



Università degli Studi di Cagliari

Project Report

for the subject

Internet of Things

Topic:

Molt Prevention System

Student:

Jonas Ohmäscher

Matricola:

IA/65882

Professor:

Prof. Michele Nitti

Date:

02.02.2025

Table of Contents

| | |
|--|-----------|
| 1. Introduction | 3 |
| 1.1 How to prevent mold from growing | 3 |
| 1.2 Relative Humidity and Absolut Humidity | 3 |
| 2 Analysis | 5 |
| 2.1 Scenario Definition | 5 |
| 2.2 Competitors | 5 |
| 2.2.1 Xiaomi "Mi Temperature and Humidity Monitor 2" | 5 |
| 2.2.2 Bosch Smart Home "Room thermostat II" | 6 |
| 2.2.3 KNX | 7 |
| 2.3 Functional Requirements | 7 |
| 2.4 Non-functional requirements | 8 |
| 2.5 Hardware | 9 |
| 2.5.1 Humidity and Temperature Monitoring | 10 |
| 2.5.2 Remote Window Opening | 11 |
| 2.6 Communication | 12 |
| 2.6.1 HTTP Request (REST API) | 12 |
| 2.6.2 WebSocket | 13 |
| 2.6.3 MQTT Protocol | 13 |
| 3 Design | 16 |
| 3.1 Virtualization Layer | 17 |
| 3.2 Services Layer | 18 |
| 3.2.1 Fetch Weather Service | 18 |
| 3.2.2 Humidity Comparison Service | 18 |
| 3.2.3 User Notification Service | 18 |
| 3.3 Digital Twin Layer | 19 |
| 3.4 Application Layer | 20 |
| 3.4.1 Database APIs | 20 |
| 3.4.2 Telegram APIs | 21 |
| 3.4.3 MQTT-Handling | 21 |
| 4 Prototype | 24 |
| 4.1 Measuring Device | 24 |
| 4.2 Ventilation Device | 24 |
| 5 Conclusion | 26 |
| 6 Literature | 27 |

1. Introduction

This project aims to develop a comprehensive IoT system to monitor and analyze indoor and outdoor environmental conditions in real time. By utilizing a network of sensors to measure temperature and humidity, the system calculates absolute humidity and recommends the best times for ventilation based on outdoor weather data. Users can register their houses, configure rooms, and receive timely notifications through a user-friendly interface, ensuring efficient moisture control and healthier living environments.

With the integration of digital twins to represent houses and their rooms, the system also leverages geolocation data to fetch precise weather information for external comparisons. This feature ensures accurate recommendations tailored to the unique environmental context of each user.

The report details the design, implementation, and evaluation of this IoT-based mold prevention system, highlighting its functional and non-functional requirements, technical architecture, and real-world applications. By bridging smart technology with practical problem-solving, this project showcases how IoT can be effectively applied to improve indoor living conditions while promoting energy-efficient practices.

1.1 How to prevent mold from growing

Moisture is the most critical factor for mold development. It can result from various sources, such as water leaks, condensation on windows or walls, or high humidity levels. Indoor relative humidity above 60% creates an ideal environment for mold. A poor ventilation allows mold to grow. This often happens in spaces with limited airflow, such as basements, bathrooms and closed rooms.

Key strategies that prevent mold from growing include managing the indoor humidity level below 60% and ensuring proper ventilation. Also the timing for the ventilation to happen is important. If air becomes warmer, it can contain a higher amount of water. So for example if during ventilation hot air comes inside and it's cooling down there, it can contain less water. This causes water to condense, which results in mold. To prevent this, it could be useful to compare the humidity levels of indoor and outdoor air.

1.2 Relative Humidity and Absolut Humidity

Two key measures of humidity are relative humidity (RH) and absolute humidity (AH). Relative humidity (RH) represents the amount of water vapor in the air compared to the maximum amount the air can hold at a given temperature. It is expressed as a

percentage. For example, if the air contains half of its moisture-holding capacity, the relative humidity is 50%. However, this measurement is highly temperature-dependent. Warmer air can hold more water vapor, so RH increases as the air cools, even if no additional moisture is introduced. For this reason, RH is helpful in understanding how "humid" the air feels but does not provide an accurate picture of the actual moisture content when comparing environments with different temperatures. Absolute humidity (AH), on the other hand, measures the actual quantity of water vapor in the air, expressed in grams of water vapor per cubic meter of air (g/m³). Unlike RH, AH is not influenced by temperature changes, making it a more reliable metric for assessing the actual moisture content of the air. For example, if the absolute humidity indoors is 10 g/m³ and the absolute humidity outdoors is 7 g/m³, it would be effective to ventilate the room to reduce indoor moisture. The formula to calculate absolute humidity (AH) from temperature (T) and relative humidity (RH) is written down in (1). [1]

$$AH = \frac{6.11 * e^{\frac{17.67 * T}{243.5 + T}} * RH * 2.1674}{273.15 + T} \quad \left(\frac{g}{m^3} \right) \quad (1)$$

The key difference between these two measurements lies in their dependency on temperature. RH varies with temperature and provides a relative indication of how saturated the air feels, whereas AH remains constant regardless of temperature and directly reflects the water vapor present in the air.

For mold prevention, both metrics are important. High relative humidity, particularly above 60%, creates conditions conducive to mold growth, especially in warm spaces. However, comparing absolute humidity levels between indoor and outdoor environments is essential to determine when ventilation is effective.

2 Analysis

This part contains a definition of the given scenario and an analysis of the functional and non-functional requirements.

2.1 Scenario Definition

Mold formation is a common problem in homes with poor ventilation, especially during cold and damp seasons. Excessive indoor humidity contributes to mold growth, which can damage property and pose health risks.

By monitoring environmental conditions and guiding users on optimal ventilation times, this system aims to create a healthier living environment.

A typical use case involves:

- Monitoring temperature and humidity in rooms such as the living room, bedroom, and kitchen.
- Fetching outdoor climate data from sensors or APIs.
- Computing absolute humidity for both indoor and outdoor conditions.
- Comparing humidity levels to recommend when windows should be opened.
- Providing notifications trends via a user-friendly dashboard or mobile app.

