval of (10:45 till NOW()). The second time bucket would be the full interval starting from (10:30 till 10:45).

5. 5. Data Collection

5.1 Sampling-Rate and -Strategy

This chapter discusses our selection of sampling rates and it's decision way. We do this by examining which sensors we have and what could be a minimum / maximum reasonable sampling time for our use case. Then we calculate the data amount for our project period of around 60 days in order to finally decide our sampling rate / strategy.

This chapter defines the sampling rate that each hardware uses to fetch sensor data AND the transmission rate on which those points are sent to the MQTT Broker

5.1.1 Reasonable transmission rate by Sensor

We identified the sound sensor as most critical sensor regarding sampling time, due to two reasons:

1. The sensor measures sound by a sine wave, therefore we must first aggregate multiple values in order to calculate the sound level (multiply points at 20hz sampling are already enough based on public information)
2. In order to generate a proper recommendation regarding the sound level of a room multiple datapoints are needed because only a constantly high sound level is relevant regarding study quality.

Therefore our transmission rate can't be set below 1/20s (Reason 1) and should be high enough in order to measure a constant sound level of a room instead of some spikes (plane, door slams, pitches, etc.)

In order to comply with those requirements we discussed to have a final transmission rate between the intervall of **[1s; 5min]**.

The other sensors definitely do not a data rate in this intervall. Those could be even higher (since our use case does not need such high resolution data -> goal: recommendations) therefore we do not focus on those anymore.

5.1.2 Calculation of data volume in database by point

Our goal is to have raw datapoints within the database. We do not want to lose information if we don't have to especially due to the machine learning use case. Therefore all reasonable statistical calculation shall be performed in a second table (called silver). Therefore if we have a transmission rate of f.e. 30 seconds for the MQTT Broker, each sent message contains a vector of data points (collected in intervalls of sampling rate of each sensor). The timestamp then refers only to the moment on which the mqtt message was sent from each arduino to the broker. By that we can extrapolate each timestamp of the collected sensor data if needed.

In order to select a final transmission rate we calculated the following data volumes (lines in table) for each transmission rate and time period of 60 days (project run time)

1. ~86k -> 1 min
2. ~172k -> 30s
3. ~345k -> 15s
4. ~860k -> 10s

Based on the fact that none of the data volumes is generally considered high, we chose a rate of 30s. It lies on the lower intervall of the critical sound sensor, therefore we should have enough data to aggregate a final usable value from. Since the data volume only amounts to 172k it also shouldn't result in any problem to send (perhaps unnecessary) high resolution data from the other sensors as well.

5.1.3 Sampling Rate

Within those 30 seconds we sample each datapoint as follows:

1. Sound Sensor: 1s -> Vector[30]
2. Humidity: 30s -> Vector[1]
3. Temperature: 30s -> Vector[1]
4. VOC: 5s -> Vector[6]

Later we aggregate those data within the database / rules in order to reduce outliers. Our first priority is to ensure no loss of information.

6. 6. Non Functional Requirements Methodology

6.1 Backup Strategy

This document will describe how we choose to backup our database in case of unexpected crashes or unrecoverable states.

6.1.1 Expectation from the Backup Strategy

1. No interference with current dataflow
2. Everything is inside the backup
3. The data backup is written automatically via a cronjob or systemd-unit
4. The data restoring is performed manually or with a manually exectuable script
5. A Backup with a timestamp is created every 24H

6.2 Implementation

To meet the expectations and criterias, which are needed in order to have a successful backup strategy, there are two ways to solve this
1. cronjob
2. Systemd-Unit

Although systemd units are more complex to configure, they offer more options for controlling them. In theory a systemd-unit would also be suitable for backing up the data.
Nevertheless in the scope of the project it is not possible to run those systemd-units, even in user-space, because of lack of privilege. Therefore a cronjob is used in order to run the backup bashscript. The entire backup process produces a GZip file as a result, which contains the database dump. The restore process is **not** automatic.
To ensure that the data is not only stored on the server's hard disk in the event of a total failure, the dumps are also archived on individual devices of the members of this project.

6.3 Configuring a Cronjob

- The backup.sh is triggered every 24h at 00:00.
- The uptime.sh is triggered every minute.
- Edit a cronjob with crontab -e

```
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# m h  dom mon dow   command
* * * * * bash /home/$USER/uptime.sh
0 0 * * * bash /home/$USER/backup.sh
```

6.4 NFR testability

There are three Quality Goals which need to be tested.

6.4.1 NFR 1.1

The criteria from the NFR 1.1 states that the measured sensor data must reach the bronze layer of the database within 5 minutes. The silver and gold layers are based on the bronze layer, therefore only a few calculations are needed for the bronze data to be viewable inside of the those layers. Therefore the time, which is needed for the the data transfer from bronze to silver and from silver to gold is negligible. This is the reasoning, why the time measurement between sensor and bronze is accurate enough to evaluate the NFR 1.1.

For the established measurement scope, the validity of the data transfertime with the limit of 5 minutes can be evaluated now. A function in the frontend takes care of the validation of this NFR. The function should take the timestamp of the sensor data and creates a comparison between the current timestamp across multiple sensortypes. The NFR 1.1 is valid, if all comparisons result in a time difference below 5 minutes. The frontend should also display the result of the comparisons.

6.4.2 NFR 1.2

To measure the quality goal for the NFR 1.2, a script is used to determine the uptime of the database. In order to calculate the uptime a log of the uptime is needed. The Uptime Monitoring is already covered. Therefore the already exsitent CSV-File needs to be evaluated, such that an uptime for the last week can be calculated. To achieve this, a python script is used to read the CSV-File and relatively calculate a weekly uptime based on this file.

6.4.3 NFR 3.1

The NFR 3.1 is a criterea for accessibility. The Quantification of a accessibility criteria is usually difficult. There isn't any existing data or collectable logs for the quantification for this non functional requirement.

What comes closest to this, is to test, if there are any inputfields, which require keyboard inputs are existing in the UI, which can be automatically tested. Since the existence of an input field does not rule out operability using only the mouse a usertest, which is usual best practice for UI-Testing and validation, is additionally required.

Thats the reason why this project is also relying on usertests to quantify the NFR 3.1.

Usertest:

The usertest relies on the following metrics. The measurement needs to be adjusted before the usertest to the current UI.

Metric	How to Measure
Task completion rate	Percentage of users able to complete key tasks (e.g., navigate to x, ...) **without using a keyboard**.
Time to complete tasks	Average time users take to finish a task with mouse/trackpad alone.
Error rate	Number of times a user fails or misclicks because a function isn't/is harder accessible without keyboard.

NAVIGATION:

- Open the dashboard start page.
- Switch to a specific room.

FILTERING / PARAMETERS

- Display the graph in a specific time range (last 30 days).

CONTENT INTERACTION

- Expand or collapse a section/card.
- Open a detail view (e.g., click on a specific data item).

6.5 Uptime monitoring

In order to get the uptime of the database, a cronjob can be used. Since a cronjob is already in use by the backup strategy of the database, it is straight forward to setup up another one. The Quality Goals state that the uptime needs to be measured every minute the system is running. Therefore the cronjob will run a script that creates an entry into a CSV-File every minute. This entry contains the start Unix timestamp of the database and the current timestamp, when the start timestamp was requested. The start Unix timestamp of the database is retrieved via a request to the database, since postgres has an internal timer, which keeps track of the startup timestamp.

If the database is not reachable because of unexpected failure, the script will write the default value zero into the CSV-File.

7. 7. Software Engineering

7.1 Requirements Engineering

7.1.1 Problem Description

Problem Description Often, students in classrooms felt that they had difficulty learning.