The following assumptions has been made:

- The user has access to Wi-Fi or a reliable network for data communication.
- The user will install sensors in appropriate locations to ensure accurate readings.
- The system will rely on external weather APIs if outdoor sensors are not available.

2.2 Competitors

This Part contains research on different competitors, who fulfill atleast a part of the scenario.

2.2.1 Xiaomi “Mi Temperature and Humidity Monitor 2”

Xiaomi is a chinese brand, which is speazialised on consumer electronics, including devices for smart homes. There they are offering a smart device to measure temperature and humidity. The device includes a display and is connected via bluetooth to other devices, for example heating or vaporation. It’s shown in Image 1.



Image 1: Mi Temperature and Humidity Monitor 2 [2]

The costs for one device are around 10€. [2]

For User Notification you need the “Smart Home Hub 2”. It’s connected to the internet and offer bluetooth connectivity. The user could connect the temperature and humidity monitor to the internet and get notifications on his smartphone. The Costs for the “Smart Home Hub 2” are around 35€. You can connect multiple sensors to it. [3]

2.2.2 Bosch Smart Home “Room thermostat II”

The “Room thermostat II” is a smart home measuring device by Bosch. It’s ued in a similar way to the one made by Xiaomi.



Image 2: Room thermostat II [4]

The devices are connected via bluetooth to a Smart Home Controller, which is connected to the internet. Bosch Smart Home offers then a mobile application to check temperature and humidity and offers the opportunity to turn on heating or ventilation.

The costs compared to the solution offered by xiaomi are much higher. One "Room thermostat II" is offered for 79,99€ and the Controller for 100€. [4] [5]

2.2.3 KNX

KNX is a standardized, open protocol for home and building automation that allows various devices and systems to communicate with each other, regardless of the manufacturer. It is widely used for smart home installations and building management systems, offering features like lighting control, HVAC (heating, ventilation, and air conditioning), security, and energy management. It could be useful to connect our sensors to the knx-network. We could also implement the ventilation service and the user client with this communication protocol. But there are a few challenges. On the one hand knx devices are usually wired to each other. Also there are high costs knx user licences. This makes the system not affordable for everybody.

2.3 Functional Requirements

The Functional Requirements (FR) defining the functional behavior of the system. They are listed in Table 1.

Table 1: Functional Requirements

| Functional Requirements |
|---|
| FR1 – Indoor Temperature and Humidity Monitoring Requirement: The System can measure the temperature and humidity in multiple indoor rooms. Input: Current room temperature and humidity Output: System status update |
| FR2 – Outdoor Temperature and Humidity Monitoring Requirement: The System can fetch the outdoor temperature and humidity using API's or sensors. Input: Current outdoor temperature and humidity at a specific location Output: System status update |
| FR3 – Absolute Humidity Calculation Requirement: The System can calculate the Absolute Humidity for indoor and outdoor conditions Input: Temperature and Relative Humidity Output: System status update |
| FR4 – Compare Indoor and Outdoor Absolute Humidity Requirement: The System can compare the Absolute Humidity for indoor and outdoor conditions. Raises a flag, if the user should open a window. Input: Absolute Humidity Output: System status update, Flag |
| FR5 – User Notification to recommend when the window should be opened Requirement: Notify the user when outdoor Absolute Humidity is lower than indoor Absolute Humidity. |

| |
|--|
| Input: Absolute Humidity Comparison Output: Notification to the user |
| FR6 – User Warning for high indoor Relative Humidity Requirement: Send a warning to the user, when the indoor Relative Humidity exceeds a defined threshold. Input: Indoor Relative Humidity Output: Alert notification to the user |
| FR7 – Update Sensor Configuration Requirement: Allows the user to dynamically add or remove sensors Input: Sensor Configuration Data Output: Updated System |
| FR8 – Update Thresholds Requirement: Allows the user to dynamically update thresholds for alerts and notifications. Input: User Input Output: Updated System |
| FR9 – User registration Requirement: Allow user registration. Input: User data Output: Registered User |
| FR10 – House registration Requirement: Allow house registration. A house contains one or more rooms. Input: House data Output: Updated System |
| FR11 – Remote Window Opening Requirement: The user can open the windows remotely or start a ventilation system. Input: User command Output: Window opened, or ventilation system started |

2.4 Non-functional requirements

Beside their functional requirements there are also non-functional requirements. They are defining how the system could be used. Table 2 shows a list.

Table 2: Non-functional requirements

| Non-Functional Requirements |
|---|
| NFR1 - Cost-Effectiveness The system must be affordable to ensure accessibility for a broad range of users. |
| NFR2 - Energy Efficiency Sensors must operate with low power consumption to ensure extended battery life. Data transmission must be optimized to reduce energy usage. |
| NFR3 – Real Time Notifications |

The system must send real-time notifications to users about updates or issues, ensuring timely actions.

NFR4 - Environmental Conditions

Indoor sensors must operate reliably within 0°C to 50°C and 10%-90% Humidity.

Outdoor sensors must operate reliably within -20°C to 50°C and 0%-100% Humidity.

Outdoor sensors must withstand environmental conditions such as rain, frost, and heat.

2.5 Hardware

In this part the functional and non-functional requirements are analysed to check, which hardware is needed to fulfill them. Therefore all FR and NFR connected to the Hardware Setup are listed in Table 3.

Table 3: FR and NFR linked to hardware design

FR1 – Indoor Temperature and Humidity Monitoring

Requirement: The System can measure the temperature and humidity in multiple indoor rooms.

Input: Current room temperature and humidity

Output: System status update

FR2 – Outdoor Temperature and Humidity Monitoring

Requirement: The System can fetch the outdoor temperature and humidity using API's or sensors.

Input: Current outdoor temperature and humidity at a specific location

Output: System status update

FR11 – Remote Window Opening

Requirement: The user can open the windows remotely or start a ventilation system.

Input: User command

Output: Window opened, or ventilation system started

NFR1 - Cost-Effectiveness

The system must be affordable to ensure accessibility for a broad range of users.

NFR2 - Energy Efficiency

Sensors must operate with low power consumption to ensure extended battery life. Data transmission must be optimized to reduce energy usage.

NFR3 – Real Time Notifications

The system must send real-time notifications to users about updates or issues, ensuring timely actions.