Problem Statement Worksheet

- **Who does the problem affect?:** Students and Lecturers (indirectly)
- **What does the problem affect? What are the factors of the problem?:** Concentration problems => Impaired learning ability (**Drivers from student interviews:** Temperature, stuffiness => Air quality (CO2, humidity), noise level)
- **Why is solving the problem important?:** To optimally achieve the goal of studying (learning).
- **Where does the problem lie or where does it have influence:** The problem particularly lies in the fact that poor room quality only becomes a burden/noticeable during prolonged use => Especially problematic for in-person studies, like at DHBW
- **When did the problem start? When does it need to be solved:** Since the beginning of studies => But unevenly (e.g., temperature in summer)
- **How was the problem created? How can it be solved:** The problem naturally occurs in closed rooms (especially with many people) => Actively counteract it (ventilation, notes on volume, etc.) => **Changing rooms, etc., is not a solution due to the effort involved**

7.1.2 Stakeholders

1. Students => Have an interest in room quality (and thus in improvement through the system to be developed)
2. Developers => Have an interest in project success
3. Lecturers => Have an interest in room quality to ensure optimal teaching
4. Study Program Organization Team => Has an interest in improving room quality to enable optimal teaching
5. Partner Company (Student) => Has an interest in the best possible education for its students

---

7.1.3 Goals

1. Improve the learning environment
2. Cause identification => Continuous improvement of the environment

---

7.1.4 Use Cases

1. The system is used during lectures / exams in rooms to continuously improve the learning environment.
2. The system is used to retrospectively identify problems.

7.1.5 Functional Requirements

Monitoring

1. The system must collect room quality data (temperature, CO2 content of the air, humidity, noise level) at regular intervals and make it retrievable.

Analysis / Interpretability

1. The system must analyze the collected data and provide current recommendations for action to improve the learning environment.
2. The system must visually prepare the data for better readability and interpretability.

Operation

1. The system must provide the interpretable data and analyses within the DHBW Digital Campus.

7.1.6 Non-Functional Requirements

Monitoring

1. The system must make the cyclically collected room quality data (and recommendations) retrievable within 5 minutes of collection.
2. The system must persistently store the collected room quality data.
3. The system must provide data of high quality regarding comprehensibility (unit), timeliness (see NFR 1), appropriateness, and correctness.

Analysis / Interpretability

1. The system may only issue truly beneficial (with regard to room climate) recommendations for action.
2. The system must clearly distinguish current data from past data.

Operation

1. The system must be fully operable without the use of a keyboard.

7.2 Proposed Solutions

In principle: Sensor data must be collected and persisted (project requirement)

7.2.1 Improvement of Room Climate

Automated Actions

Based on the collected data, actuators should be triggered here, for example, to improve the room climate => Window openers, traffic light for noise recommendations, etc.

Dashboard

Based on the collected data, recommendations for action as well as current data should be displayed here. This can be accessed from the Digital Campus.

Regular Room Change with Breaks

Rooms should be regularly rotated with ventilation => Climate improvement. The noise level should be reduced by means of breaks.

Since our team does not have any influence on the management of rooms within DHBW this solution is not applicable

7.2.2 Monitoring of Room Quality Data

Dashboard

The collected data is displayed within a web dashboard and retrieved via frontend elements (without text input => NFR).

Reporting in Filesystem

The collected data is regularly transferred to a filesystem in a suitable file format (e.g., PDF for tables, diagrams, etc.).

7.3 Selection of Solution

The selection of a final solution was developed by a focus-area matrix. We propose 4 important (development focused) categories that are graded (rough estimation 0-5) for each possible solution. The categories are independent from the declared requirements, since all possible solutions are capable of implementing the specified requirements.

Category	Dashboard (Improvement of Climate)	Automated Actions	Dashboard (Monitoring)	Reporting in Filesystem
Technical uncomplexity	4.5	0	3	5
Extensibility, Maintainablity	4.5	1	4.5	1
Duration of Implementation (shorter = higher)	4	2	3	4
UX	4	5	5	0
Final Grade	**4.25**	**2**	**3.88**	**2.5**

The focus-area matrix recommends the usage of a full dashboard approach for each monitoring and recommendation specific problem solutions.

(Translation German -> English by Google Gemini 2.5 Flash (15.07.2025))

7.4 SFMEA Project start

7.4.1 Components - Functinalities of the System

- Arduino
- Subscription script
- Database
- UI -> Work in progress, because of LRM

7.4.2 Classification

1. Deployment
2. Runtime Failures

3. Design/Conceptional Mistakes

ID	Component	Failure Mode	Failure mode	Cause of Failure	Effect of Failure
FA1	Arduino	Data Missing	Loss of function	Power loss, sensor hardware error	End effect: Database, UI, Subscriber script
FA2	Arduino	Data Inaccurate	Incorrect function	Power loss, Sensor inaccuracy	End effect: Database, UI, Subscriber script
FA3	Arduino	Data Timeless	Erroneous function	Power loss, NTP Server not reachable	End effect: Database, UI, Subscriber script
FA4	Arduino	Erroneous/ Inconsistent Datapoints	Loss of function	Power loss, Unexpected Environmental influences	End effect: Database, UI, Subscriber script
FA5	Arduino	Data cannot be transferred	Erroneous function	Power loss, Server not reachable	End effect: Database, UI, Subscriber script
FA6	Subscription Script	Data cannot be received (from MQTT-Server)	Loss of function	Arduino down	End effect: Database, UI
FA7	Subscription Script	Data cannot be transferred (to the database)	Erroneous function	Database connection error, Database down	End effect: Database, UI
FA8	Database	Not available (permanent)	Loss of function	DHBW-Server down/crashed	End effect: Database, UI, Subscriber script
FA9	Database	Not available (temporary)	Loss of function	Restart / Maintenance / Overload	End effect: Database, UI, Subscriber script
FA10	Database	Faulty data cleaning	Incorrect function	Incorrect scripting code	End effect: Database, UI, Subscriber script
FA11	Database	Reading not possible	Incorrect function	Query issue / Permission issue	End effect: Database, UI, Subscriber script
FA12	Database	Writing not possible	Incorrect function	Disk full / Permission issue	End effect: Database, UI, Subscriber script

7.4.3 Risk and Criticality

1. (S)everity can be rated in a scope of 1 (No effect) - 10 (Severe System Failure)
2. (O)ccurrence can be rated in a likelyhood of 1 (Failure unlikely) - 10 (Failure is almost inevitable)

3. (D)etectability can be rated in a scope of detectable from 1 (certain to be detected) - 10 (Not likely to be detected)
   Risk priority number (RPN) = S * O * D, The higher the RPN, the more critical the failure mode.

ID	Effect of Failure	Severity	Occurrence	Detection	RPN (S×O×D)
FA1	End effect: Database, UI, Subscriber script	8	5	4	160
FA2	End effect: Database, UI, Subscriber script	7	6	5	210
FA3	End effect: Database, UI, Subscriber script	6	4	6	144
FA4	End effect: Database, UI, Subscriber script	7	5	5	175
FA5	End effect: Database, UI, Subscriber script	8	4	6	192
FA6	End effect: Database, UI	9	3	7	189
FA7	End effect: Database, UI	9	4	6	216
FA8	End effect: Database, UI, Subscriber script	10	2	2	40
FA9	End effect: Database, UI, Subscriber script	7	5	6	210
FA10	End effect: Database, UI, Subscriber script	6	4	5	120
FA11	End effect: Database, UI, Subscriber script	9	3	3	81
FA12	End effect: Database, UI, Subscriber script	9	4	3	108

7.4.4 Detection means

- EVIDENT: The failure is readily detected during operation.
- DORMANT: The failure can be detected when maintenance is performed.
- HIDDEN: The failure is not detected unless intentionally sought, for instance, by testing the system.