NFR4 - Environmental Conditions

Indoor sensors must operate reliably within 0°C to 50°C and 10%-90% Humidity.

Outdoor sensors must operate reliably within -20°C to 50°C and 0%-100% Humidity.


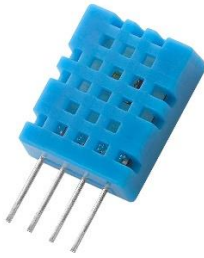
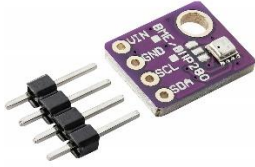
Outdoor sensors must withstand environmental conditions such as rain, frost, and heat.

2.5.1 Humidity and Temperature Monitoring

The system should measure the temperature and humidity in indoor and outdoor environments and upload them to the system. As a system we used in this case a flask server. We already gained a lot of experiences with this framework during the lab. This makes it in this case easier to implement the programm code. As a database we are using MongoDB. So the requirements is here to monitor the temperature and humidity and upload the values to the database.

During research three different temperature and humidity measuring devices have been found. They are listed and compared in Table 4.




Table 4: Compared Temperature and Humidity Sensors

| | | | |
|-----------------|--|---|--|
| Image |  |  |  |
| Name | DHT22 [6] | DHT11 [7] | BME280 [8] |
| Price | 9,49€ | 4,99€ | 10,99€ |
| Measuring Range | -40°C - 80°C 0% - 100% | 0°C - 50°C 20% - 90% | -40°C bis 85 °C 0% - 100% |
| Accuracy | ±0.5°C ±2% | ±2°C ±5% | ±1.0°C ±3% |
| Communication | Serial | Serial | I ² C |

If we analyse the sensors for the desired usecase, the DHT11 sensor isn't fulfilling the requirements in NFR4. Even if it's the cheapest one, the accuracy is quite low. For this reason, we are going to choose the DHT22. The measuring range fulfills NFR4 and it has a good accuracy.

For the board the sensor is connected to, they are different possibilities. Some of them are listet and compared in Table 5.

Table 5: Comparison of different boards

| | | | |
|----------------------------|---|--|---|
| Image |  |  |  |
| Name | Arduino Nano ESP32 [9] | Raspberry Pi Zero 2 W [10] | NodeMCU V3 ESP8266 |
| Processor | Dual-Core Xtensa LX7, up to 240 MHz | Quad-Core ARM Cortex-A53, 1 GHz | Tensilica L106, 80/160 MHz |
| RAM | 520 KB SRAM | 512 MB LPDDR2 | 32 KB Instruction + 96 KB Data |
| Storage | 384 kB ROM 16 kB SRAM in RTC (low power mode) | microSD card (user-defined size) | 4 MB Flash |
| Communication Ports | SPI, I2C, I2S, UART, CAN, GPIO | I2C, SPI, UART, GPIO, Mini HDMI port, CSI-2 camera connector | SDIO 2.0, SPI, UART, I2C, I2S, IRDA, PWM, GPIO |
| Wi-Fi | Yes (2.4 GHz + 5 GHz, Wi-Fi 4) | Yes (2.4 GHz, Wi-Fi 4) | Yes (2.4 GHz, Wi-Fi 4) |
| Bluetooth | Yes (Bluetooth 4.2 + BLE) | Yes (Bluetooth 4.2, BLE) | No |
| GPIO-Pins | 21 | 40 | 18 |
| Key Features | Dual-core, energy-efficient, versatile for IoT projects | Powerful, suitable for Linux-based applications | Simple for IoT, compact, cost-effective |
| Price | 21,96€ | 18,90€ | 7,99€ |

All three devices fulfill the requirements. For this project, the NodeMCU has been chosen. It's by far the cheapest option and offers all the connections that are needed. We also gained some experience with this device, which makes it easier and less time intensive to implement the functions. For costs reasons we want to use the same sensor for outdoor and indoor measurements. Therefore only one prototype is needed to check the functionalities.

2.5.2 Remote Window Opening

For opening and closing windows or starting ventilation remotely, we need another hardware setup. At this point of the project we don't know, what kind of actuator we might use to open the window. For this reason we want to start with a relais. The relais

could be controlled by a microcontroller and start a motor, which opens a window or power the ventilation. The relais is needed, because a microcontroller is typically not able to deliver the needed current.

A possible relais module is shown in Image 3. It consists of an ESP8266 board, which is mounted on top of a relais. It's therefore easy controllable using a wifi connection. Costs are 6,49€, which makes the module a really affordable solution. [11]



Image 3: ESP8266 Relais Module [11]

The mounted Relais is capable of handling up to 50V and 5A AC or 30V and 5A DC.

2.6 Communication

For the communication between the choosen NodeMCU and the Flask Server there are multiple ways. Three different ones are presented in the following part:

2.6.1 HTTP Request (REST API)

HTTP is the most common communication method used in web development. Each NodeMCU device can act as a client, sending data to the Flask server via HTTP requests (e.g., POST or GET). The Flask server processes these requests and responds accordingly.

How it Works:

- Each device sends a request to a specific endpoint (URL) defined on the Flask server. For example, /device/data could receive temperature readings, while /device/status might handle status updates.

Strengths:

- Simple to set up and widely supported.
- Ideal for infrequent or non-time-critical updates.

Weaknesses:

- Not efficient for real-time communication because each request is independent. Overhead in constantly re-establishing a connection for each request.

Scalability:

- Works well with several devices but can become inefficient as the number of devices or the frequency of updates increases.

2.6.2 WebSocket

WebSocket provides a persistent, full-duplex communication channel between the NodeMCU devices and the Flask server. Unlike HTTP, the connection remains open, allowing for instantaneous, real-time data exchange in both directions.

How it Works:

- The Flask server creates a WebSocket endpoint that all devices can connect to. Once connected, devices can send data to the server or receive updates from it without repeatedly opening and closing connections.

Strengths:

- Real-time, low-latency communication.
- Efficient for frequent or continuous data exchange.

Weaknesses:

- Slightly more complex to set up compared to HTTP.
- Requires keeping track of active connections for multiple devices.

Scalability:

- Works well for several devices but may require additional optimization or resources as the number of connections increases.

2.6.3 MQTT Protocol

MQTT is a lightweight messaging protocol designed for IoT systems. It uses a publish/subscribe model, where NodeMCU devices and the Flask server communicate through an intermediary called the MQTT broker.

How it Works:

- Each NodeMCU device publishes messages (e.g., sensor readings) to a specific topic on the MQTT broker.
- The Flask server subscribes to these topics to receive the data. Similarly, the Flask server can publish messages to topics that the devices subscribe to, enabling two-way communication.

Strengths:

- Designed for IoT and supports connecting a large number of devices efficiently.
- Decouples devices and the server, meaning devices don't need to know about each other's existence. Reliable delivery of messages even with intermittent connections (depending on Quality of Service settings).

Weaknesses:

- Requires setting up or using an MQTT broker (an additional component).
- Learning curve if unfamiliar with MQTT concepts.

Scalability:

- Highly scalable and supports many devices, making it the best choice for large IoT deployments.

A Comparison between the three methods is presented in Table 6.

Table 6: Comparison between communication methods

| Method | Best Use Case | Real-Time | Ease of Setup | Scalability | Efficiency |
|-----------|---|-----------|---------------|-------------|------------|
| HTTP | Periodic updates or simple data logging | X | Easy | Moderate | Moderate |
| WebSocket | Real-time control or frequent data updates | ✓ | Medium | Good | High |
| MQTT | IoT systems with many devices (publish/subscribe) | ✓ | Medium | Excellent | Very High |

For the mold prevention system the mqtt-protocol is chosen. It offers a high efficiency and scalability, which is perfect for the given problem. It can be used for the communication between server and measuring device and also between server and relais.

3 Design

The general software architecture that is used is shown in Image 4. It includes the earlier chosen MQTT-Protocol.

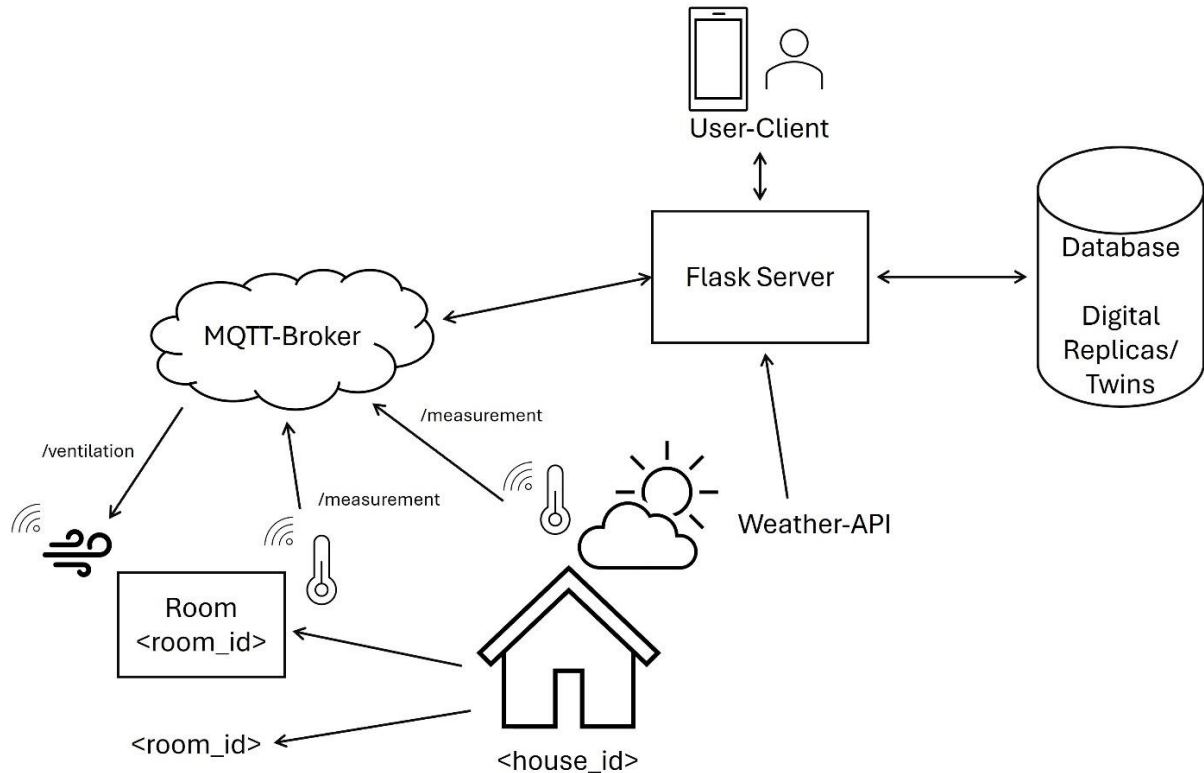


Image 4: Software Architecture

The Flask Server acts as a main part. It manages the connection to the database, where all the information is stored. The Server is also interacting with the MQTT-Broker for receiving measurements and sending commands to the ventilation system. There is also one thread to receive weather information through a public Weather-API. The server also interacts with the user through a user client. The used services and algorithms are explained in the following chapter.

The system is built on a layered architecture that promotes separation of concerns and modularity. It consists of four main layers: the Application Layer, the Digital Twin Layer, the Services Layer, and the Virtualization Layer.

- Application Layer (Interface & APIs)
- Digital Twin Layer (Core Logic)
- Services Layer (Business Logic)
- Virtualization Layer (Digital Replicas)

The Application Layer serves as the interface and API layer, providing endpoints for interacting with the system. The Digital Twin Layer contains the core logic for managing Digital Twins, orchestrating services, and coordinating data flow. The Services Layer implements the business logic, providing data processing capabilities, analytics, and monitoring. The Virtualization Layer is responsible for creating and managing Digital Replicas, handling schema validation, managing entity templates, and ensuring data consistency. In the following text the layers will be explained including more details and focus on the functionalities that we added to the existing structure.

3.1 Virtualization Layer

This Layer handles the Digital Replicas of existing physical objects. For the creation templates are used. The templates and their corresponding connections are shown in Image 5.