ID	Component	Failure Mode	Detection measure
FA1	Arduino	Data Missing	Dormant
FA2	Arduino	Data Inaccurate	Dormant
FA3	Arduino	Data Timeless	Dormant
FA4	Arduino	Erroneous/Inconsistent Datapoints	Dormant
FA5	Arduino	Data cannot be transferred	Dormant
FA6	Subscription Script	Data cannot be received (from MQTT-Server)	Hidden
FA7	Subscription Script	Data cannot be transferred (to the database)	Hidden
FA8	Database	Not available (permanent)	Evident
FA9	Database	Not available (temporary)	Evident
FA10	Database	Faulty data cleaning	Evident
FA11	Database	Reading not possible	Evident
FA12	Database	Writing not possible	Evident

7.4.5 Corrective Actions

ID	Priority	Component	Failure Mode	RPN	Suggested Corrective Action
FA7	**High**	Subscription Script	Data cannot be transferred (to the database)	216	Implement retry logic, use a message queue or buffer in case of failure
FA2	**High**	Arduino	Data Inaccurate	210	Calibrate sensors regularly; add data validation checks
FA9	**High**	Database	Not available (temporary)	210	Introduce local caching; add reconnect and retry strategies
FA5	**Medium**	Arduino	Data cannot be transferred	192	Add ACK checks; implement timeout and retransmission logic
FA6	**Medium**	Subscription Script	Data cannot be received (from MQTT-Server)	189	Auto-reconnect on failure; improve server health monitoring
FA4	**Medium**	Arduino	Erroneous/ Inconsistent Datapoints	175	Include timestamp validation, sequence number and anomaly detection logic
FA10	**Monitor**	Database	Faulty data cleaning	120	Strengthen QA/test coverage; use data integrity checks
FA12	**Monitor**	Database	Writing not possible	108	Introduce write verification; enable rollback and logging

7.4.6 Summary

This FMEA analyzes the failure modes in a sensor-to-database pipeline involving Arduino hardware, MQTT-based data transmission, and a backend database.

Components analyzed include the Arduino sensor board, subscription script (MQTT), and the central database. The analysis focuses on data integrity, availability, and flow consistency.

The highest RPN (216) was found in the Subscription Script, failing to transfer data to the database.

Arduino inaccuracies and temporary database unavailability also scored critical RPNs of 210. Detection was often rated poor for transient errors and silent data corruption. Each failure mode was rated by Severity (S), Occurrence (O), and Detection (D), each on a scale from 1–10. The Risk Priority Number (RPN) was calculated as RPN = S × O × D. Detection types were classified as Evident, Dormant, or Hidden.

7.4.7 List actions for top risks

- Improve retry and queuing logic in the Subscription Script.
- Calibrate and validate Arduino sensor input more frequently.
- Implement local caching and robust reconnect logic for database access.

If implemented, these corrective actions are expected to reduce RPN scores by improving detection and reducing occurrence likelihood. This contributes to a more robust and certifiable data pipeline.

7.5 SFMEA Project End

7.5.1 Components - Functinalities of the System

- Arduino
- Subscription script
- Database
- UI

7.5.2 Classification

1. Deployment
2. Runtime Failures

3. Design/Conceptional Mistakes

ID	Component	Failure Mode	Failure Effect	Cause of Failure	Effectlocation
FA1	Arduino	Data Missing	Loss of function	Power loss, sensor hardware error	End effect: Database, UI, Subscriber script
FA2	Arduino	Data Inaccurate	Incorrect function	Power loss, Sensor inaccuracy	End effect: Database, UI, Subscriber script
FA3	Arduino	Data Timeless	Erroneous function	Power loss, NTP Server not reachable	End effect: Database, UI, Subscriber script
FA4	Arduino	Erroneous/ Inconsistent Datapoints	Loss of function	Power loss, Unexpected Environmental influences	End effect: Database, UI, Subscriber script
FA5	Arduino	Data cannot be transferred	Erroneous function	Power loss, Server not reachable	End effect: Database, UI, Subscriber script
FA6	Subscription Script	Data cannot be received (from MQTT-Server)	Loss of function	Arduino down, Invalid Network Configuration (Bridge)	End effect: Database, UI
FA7	Subscription Script	Data cannot be transferred (to the database)	Erroneous function	Database connection error, Invalid network configuration, Database down	End effect: Database, UI
FA8	Database	Not available (permanent)	Loss of function	DHBW-Server down/crashed	End effect: Database, UI, Subscriber script
FA9	Database	Not available (temporary)	Loss of function	Restart / Maintenance / Overload / DHBW-Server out of memory	End effect: Database, UI, Subscriber script
FA10	Database	Faulty data cleaning	Incorrect function	Incorrect (materialized) View definition, Data manipulation outside of job time window (60d)	End effect: Database, UI

ID	Component	Failure Mode	Failure Effect	Cause of Failure	Effectlocation
FA11	Database	Reading not possible	Incorrect function	Query issue / Permission issue / DHBW-Server out of memory	End effect: Database, UI, Subscriber script
FA12	Database	Writing not possible	Incorrect function	Disk full / Permission issue / DHBW-Server out of memory	End effect: Database, UI, Subscriber script

7.5.3 Risk and Criticality

1. (S)everity can be rated in a scope of 1 (No effect) - 10 (Severe System Failure)
2. (O)ccurrence can be rated in a likelyhood of 1 (Failure unlikely) - 10 (Failure is almost inevitable)
3. (D)etectability can be rated in a scope of detectable from 1 (certain to be detected) - 10 (Not likely to be detected)

Risk priority number (RPN) = S * O * D, The higher the RPN, the more critical the failure mode.

ID	Effect of Failure	Severity	Occurrence	Detection	RPN (S×O×D)
FA1	End effect: Database, UI, Subscriber script	8	5	4	160
FA2	End effect: Database, UI, Subscriber script	7	6	5	210
FA3	End effect: Database, UI, Subscriber script	6	2	6	72
FA4	End effect: Database, UI, Subscriber script	7	4	1	28
FA5	End effect: Database, UI, Subscriber script	8	4	6	192
FA6	End effect: Database, UI	9	3	3	81
FA7	End effect: Database, UI	6	4	4	96
FA8	End effect: Database, UI, Subscriber script	10	2	2	40
FA9	End effect: Database, UI, Subscriber script	7	5	3	105
FA10	End effect: Database, UI, Subscriber script	3	4	5	60
FA11	End effect: Database, UI, Subscriber script	9	3	3	81
FA12	End effect: Database, UI, Subscriber script	9	4	3	108

7.5.4 Detection means

- EVIDENT: The failure is readily detected during operation.
- DORMANT: The failure can be detected when maintenance is performed.
- HIDDEN: The failure is not detected unless intentionally sought, for instance, by testing the system.

ID	Component	Failure Mode	Detection measure
FA1	Arduino	Data Missing	Dormant
FA2	Arduino	Data Inaccurate	Dormant
FA3	Arduino	Data Timeless	Dormant
FA4	Arduino	Erroneous/Inconsistent Datapoints	Dormant
FA5	Arduino	Data cannot be transferred	Dormant
FA6	Subscription Script	Data cannot be received (from MQTT-Server)	Evident
FA7	Subscription Script	Data cannot be transferred (to the database)	Evident
FA8	Database	Not available (permanent)	Evident
FA9	Database	Not available (temporary)	Evident
FA10	Database	Faulty data cleaning	Dormant
FA11	Database	Reading not possible	Evident
FA12	Database	Writing not possible	Evident

7.5.5 Corrective Actions

This only shows the changes since last time since the project is discontinued after this sprint!

ID	Old Priority => New Priority	Component	Failure Mode	RPN	Suggested Corrective Action
FA7	**High => Monitor**	Subscription Script	Data cannot be transferred (to the database)	96	use a message queue or buffer in case of failure
FA2	**High**	Arduino	Data Inaccurate	210	Calibrate sensors regularly; add data validation checks
FA9	**High => Medium**	Database	Not available (temporary)	105	Introduce local caching
FA5	**Medium**	Arduino	Data cannot be transferred	192	Add ACK checks; implement timeout and retransmission logic
FA6	**Medium => Monitor**	Subscription Script	Data cannot be received (from MQTT-Server)	81	Add active alert (f.e. via email) to increase even increase detectibility more
FA4	**Medium => Low**	Arduino	Erroneous/ Inconsistent Datapoints	28	Include timestamp validation, sequence number and anomaly detection logic
FA10	**Monitor => Monitor (but basically irrelevant)**	Database	Faulty data cleaning	60	Strengthen QA/test coverage; use data integrity checks
FA12	**Monitor**	Database	Writing not possible	108	Introduce write verification; enable rollback and logging

7.5.6 Summary

This FMEA analyzes the failure modes in a sensor-to-database pipeline involving Arduino hardware, MQTT-based data transmission, and a backend database.

Components analyzed include the Arduino sensor board, subscription script (MQTT), and the central database. The analysis focuses on data integrity, availability, and flow consistency.

The highest RPN (216) was found in the Subscription Script, failing to transfer data to the database.

Arduino inaccuracies and temporary database unavailability also scored critical RPNs of 210. Detection was often rated poor for transient errors and silent data corruption. Each failure mode was rated by Severity (S), Occurrence (O), and Detection (D), each on a scale from 1–10. The Risk Priority Number (RPN) was calculated as RPN = S × O × D. Detection types were classified as Evident, Dormant, or Hidden.

7.5.7 List actions for top risks

- Improve retry and queuing logic in the Subscription Script.
- Calibrate and validate Arduino sensor input more frequently.
- Implement local caching and robust reconnect logic for database access.

If implemented, these corrective actions are expected to reduce RPN scores by improving detection and reducing occurrence likelihood. This contributes to a more robust and certifiable data pipeline.

8. 8. Tests

8.1 UI Usability Test (Mouse Only)

This document lists all mouse-only test cases for the current UI.
Goal: Every feature must be operable without using a keyboard.

Test ID	Scenario	Steps (mouse only)	Expected Result	Status
TC-01	Expand side menu	Click on the arrow icon to expand/collapse the side menu	Menu expands/ collapses correctly	✅
TC-02	Select room/ overview	In the side menu, click on a room or on “Overview”	Correct page is displayed, active item highlighted	✅
TC-03	Select sensor	In a room view, open the dropdown and click a sensor parameter	Graph updates to selected parameter	✅
TC-04	Select past days	Open the dropdown for time selection and choose number of past days	Graph updates to chosen time range	✅
TC-05	Fullscreen/Save plot	Click fullscreen button OR “Save as PNG” button	Graph is shown fullscreen OR PNG is saved locally	✅
TC-06	Adjust Y-axis range	Drag min/max handles of the axis range bar with the mouse	Y-axis range updates accordingly	✅
TC-07	Adjust X-axis range	Drag handles of the time range bar with the mouse	X-axis range updates accordingly	✅
TC-08	Graph interactions	Use zoom-in, save as picture, pan, reset axes (via toolbar buttons)	Graph responds correctly to each action	✅
TC-09	Enlarge selected zone	Drag to mark a zone on the graph, then release	Marked zone is zoomed into / enlarged view shown	✅
TC-10	Open overview page	From the side menu, click “Overview”	Overview page is displayed with summary data	✅

9. 9. Alerting

9.1 Alerting & Monitoring (Subscription Script)

This document describes the runtime alerting features implemented in the **subscription_script** component. Its goal is to detect data ingestion problems early and notify operators via e-mail.

9.1.1 1. Overview

The subscription script ingests sensor data from MQTT and writes it to TimescaleDB. Two failure classes are monitored:

1. Inactivity (no MQTT messages for a configurable time window)
2. Sequence anomalies (skipped or repeated sequence numbers per sensor topic)

Notification channel: SMTP e-mail (one alert + one recovery per inactivity window, rate-limited alerts for sequence anomalies).

9.1.2 2. Features

- Per-topic sequence tracking (`temp` , `hum` , `co2` /VOC, `mic`)
- Automatic baseline detection for the first observed sequence (prevents false positives when the consumer starts late)
- Inactivity watchdog thread with a threshold (default 300s unless overridden by env)
- Inactivity alert only once per outage, plus an explicit recovery e-mail when data resumes
- Cooldown-based sequence anomaly alerting (per topic key)
- Graceful failure (alert sending errors never crash ingestion)
- Support for implicit SSL SMTP (port 465) and STARTTLS (e.g. port 587)

9.1.3 3. Sequence Monitoring

Each sensor category maintains its last seen sequence number. On every new message: 1. If last stored sequence is 0 (initial), the current value becomes the baseline (info log only) 2. Else the expected number is `last + 1` 3. If the received number differs → warning log + sequence alert (subject: "MQTT Sequence Anomaly")

This helps detect dropped messages or publisher restarts that reset counters unexpectedly.

Limitations

- Baseline resets on process restart (no persistent state). For strict global continuity you would need external storage.
- Mixed ordering across topics is not correlated (each topic family is independent).

9.1.4 4. Inactivity Watchdog

A background thread records the timestamp of the last received MQTT message (even if payload JSON is malformed). Steps: 1. Every second the thread checks `now - last_message_ts` . 2. If the difference exceeds the threshold (`WATCHDOG_INACTIVITY_THRESHOLD_SECONDS`) and no active alert → send Inactivity Alert (one time) and mark outage active. 3. When messages resume (difference below threshold) and outage was active → send Recovery Alert and clear state.

Rationale

Avoids spam during prolonged outages and gives a clear start/end signal.

9.1.5 5. Configuration (Environment Variables)

Variable	Description	Default
`WATCHDOG_INACTIVITY_THRESHOLD_SECONDS`	Seconds without any MQTT message before inactivity alert	300 (example)
`ALERT_COOLDOWN_SECONDS`	Cooldown for sequence anomaly alerts per topic key	300
`SMTP_HOST`	SMTP server hostname	(required for e-mail)
`SMTP_PORT`	SMTP server port (465 implicit SSL, 587 STARTTLS)	587
`SMTP_USER`	SMTP auth username	optional
`SMTP_PASS`	SMTP auth password	optional
`ALERT_EMAIL_FROM`	From address (fallback: `SMTP_USER`)	derived
`ALERT_EMAIL_TO`	Comma separated recipient list	(required)

Example (.env excerpt)

```
SMTP_HOST=mail.example.org
SMTP_PORT=465
SMTP_USER=alerts@example.org
SMTP_PASS=secret
ALERT_EMAIL_FROM=alerts@example.org
ALERT_EMAIL_TO=ops@example.org,dev@example.org
WATCHDOG_INACTIVITY_THRESHOLD_SECONDS=300
ALERT_COOLDOWN_SECONDS=600
```

9.1.6 6. E-mail Flow

Scenario	Mail(s) Sent	Notes
Inactivity begins	1x Inactivity Alert	Only once until recovery
Continued inactivity	none	Outage flagged active
Data resumes	1x Recovery Alert	Resets state
Sequence anomaly	Sequence Anomaly Alert	Subject key per topic, respects cooldown

9.1.7 7. Error Handling

- SMTP errors are logged with level ERROR; ingestion continues.
- If TLS negotiation fails for STARTTLS, code falls back to plain unless mode is strictly SSL (implicit via port 465).
- Missing config (no recipients or host) logs a single warning and disables sending.