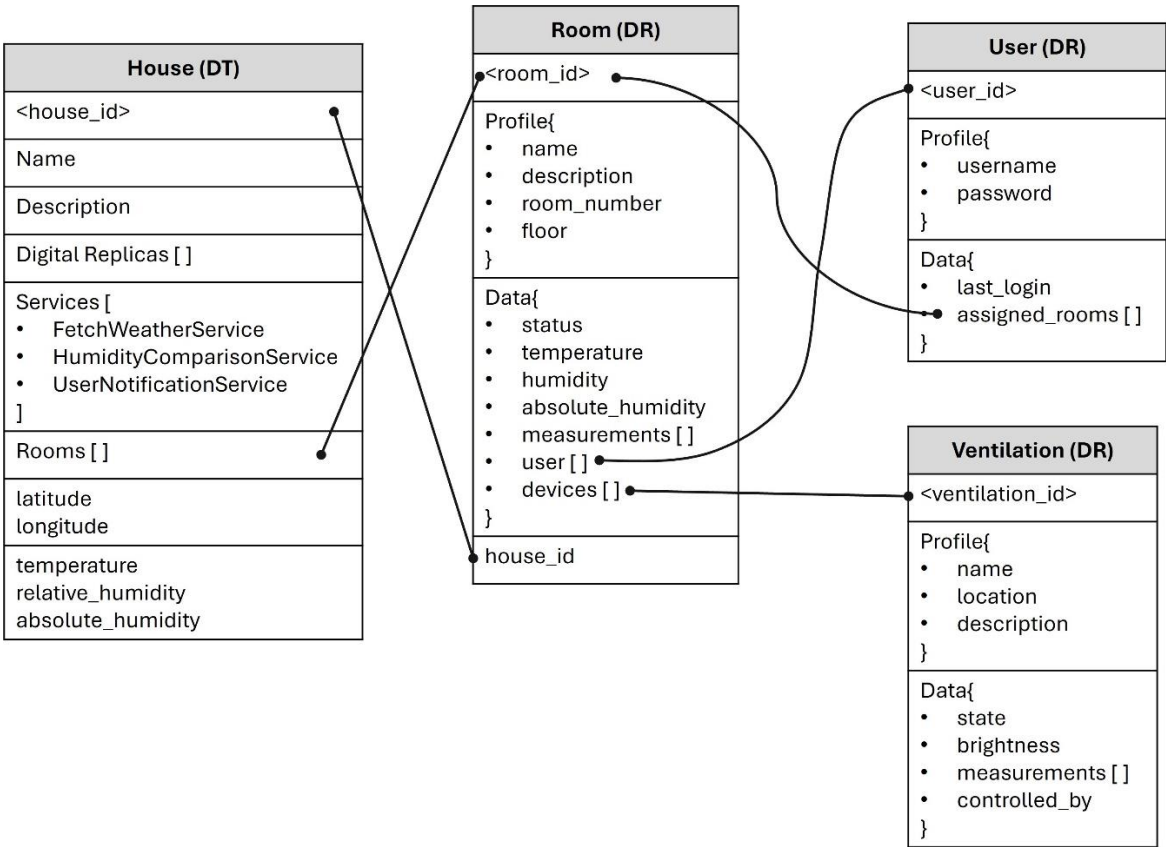


Image 5: Digital Replicas

For the House a Digital Twin is used. This allows to add services to the object. The longitude and latitude of the house location are later used, to fetch weather information. The received information are then saved to variables.

The Room is a Digital Replica. It contains general information and specific information about measured data and lists of users and ventilation devices.

The Digital Replica of a user contains the username and the password. There are also lists of assigned rooms and ventilation devices.

3.2 Services Layer

The services layer implements the core services of the Digital Twin system, providing various functionalities that can be attached to Digital Twins to extend their capabilities. This layer is built on fundamental concepts such as service independence, pluggable architecture, standardized interfaces, data processing, and extensibility. Each service is a self-contained module that can be dynamically attached to Digital Twins, ensuring flexibility and modularity. All services implement a common base interface, allowing for consistent interaction and integration. The services layer can process both real-time and historical data, enabling advanced analytics, monitoring, and control. New services can be easily added to the system, enhancing its functionality and adaptability.

In this project three main services are used:

3.2.1 Fetch Weather Service

This service aims to receive current weather information through a weather API. In this project Open-Meteo [12] is used. It is an open-source weather API and offers free access for non-commercial use. It's also easy to use.

The Open-Meteo API needs the location given in latitude and longitude as an input. It also needs an information about the requested data. In this case, the current temperature and humidity is needed. It returns the requested information as a JSON object. The values can be stored to the Digital Twin Database.

3.2.2 Humidity Comparison Service

This service compares the absolute humidity of the room and the house, the room is located in. It needs the room_id and the house_id as an input. The service then gets the environment values through the database. It calculates the difference between room and house absolute humidity and returns the result.

3.2.3 User Notification Service

This service informs the user if needed. It needs the user_id and the message as an input. The service then checks if the user is currently logged in. Then it sends the message.

3.3 Digital Twin Layer

The Digital Twin Layer is the core logic layer of the system, responsible for managing the lifecycle of Digital Twins, orchestrating services, and coordinating data flow between different components. This layer integrates the Virtualization and Services Layers to provide a cohesive and dynamic representation of physical entities in the digital space. The Digital Twin Layer manages the creation, configuration, and operation of Digital Twins, ensuring that each Digital Twin accurately reflects its physical counterpart by synchronizing state and behavior. This layer also handles the orchestration of services, enabling advanced analytics, monitoring, and control functionalities.

For this project every house is represented by a Digital Twin, as shown in Image 5. To create a new Digital Twin a new class HouseFactory, based on DTFactory is created. The added functions are described in Table 7.

Table 7: Added functions to HouseFactory

| Function Name | Description |
|-----------------------------|---|
| add_room | Adds a Room reference to a Digital Twin |
| remove_room | Removes a Room reference from a Digital Twin |
| update_temperature_humidity | Updates temperature and humidity values for a Digital Twin. |

The DigitalTwin class is also modified. A new class HouseTwin is initialized, with the DigitalTwin class as parent class. Some functions are added to manage the DigitalTwin Instance. Those are listed in Table 8.