10. 10. Software Quality Workshop

10.1 Current State & Weakness Analysis

Goal: Provide a structured snapshot of product & process quality, highlight gaps against ISO/IEC 25010 and internal quality goals, and form a prioritized baseline for improvements.

10.1.1 Reference Frameworks

- ISO/IEC 25010 (Product quality & Quality in use)
- Internal quality goals (arc42 “Introduction and Goals”)
- SFMEA (initial & extended) as risk driver
- Deployment & operational artifacts (Docker, monitoring, backup)
- Delivery process (Definition of Done, tests, boards)

10.1.2 Evaluated ISO 25010 Characteristics (selection & relevance)

ISO 25010 Attribute	Rationale	Current Maturity	Primary Evidence
Functional Suitability	Correct acquisition & presentation of room climate data	Medium	Functional req. FR.1.x, UI pages
Reliability (Availability, Fault Tolerance)	Pipeline must tolerate short outages	Low–Medium (manual aspects)	Missing active alerting, simple retry patterns
Performance Efficiency	Data should reach gold layer in <5 min	Medium	NFR.1.1, no E2E latency metric captured
Compatibility	Components integrate via MQTT / DB	High (simple coupling)	Clear interfaces (topics, tables)
Usability (Accessibility)	Keyboard-free operation (NFR.3.1)	Partial (rudimentary tests)	`frontend/tests/accessibility/test_accessibility.py` (limited)
Security	Sensor data + internal infra	Basic measures, no hardening doc	Missing threat model / auth layer
Maintainability	Change & test effort	Medium (modular, naming inconsistencies)	Mixed naming, some missing tests
Portability (Deployability)	Rapid environment setup	Improved (unified compose planned)	Previously multiple separate compose files

10.1.3 Internal Quality Goals vs. Reality

Quality Goal (arc42)	Linked Failure Modes (SFMEA)	Current State	Gap
Data Quality (NFR. 1.1)	FA7, FA9, FA5, FA6, FA12	Partially addressed, no end-to-end lead time tracking	Missing measurability
DB Availability 95% (NFR.1.2)	FA9, FA12	Uptime script exists, no escalation	No alerting / trend report
Accessibility (NFR.3.1)	(None)	Not evidenced	Missing verification & test cases

10.1.4 Artifact Consistency (Board / Docs / Code)

Aspect	Observation	Risk
SFMEA → corrective actions	Not always referenced in commits	Erosion of traceability
Backup concept	Restore path undocumented	False sense of safety
Mock data	Outdated vs. schema	Misleading test assumptions
Frontend error handling	Now systematic (panels, fallbacks)	Previously white screen / raw trace
Deployment	Manual multi-step	Onboarding latency, human error
Definition of Done	Unclear / not enforced	Quality variance per story

10.1.5 Key Weakness Shortlist

1. Missing active alerting (Arduino offline / data flow stall)
2. Outdated mock data → inconsistent test runs
3. No documented restore procedure (despite backup script)
4. Incomplete process definition (DoD not institutionalized)
5. No end-to-end data latency metric (Sensor → Gold layer)
6. Limited Arduino test coverage (edge/error cases)
7. Insufficient accessibility verification (NFR.3.1)

10.1.6 Detailed Weakness Analysis

Alerting gap

- Symptom: No active notification when MQTT messages stop.
- Effect: Delayed detection → risk of prolonged data gaps.
- Reference: SFMEA FA1, FA5, FA6 (detection often dormant/hidden).
- Impact: High (prevents proactive response), Effort: Medium.

Inconsistent mock data

- Symptom: Test data does not reflect schema evolution.
- Effect: Wrong UI assumptions, skewed performance perception.
- Impact: Medium, Effort: Low–Medium.

Missing restore documentation

- Symptom: Only backup path covered.
- Effect: Backups potentially unusable (“restore gap”).
- Impact: Medium (longer recovery), Effort: Low.

Fuzzy Definition of Done

- Symptom: No consistent quality gate.
- Effect: Variable merge quality, reduced traceability.
- Impact: Medium, Effort: Low.

Missing latency metric

- Symptom: No measurement from send timestamp to gold availability.
- Effect: NFR.1.1 unverifiable.
- Impact: High (audit), Effort: Medium.

Arduino test coverage

- Symptom: Edge / error cases missing.
- Effect: Possible silent failures.
- Impact: Medium, Effort: Medium–High.

Accessibility validation

- Symptom: Only rudimentary test (absence of input widgets); no comprehensive audit (focus order, ARIA, contrast).
- Evidence: `frontend/tests/accessibility/test_accessibility.py` scope is limited.
- Effect: NFR.3.1 only partially supported; hidden barriers possible.
- Impact: Low–Medium (dashboard complexity limited), Effort: Medium (automatable + manual checklist).
- Improvement ideas:
- Add automated axe-core / Pa11y checks via Playwright
- Manual heuristic checklist (contrast, focus indicator, semantics)
- Clarify explicitly optional vs. required keyboard interaction.

10.1.7 Prioritized optimization focus areas

ID	Topic	Selection Rationale	In current scope?
A1	Mock data update	Fast quality lever, reduces misinterpretation	Yes
A2	Docker compose deployment	Boosts reproducibility & onboarding	Yes
A3	Frontend error handling	Immediate UX / resilience uplift	Yes
A4	Arduino offline alerting	Critical missing detection	Yes (new feature)
A5	Restore documentation	Important but secondary	Later
A6	DoD sharpening	Process quality iteration	Later
A7	Latency metric	Higher technical effort	Later

10.1.8 Metric proposals

Goal	Proposed Metric	Capture Method
Data latency	P95 end-to-end (Arduino ts → gold row ts)	Log correlation + Timescale query
DB availability	% minutes health check passes	Uptime script aggregation
UI fault robustness	# uncaught exceptions / session	Logging + test harness
Mock data quality	Schema diff count = 0	Automated schema vs. sample diff
Deployment reproducibility	Onboarding time → first dashboard	Self report + timer
Alerting effectiveness	Mean Time To Detect (MTTD)	Simulated outage timestamp diff

10.1.9 Summary

Base functionality & modular structure are solid. Largest gaps: (1) missing active monitoring (alerting), (2) representative / validated mock data, (3) unified deployment & consistent UI resilience. Selected optimizations directly improve risk reduction, transparency and perceived quality.

10.2 Extended SFMEA & Comparison

This chapter augments and reflects the original SFMEA versions (project start vs. extended state). Focus: consistency, RPN evolution, remaining gaps (especially detection) and derived additional mitigations (notably alerting & data validation).

10.2.1 Overview comparison

Aspect	Initial SFMEA	Extended SFMEA	Observation
Component scope	Arduino, subscription, DB (UI WIP)	UI added	Full system view achieved
Peak RPN	216 (FA7)	210 (FA2 still high, FA7 reduced)	Some mitigation effect (FA7)
Detection classification	Many dormant/hidden (transfer chain)	Partially evident (subscription)	No active outage notification
Mitigation focus	Retry, calibration, caching	Priority reductions listed	Effect only partly evidenced

10.2.2 Notable issues & plausibility

- FA2 (sensor inaccuracy) remains high → no documented calibration or drift detection.
- FA6/FA7 detection shift (Hidden→Evident) not fully justified (missing monitoring/log artefacts).
- FA7 RPN drop (216→96) seems aggressive without evidence of persistent buffering.
- Missing explicit "Arduino offline detection" mode (now added as FA13) – previously implicit across FA1/FA5.