Table 8: Added functions to HouseTwin class

| Function Name | Description |
|-----------------------------|--|
| add_longitude | Adds longitude attribute |
| add_latitude | Adds latitude attribute |
| add_temperature | Adds temperature attribute |
| add_relative_humidity | Adds relative humidity attribute |
| calculate_absolute_humidity | Calculates and sets absolute humidity based on temperature and relative humidity |
| add_rooms | Adds rooms attribute |

During creation the Services FetchWeatherService, HumidityComparisonService and UserNotificationService are added to the house.

3.4 Application Layer

The Application Layer is the topmost layer of the Digital Twin system, providing interfaces and tools for interacting with Digital Twins. This layer serves as the interface between the system and its users, offering APIs, visualization components, and user interfaces to facilitate interaction with the Digital Twins.

3.4.1 Database APIs

The Application layer includes several APIs to create, store, update or delete digital replicas. The Database APIs ensure, that the digital replicas are getting created according to our structure in Image 5. To handle the digital replicas, we are using HTTP-Requests. The API functions are listed in Table 9, Table 10 and Table 11.

Table 9: House and room API functions

| Function name | Route | HTTP Method | Description |
|---------------|---------------------------------------|-------------|---|
| create_house | /api/house/ | POST | Creates a new house digital replica. |
| get_house | /api/house/<house_id> | GET | Retrieves details of a specific house. |
| list_houses | /api/house/ | GET | Lists all houses with optional filtering. |
| create_room | /api/house/<house_id>/rooms | POST | Creates a new room within connected to a house. |
| get_room | /api/house/<house_id>/rooms/<room_id> | GET | Retrieves details of a specific room. |

Table 10: Ventilation API functions

| Function name | Route | HTTP Method | Description |
|--------------------|--|-------------|--|
| create_device | /api/ventilation/ | POST | Creates a new ventilation digital replica. |
| get_device | /api/ventilation/<ventilation_id> | GET | Retrieves details of a specific ventilation device. |
| list_devices | /api/ventilation/ | GET | Lists all ventilation devices with optional filtering. |
| toggle_ventilation | /api/ventilation/<ventilation_id>/toggle | POST | Toggles the state of a specific ventilation device between on and off. |
| create_device | /api/ventilation/ | POST | Creates a new ventilation digital replica. |

Table 11: User API functions

| Function name | Route | HTTP Method | Description |
|---------------|--|-------------|---|
| register_user | /api/user/register | POST | Registers a new user. |
| list_rooms | /api/user/<user_id> /rooms | GET | Lists all rooms connected to a specific user. |
| assign_user | /api/user/<user_id> /assign/<room_id> | POST | Assigns a room to a specific user. |

3.4.2 Telegram APIs

As a user-client this project is using a telegram bot. It's easy to configure and to use. For a final product a full smartphone application would be nice, but for prototyping and testing the telegram api is perfect.

The most important functions are listed in Table 12.

Table 12: Added functions to the telegram API

| Function Name | Command | Description |
|-------------------------|--------------------------------|---|
| login_handler | /login <username> <password | User login with credentials. List assigned rooms to the user. |
| logout_handler | /logout | User logout |
| list_rooms | /list_rooms | List all the rooms assigned to the user |
| get_room_status | /status | Handler to get the status of all rooms assigned to the user |
| ventilation_on_handler | /ON <ventilation_id> | Switches on <ventilation_id> Device |
| ventilation_off_handler | /OFF <ventilation_id> | Switches off <ventilation_id> Device |

3.4.3 MQTT-Handling

The MQTT-Handling is also included in the Application Layer. As a broker, a free cloud version offered by HiveMQ is used. It also offers an online tool to subscribe to topics and send messages. For our project we used MQTT for sending measurements to the

server and to start/ stop ventilation devices. The MQTT-Handling on the Device is explained later.

Measurement Handler

The Measurement Handler on the Flask Server listens to messages in the */measurements* topic. The flowchart in Image 6 describes the actions.

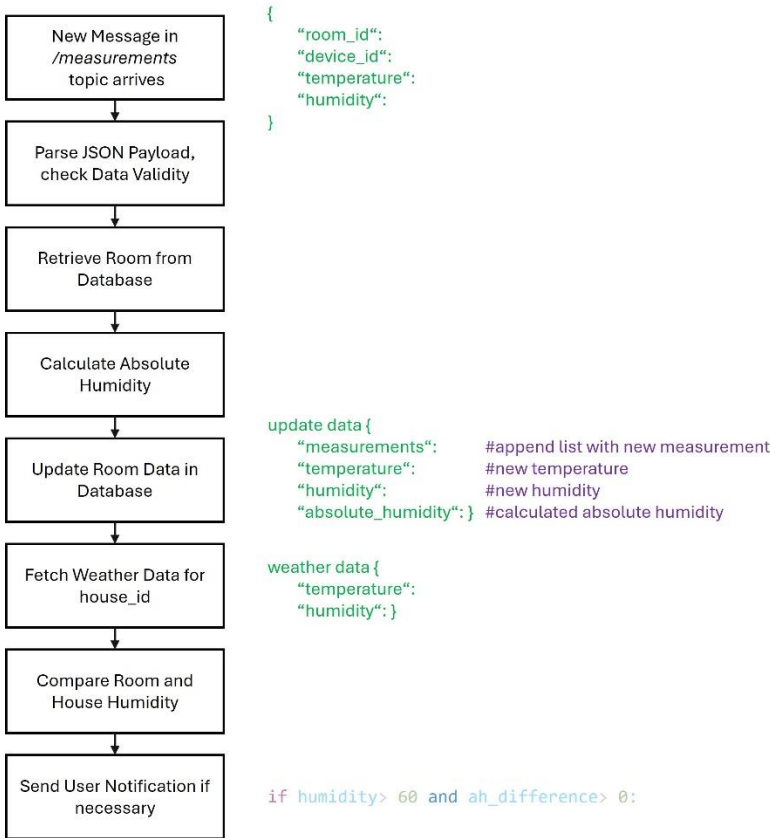


Image 6: Handling of new measurements

To execute the described actions, the mqtt handler uses the *FetchWeatherService*, the *HumidityComparisonService* and the *UserNotificationService* of the Digital Twin. If the measured environmental data is for the house, there is just a simple algorithm to update the values.

Ventilation Handler

The Ventilation Handler is sending messages to the NodeMCU that controls the Ventilation Device. For now it only switches a relais. The process of sending MQTT-Messages is shown in Image 7.

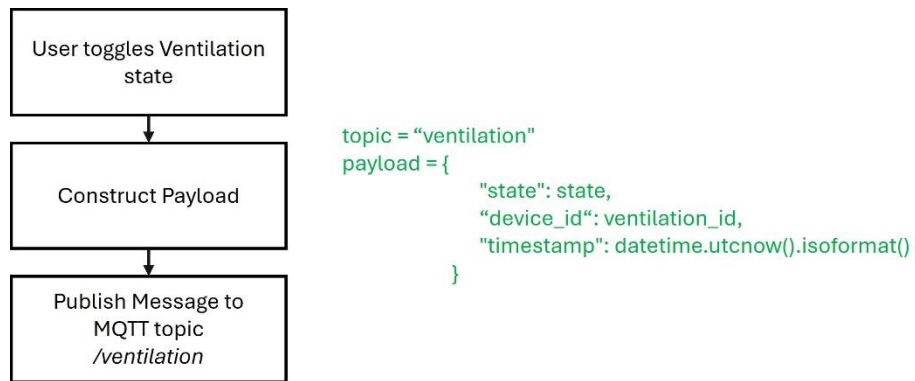


Image 7: Handling of sending ventilation status updates

There is also a second function, to publish the brightness. This could be later useful to control the ventilation, or open a window just halfway. But for now, there is only a relay. It's either open or closed, so we don't need this functionality here.

4 Prototype

This chapter describes the prototyping process of the devices.

4.1 Measuring Device

This device measures the temperature and humidity and sends it to the flask server using the mqtt protocol. The wiring is shown in Image 8.

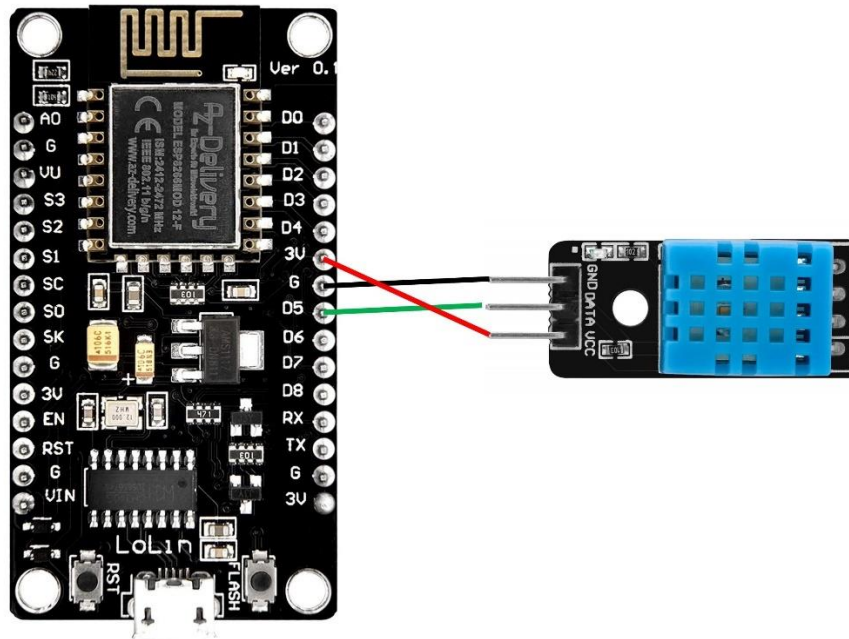


Image 8: Prototype for measuring temperature and humidity

The prototype is using a DHT11 instead of the chosen DHT22, because it was the only available sensor during the prototyping process.

The device measures humidity and temperature of the surrounding. It then publishes both to the *measurement* topic. The server subscribes to the topic and handles incoming messages as described above.

4.2 Ventilation Device

The Ventilation Device is build with the same NodeMCU. It acts as a subscriber. The topic in this case is *ventilation*. If there is a incoming message in this topic, the callback function is called. The then executed steps are shown in Image 9.

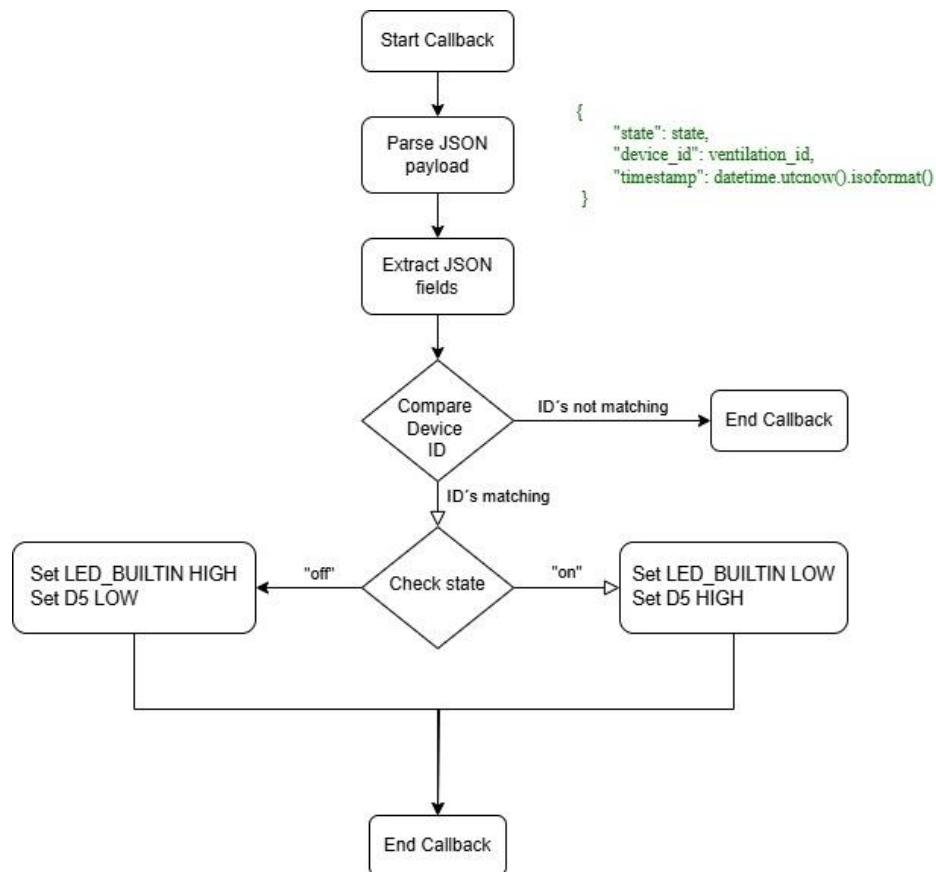


Image 9: Callback function

If the `device_id` in of the NodeMCU and the one included in the message are matching, the digital pin D5 is either set to high or low according to the state (“on” or “off”). The builtin LED is also set.