10.2.3 Added / refined failure modes

New ID	Component	Failure Mode	Cause	Effect	Remark
FA13	Monitoring / alerting	Data stream outage unnoticed	No heartbeat / timeout	Delayed response, data gap	Separates detection aspect of FA1/ FA5
FA14	Data quality pipeline	Drift undetected	Sensor aging	Gradual incorrect insights	Adds temporal dimension to FA2
FA15	Process	Incomplete merge (missing tests/ docs)	Fuzzy DoD	Introduces defects	Process risk spanning multiple modes

10.2.4 New / extended scoring (excerpt)

ID	S	O	D	RPN	Rationale (detection focus)
FA13	8	4	8	256	Highest RPN – no active detection (purely reactive)
FA14	6	5	7	210	Drift is hard to detect without routine
FA15	5	5	6	150	Process gap drives quality fluctuation

This shifts the critical hotspot clearly to FA13 (alerting gap) – justifying the new feature.

10.2.5 Mitigation matrix (old vs. new)

Failure Mode	Existing Control	Gap	Proposed Addition
FA2	Generic "calibration"	No interval / drift metric	Quarterly calibration + median profile plausibility
FA5	ACK / retry	No sequence supervision	Sequence number + backoff + offline counter
FA7	Claimed queue/ buffer	No artefact evidence	Persistent replay buffer (local SQLite)
FA9	Caching & reconnect	No timeout metrics	Health endpoint + Prometheus counter
FA13	(—)	No timeout defined	Heartbeat interval + offline mail alert
FA14	(—)	No drift metric	Rolling z-score / IQR outlier stats
FA15	(—)	DoD not enforced	Checklist gate in PR template

10.2.6 RPN reduction prognosis after alerting

Mode	Base RPN	Measure	Expected change (D or O)	Target RPN
FA13	256	Heartbeat topic + 5-min timeout + mail alert	D: 8→3	96
FA5	192	Sequence + retransmit	D: 6→4	128
FA2	210	Quarterly calibration + drift stats	O:6→4, D:5→4	112

10.2.7 Alerting feature priority rationale

- Highest newly identified RPN (FA13)
- Low implementation complexity (timeout + email)

- Improves detection across multiple modes (FA1, FA5, FA6)
- Enables measurable MTTD metric

10.2.8 Recommended quality assurance hooks

Layer	Mechanism	Purpose
MQTT ingestion	Last timestamp cache	Offline detection
Subscription script	Sequence validator	Loss / reordering visibility
DB layer	Write result counter	Early persistence failure signal
Frontend	Staleness badge	Transparency of data freshness
Process	PR checklist	DoD enforcement

10.2.9 Summary

Original SFMEA covers key data flow risks but systematically underestimates detection and process risks. Extension with FA13–FA15 sharpens prioritization. The alerting feature addresses structural blindness and shifts the risk apex. Follow-up: drift detection (FA14) and process hardening (FA15).

10.3 Documentation of Implemented Optimizations

This chapter evidences three implemented optimizations: objectives, actions, current state, and (where feasible) qualitative / potential quantitative impact vs. the previous state.

Optimizations in scope: 1. Update & process clarification for mock data 2. Unified deployment via single `docker-compose.yml` 3. Frontend error handling & resilience layer

10.3.1 Mock Data Update

The existing SQL script `database/utils/fill_dummy.sql` was updated (fields / periods) but still: no automated generator, no schema diff check, no parameterization.

Aspect	Before (outdated)	Now (current)	Target (planned / recommended)	Value if achieved
Script freshness	Outdated	Updated	Continuous drift check	Reliable test basis
Generation process	Manual	Manual	Parameterized generator	Reproducibility
Quality assurance	None	None	Automated schema diff	Early drift detection
Data realism	Random	Random	Patterns (day/night, peaks)	More valid UI/perf tests
Reproducibility	No	No	Seed control	Comparability
Data volume	Fixed 60 days	Fixed 60 days	Configurable (days)	Faster local runs

Implemented action (actually done)

- Update of existing SQL script (generates 60 days random historical data for multiple sensors/devices).

Recommended follow-ups

- Separate Python generator + schema diff.
- Parameterization (duration, device count, pattern profiles).
- CI drift check.

Metric ideas (not implemented)

Metric	Target	Planned capture
Schema diff count	0	`schema_compare.py`
Dummy rows per sensor	≥24h coverage	Generator log

Rationale

More realistic data lowers false positives in UI tests & improves performance assessments.

10.3.2 Deployment via Docker Compose

Aspect	Before	After	Benefit
Startup complexity	Multiple manual steps (DB, MQTT, services)	Single `docker compose up`	Faster onboarding
Environment consistency	Divergent local setups	Unified defined services	Fewer “works on my machine” cases
Documentation	Scattered	Central in compose + README	Transparency
Scaling tests	Cumbersome	Profiles / overrides feasible	Faster experiments

Action

- Created unified `docker-compose.yml` (TimescaleDB, MQTT broker, subscription script, frontend, monitoring (optional)).
- Standardized `.env` variables + `example.env`.

Before / After indicators

Indicator	Before (estimate)	After (target)
Onboarding time to first dashboard	60 min	<10 min
Manual steps	>6	1

Rationale

Reduces process variability; baseline for QA & production likeness.

10.3.3 Frontend Error Handling & Resilience

Implemented: DB health check, safe selection (empty list fallback), basic error panel, staleness checking functions. Not (or only partially) implemented: consistent info/warning differentiation across all paths, success toasts, external API specific fallbacks, circuit breaker.

Aspect	Before	Current	Target	Value if achieved
DB error handling	Crash / trace	Error panel + empty list	Add retry / circuit	Higher robustness
Error panel consistency	Ad-hoc / inconsistent	Basic panel `render_error_panel`	Unified severity mapping	User clarity
Staleness transparency	None	Functions exist (partially used)	Unified warning display	Freshness awareness
External API errors	Raw exception	No specific handling	Info panel + degradation status	Reduced frustration
Success signal (toast)	None	Not present	One-time toast on first data	Positive feedback
Telemetry / observability	None	None	Sentry / OTel integration	Faster root cause analysis
Test coverage error paths	Low	Partial (selection tests)	Full (stale, empty, offline, API)	Resilience evidence

Implemented principles

- “Fail soft” for DB access (empty list fallback)
- Early health check before deep UI rendering
- Staleness check helpers present

Not yet implemented (target principles)

- Unified color / severity matrix across all flows
- Success toasts / differentiated soft-fail panels
- API specific resilience layer

Metric ideas (target – not currently measured)

Metric	Before	Target	Measurement (planned)
Uncaught UI exceptions / 100 sessions	n/a	<1	Log parser / telemetry
First-data success feedback	n/a	>90% sessions once	Session state
Error type differentiation (hard/soft/info)	None	Fully covered	Code review / UI snapshot

Rationale

Prevents white-screen situations; increases trust & resilience without overwhelming users.

10.3.4 Impact summary

Optimization	Primary ISO 25010 attributes	Current (qualitative)	Target (quantifiable)
Mock data	Maintainability (minor), Functional suitability	Script updated, still manual & random	0 schema diffs / sprint, parameterizable
Docker compose	Portability, Reliability	Unified startup (if adopted)	<10 min setup time
Error handling	Reliability, Usability	DB failures & staleness partially cushioned	<1 uncaught exception / 100 sessions

10.4 Optimization: Mock Data Update

10.4.1 Current state (real)

A manually executed SQL script `database/utils/fill_dummy.sql` uses `generate_series` to insert random / synthetic measurements (noise, temperature, humidity, voc) for ~10 days into bronze tables.

History: - Script was outdated (structure / quantity misaligned) and got updated (fields & distributions aligned). - NO automated generation or validation (no Python generator, no schema diff) – ad hoc execution only.