The prototype is built with a LED and is shown in Image 10. If the ID’s matching, the LED switches their state. For the real ventilation device, the LED is replaced with a relais.

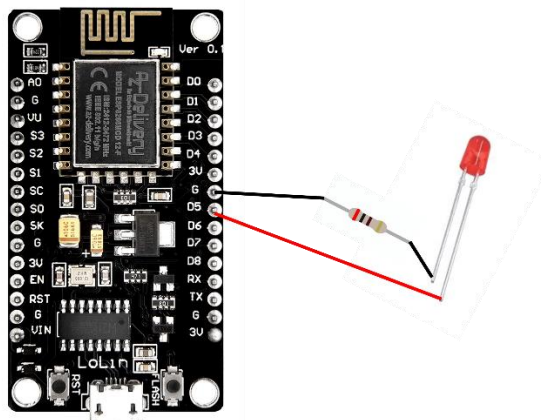


Image 10: Prototype ventilation device

5 Conclusion

The project reached most of the goals described in the functional and non-functional requirements. The status of the functional requirements are shown in Table 13.

Table 13: Functional Requirements

| ID | Requirement | Input | Output | Ful-filled |
|------|---|---|--|------------|
| FR1 | Indoor Temperature and Humidity Monitoring | Current room temperature and humidity | System status update | yes |
| FR2 | Outdoor Temperature and Humidity Monitoring | Current outdoor temperature and humidity at a specific location | System status update | (yes) |
| FR3 | Absolute Humidity Calculation | Temperature and Relative Humidity | System status update | yes |
| FR4 | Compare Indoor and Outdoor Absolute Humidity | Absolute Humidity | System status update, Flag | yes |
| FR5 | User Notification to recommend when the window should be opened | Absolute Humidity Comparison | Notification to the user | yes |
| FR6 | User Warning for high indoor Relative Humidity | Indoor Relative Humidity | Alert notification to the user | yes |
| FR7 | Update Sensor Configuration | Sensor Configuration Data | Updated System | no |
| FR8 | Update Thresholds | User Input | Updated System | no |
| FR9 | User Registration | User data | Registered User | (yes) |
| FR10 | House Registration | House data | Updated System | (yes) |
| FR11 | Remote Window Opening | User command | Window opened, or ventilation system started | yes |

The Outdoor Temperature and Humidity Monitoring as described in FR2 is at the moment only available through a weather API. The system could be better designed for that purpose.

Updating sensor configuration (FR7) or updating thresholds (FR8) is only possible by changing the values directly in the program code. For the future there could be an API to change the values directly on the server and on the NodeMCU. Updated values could be exchanged through the established mqtt broker.

To register a new house, user, room or ventilation device (FR9 and FR10) the user needs to send POST requests with the http protocol. This requires a lot of effort. It would be easier and better to use telegram handlers instead.

All in all most requirements are reached and this project shows a good way to handle indoor measurements. More functionality could be easily added to the existing system.

6 Literature

- [1] T. Instruments, „Using Relative Humidity to Derive Vapor Pressure, Dew,“ [Online]. [Zugriff am 25 01 2025].
- [2] Xiaomi, „Mi Temperature and Humidity Monitor 2,“ [Online]. Available: <https://www.mi.com/it/product/mi-temperature-and-humidity-monitor-2/>. [Zugriff am 25 01 2025].
- [3] Xiaomi, „Smart Home Hub 2,“ [Online]. Available: <https://www.mi.com/de/product/xiaomi-smart-home-hub-2/>. [Zugriff am 25 01 2025].
- [4] B. S. Home, „Room Thermostat II,“ [Online]. Available: <https://www.bosch-smarthome.com/de/de/produkte/geraete/raumthermostat/>. [Zugriff am 25 01 2025].
- [5] B. S. Home, „Smart Home Controller,“ [Online]. Available: <https://www.bosch-smarthome.com/de/de/produkte/geraete/smart-home-controller/>. [Zugriff am 27 01 2025].
- [6] AZ-Delivery, „DHT22,“ [Online]. Available: https://www.az-delivery.de/products/dht22?_pos=1&_sid=6f47a7717&_ss=r. [Zugriff am 27 01 2025].
- [7] AZ-Delivery, „DHT11,“ [Online]. Available: https://www.az-delivery.de/products/5-x-dht11-temperatursensor?_pos=5&_sid=6f47a7717&_ss=r. [Zugriff am 27 01 2025].
- [8] AZ-Delivery, „BME280,“ [Online]. Available: https://www.az-delivery.de/products/gy-bme280?_pos=3&_sid=6f47a7717&_ss=r. [Zugriff am 27 01 2025].
- [9] Arduino, „Arduino Nano ESP32,“ [Online]. Available: <https://store.arduino.cc/collections/internet-of-things/products/nano-esp32>. [Zugriff am 27 01 2025].
- [10] Raspberry, „Raspberry Pi Zero 2 W,“ [Online]. Available: <https://www.raspberrypi.com/products/raspberry-pi-zero-2-w/>. [Zugriff am 27 01 2025].
- [11] AZ-Delivery, „ESP8266 Relais Module,“ [Online]. Available: <https://www.az-delivery.de/en/products/esp8266-01s-mit-relais>. [Zugriff am 27 01 2025].
- [12] Open-Meteo, „Free Weather API,“ [Online]. Available: <https://open-meteo.com/>. [Zugriff am 01 02 2025].