Current script scope: | Table | Generation logic | Frequency | IDs distributed | |-------|------------------|-----------|----------------| | bronze.noise | Random 0–1028 | Every 1 min (2 offsets) over 10 days | 401/402/403 | | bronze.temperature | Random 5–30 | Every 30s over 10 days | Round-robin 401–403 | | bronze.humidity | Random 5–30 | Every 30s over 10 days | Round-robin 401–403 | | bronze.voc | Random 5–30 | Every 30s over 10 days | Round-robin 401–403 |

Limitations (current): - No semantic patterns (day/night, peaks, ventilation events) - No schema verification pre-insert (assumes tables present) - No parameterization (duration / device count) - No separation dev vs. test datasets

10.4.2 Target goals (future – not implemented)

Goal	Description	Suggested metric
Schema conformity	Automatic column/type diff	0 schema diffs / run
Pattern realism	Simulated day/night & outliers	Qualitative review / defined variance
Reproducibility	Parameterized generator w/ seed	Fixed seed → same distribution
Automation	CI job can generate sample	Successful pipeline run

10.4.3 Recommended future actions (not implemented)

1. Python generator (`/tools/mockgen/`) with params: `--rooms` , `--duration-days` , `--pattern day-night|flat` , `--seed` .
2. Schema introspection via `information_schema.columns` → drift warning.
3. Advanced patterns (sinus temperature, CO2 peaks, humidity drift, noise spikes).
4. Validation: type + min/max constraints or abort.
5. Optional nightly CI sample + diff report.

10.4.4 Risks & mitigation (current vs. recommended)

Risk	Current state	Mitigation
Schema drift	Manual noticing at insert	Automated diff check
Unrealistic distribution	Pure random	Patterned generators
Non-reproducibility	Every run differs	Seed control
Data explosion	Fixed high volume (10 days)	Parametrize duration

10.4.5 Extensions (future)

- Sample anonymized production data + hybrid mock
- Statistical distribution checks (KS test vs. reference)
- Outlier injection flags for negative tests

10.4.6 Evidence (performed change)

Aspect	Before	Now
Script freshness	Outdated	Updated (matches current schema)
Automation	None	Still none
Pattern realism	Simple random	Unchanged
Parameterization	None	Unchanged

10.4.7 Potential future value

Goal	Expected benefit
Validation checks	Early drift detection
Realistic patterns	More valid perf & UI tests
Reproducibility	Comparable builds
Parametrized duration	Faster local datasets

10.5 Optimization: Unified Deployment (Docker Compose)

10.5.1 Initial situation

Deployment required multiple separate compose files / invocations (DB, frontend, subscription script, MQTT). Consequences: - High cognitive load (ordering / dependencies) - Inconsistent environment variable configuration - Higher error risk (forgotten flags / services) - Longer onboarding for new developers

10.5.2 Goals

Goal	Description	KPI
One-command start	Entire system via one command	`docker compose up`
Unified env management	Shared `.env` / service `example.env`	Complete documentation
Health transparency	Basic healthchecks integrated	All core containers `healthy`
Faster onboarding	Reduced time to functioning dashboard	<10 min

10.5.3 Target architecture (services)

Service	Role	Key dependencies
timescaledb	Sensor / aggregate persistence	Data volume, network
mqtt-broker	Message transport	Port 1883
subscription	Consumes MQTT → writes DB	Depends on MQTT + DB
frontend	Dashboard / UI	DB (read), optional monitoring
monitoring (optional)	Uptime / metrics	DB reachability

10.5.4 Actions

1. Merge distributed compose files → single root file.
2. Conventions: network aliases `db`, `mqtt`.
3. Healthchecks for DB & optional frontend HTTP.
4. Unified env variable naming (`DB_HOST`, `MQTT_HOST`).
5. Document start/stop/logs in README.
6. Optional overrides for dev mounts / hot reload.

10.5.5 Before / After comparison

Criterion	Before	After
Startup steps	>6 manual commands	1 (compose up)
Ordering errors (DB not ready)	Frequent	Minimized
Documentation overhead	High	Reduced
New dev onboarding	45–60 min	<10 min

10.5.6 Risks & mitigation

Risk	Description	Mitigation
Over-compose (monolith)	Harder differentiated deployments	Optional profiles / flags
Image version drift	Uncoordinated tag updates	Renovate / Dependabot
Credential leaks	`.env` accidentally committed	`.gitignore` + example file

10.5.7 Extensions (future)

- Prod-specific compose (readonly volumes, stricter networks)
- Prometheus / Grafana integration
- Reverse proxy (nginx/traefik) for TLS / routing

10.5.8 KPIs (post implementation)

KPI	Target	Measurement
Onboarding time	<10 min	Self report + timer
Failed start rate (first 3 attempts)	<10%	Onboarding log
Healthcheck success rate	>99% local start	Compose logs

10.6 Optimization: Frontend Resilience & Error Handling

10.6.1 Objectives

Provide a user-facing dashboard that fails gracefully, communicates state clearly, and degrades predictably during partial outages.

10.6.2 Principles

Principle	Application
Fail fast & visibly	Detect backend/mqtt unavailability quickly and show status banners
Progressive enhancement	Core data panels render even if advanced widgets fail
Observable failures	Structured logging + browser console clarity
User guidance	Provide actionable retry / reload suggestions

10.6.3 Error classes

Class	Example	UX Strategy
Network timeout	DB query exceeds threshold	Skeleton + retry button
MQTT disconnect	Broker unreachable	Banner warning, stale badge
Data schema drift	Missing expected field	Soft fallback + log warning
Aggregation delay	Gold materialization late	Timestamp indicator + stale label
Auth / permission (future)	Unauthorized dashboard area	Redirect + message

10.6.4 Patterns Implemented

1. Central fetch wrapper with timeout + error normalization.
2. Stale data indicators (last update timestamp, color coding).
3. Component-level suspense (loading skeletons for charts / tables).
4. Banner system (stackable global alerts: WARN, ERROR, INFO).
5. MQTT connection monitor (state machine: connecting → active → degraded → offline).
6. Graceful degradation path: charts → numeric backups → placeholder.

10.6.5 Example state model (MQTT)

```
[connecting] --success--> [active]
[connecting] --timeout--> [degraded]
[active] --heartbeat_missing--> [degraded]
[degraded] --reconnect_success--> [active]
[degraded] --timeout--> [offline]
[offline] --manual_retry--> [connecting]
```

10.6.6 UI Signals

Signal	Meaning	Location
Green dot	Live stream	Header status block
Amber triangle	Degraded (stale > threshold)	Header + panel footer
Red hollow circle	Offline (no updates)	Header + panel overlay
Grey clock	Historical render (no live)	Panel footer

10.6.7 Logging taxonomy

Level	Usage
debug	Connection attempts, retries
info	Successful reconnect, resubscription
warn	Stale data threshold exceeded
error	Unhandled fetch failure, JSON parse issue

10.6.8 Testing Matrix

Scenario	Method	Expected
Broker down at load	Simulate refused TCP	Offline banner + retry
Late gold aggregate	Delay materialized view	Stale label appears
Random JSON field drop	Modify mock API	Fallback path triggered + warn log
Flaky reconnect	Inject intermittent failures	State oscillates degraded/active

10.6.9 Future Enhancements

- Client-side circuit breaker for failing endpoints
- Telemetry export (OpenTelemetry browser spans)
- Offline caching (IndexedDB) for last-known values
- Role-based feature flags (hide sensitive panels)

10.6.10 KPIs

KPI	Target
Time to visible error (network)	< 2s
False positive stale warnings	< 3%
Successful auto-reconnect ratio	> 90%
User forced refresh frequency	< 1 per hour

10.7 New Feature: Arduino Offline Alerting

Goal: Early detection of pipeline outages or stalls (Arduino → MQTT → subscription → DB) via active alerting for missing or inconsistent messages.

10.7.1 Problem & motivation

Initial (pre-implementation): No active notification when MQTT measurements stop or sequence errors occur → reactive recognition (empty UI / latency until human notice).

Current implemented state: - Inactivity watchdog thread in `subscription_script/main.py` tracks time since last message (`get_last_message_timestamp`). - Email alert on exceeding `WATCHDOG_INACTIVITY_THRESHOLD_SECONDS`, plus recovery mail when flow resumes. - Sequence anomaly alert (`send_sequence_alert`) with separate cooldown keying (`seq:{topic}`). - Cooldown & force logic in `alerting.py` core `send_alert_email`.

NOT implemented yet: - Persistent status table (`sensor_status`), event history (`alert_events`). - UI status badges / user threshold & recipient configuration. - Explicit heartbeat messages (currently inferred via normal data flow).

Impacted failure modes (extended): FA1, FA5, FA6, FA13 (new), secondary FA7.

10.7.2 Scope (current vs. target)

Aspect	Current implementation	Target vision
Inactivity detection	Time since last MQTT msg (watchdog threshold)	Add persistent status + MTTD metric
Sequence monitoring	Email on gap anomaly	Correlate w/ persistent sequence history / gap count
Delivery channel	Email only	Extend (webhook / Slack)
Configuration	Env vars (`ALERT_*`, `WATCHDOG_*`)	UI form + DB persistence
Persistence	None (in-memory cooldown registry only)	`sensor_status`, `alert_events` tables
UI visualization	None	Badges (online / offline / degraded)
Heartbeat	Implicit (data flow)	Explicit dedicated topic (optional)

10.7.3 Actual implemented flow (subscription script)

1. MQTT message arrival updates last timestamp & sequence state (`on_message`).
2. Background thread `_watchdog_loop` (1s interval) evaluates inactivity duration.
3. If threshold exceeded → `send_inactivity_alert` once; sets `_inactivity_alert_active`.
4. When flow resumes → `send_inactivity_recovery_alert`; flag reset.
5. Sequence deviation (expected vs. received) → `send_sequence_alert` (per-topic cooldown).
6. SMTP misconfiguration / missing recipients → single warning, no crash.

No persistent storage of state transitions – only logs & emails.

10.7.4 Target architecture (future extensions)

1. Persistent status table (`sensor_status`): `room_id`, `last_seq`, `last_timestamp`, `state`.
2. Event table (`alert_events`): history including `from_state`, `to_state`, `created_at`.
3. UI badge + history view + configuration surface (threshold / recipients / gap tolerance).

4. Optional heartbeat messages to reduce false positives during natural idle windows.

10.7.5 Component overview (current vs. target)

Component	Current role	Target (extended) role
Arduino	Emits measurements (implicit heartbeat)	Optional explicit heartbeat
Subscription script	Watchdog + email sending	Add persistence & metrics export
DB	Store measurements	Add status + event history
Frontend	(No alert UI)	Config + status display
Mailer	SMTP delivery	Multi-channel abstraction

10.7.6 State logic (target model – not implemented)

Condition	State
now - last_timestamp <= threshold AND no seq gap	online
now - last_timestamp > threshold	offline
now - last_timestamp <= threshold AND seq gap detected	degraded

Current implementation: binary inactivity + recovery (no explicit degraded state).

10.7.7 Acceptance criteria (current vs. target)

Criterion	Current	Target
Offline detection	Inactivity > threshold → alert	Keep + persistence
Sequence gap detection	Email alert (no state category)	Degraded state + event
One alert per outage	Implemented (flag)	+ Persistent outage ID
Configurability	Env variables	UI + DB settings
Persistent history	None	`alert_events` table
MTTD measurable	Only indirect (logs)	Prometheus metric export

10.7.8 Metrics (target – not implemented)

Metric	Target
Mean Time To Detect (MTTD)	< 2× threshold
False positives / week	≤ 1
Recovery recognition time	< 1 min

10.7.9 Risks & mitigation

Risk	Description	Countermeasure
Flapping	Frequent state toggles	Hysteresis (2 consecutive checks)
Mail failure	SMTP unreachable	Retry + log + status flag
Sequence reset	Arduino reboot lowers seq	Detect reset (seq smaller & fresh timestamp)
Clock drift	Unsynced clock	Ensure NTP / server time source

10.7.10 Extensions (backlog)

- Multi-channel alerts (Slack / webhook)
- Aggregated daily reports
- Drift detection (FA14) / quality degradation observability
- Prometheus exporter for status & counts
- Persistent state machine + UI badges

10.7.11 RPN assessment (current contribution vs. target)

Failure Mode	Base RPN	Current implementation contribution	Limitation	Target model potential
FA13 Offline not detected	256	Inactivity alert reduces detection delay	No persistence / metric	Further reduction via state persistence + MTTD tracking
FA5 Data not transferred	192	Sequence alerts surface gaps	No correlation with DB write failures	Combine with write result counters
FA6 Reception disturbed	189	Indirect detection (inactivity)	No cause differentiation	Cause-specific classification (broker vs. publisher)

10.7.12 Summary

Implemented: minimal but effective core (inactivity + recovery + sequence anomaly alerting) without persistence & UI integration. Original broader vision now clearly marked as future. Next steps: persistent status tracking, UI visualization, metric export.

10.8 Slim Quality Checklist

Scope: Rapid engineering health snapshot (implementation-focused, excludes long-term roadmap items).

10.8.1 1. Documentation

Item	Status	Notes
Chapter 10 fully in English	✅	Filenames + content aligned
Alerting scope realism (no overclaim)	✅	Future items clearly scoped
PDF build (chapter-only)	✅	mkdocs-pdf config independent

10.8.2 2. Observability & Reliability

Item	Status	Notes
Inactivity watchdog email	✅	Recovery mail included
Sequence anomaly email	✅	Cooldown logic present
State persistence	❌	Planned (sensor_status)
Metrics export (MTTD etc.)	❌	Not yet

10.8.3 3. Deployment

Item	Status	Notes
Unified compose workflow	✅	One command startup
Healthchecks (core services)	⚠️	Partial, extend DB + frontend
Version pinning	⚠️	Review tags policy

10.8.4 4. Frontend Resilience

Item	Status	Notes
Stale data indicator	✅	Timestamp + label
Connection state machine	✅	Active/Degraded/Offline model
Graceful degradation path	✅	Fallback hierarchy defined
Circuit breaker	❌	Future enhancement

10.8.5 5. Testing & Quality

Item	Status	Notes
JSON message tests	✅	Covered in test suite
Alert logic unit tests	⚠️	Partial coverage (cooldowns)
Link integrity (chapter 10)	⚠️	Run after filename swap
Automation for PDF	❌	Manual trigger currently

10.8.6 6. Risk / SFMEA Evolution

Item	Status	Notes
New failure modes documented	✅	FA13 added
RPN recalculation transparency	✅	Extended table
Mitigation alignment	✅	Consistent with actual code

10.8.7 7. Backlog Focus (Top 5)

Priority	Item	Rationale
1	Persist alert states/events	Enable history + metrics
2	Metrics export (Prometheus)	Quantify detection performance
3	Healthcheck completion	Earlier failure surfacing
4	PDF build automation	Consistency & CI artifact
5	Frontend circuit breaker	Isolate repeated failures

10.8.8 8. KPI Snapshot (Current Feasible)

KPI	Current	Target
Inactivity detection latency	≈ threshold	< 1.2× threshold
False inactivity alerts	Low (manual observation)	< 5%
Reconnect success ratio	n/a	> 90%

10.8.9 9. Immediate Next Steps

1. Update nav to English filenames only.
2. Remove legacy German-named duplicates (post link check).
3. Add link integrity validation (script / CI step).
4. Decide on stable cover logo approach (optional fallback).

Lean checklist maintained to reflect only implemented or imminent engineering concerns.