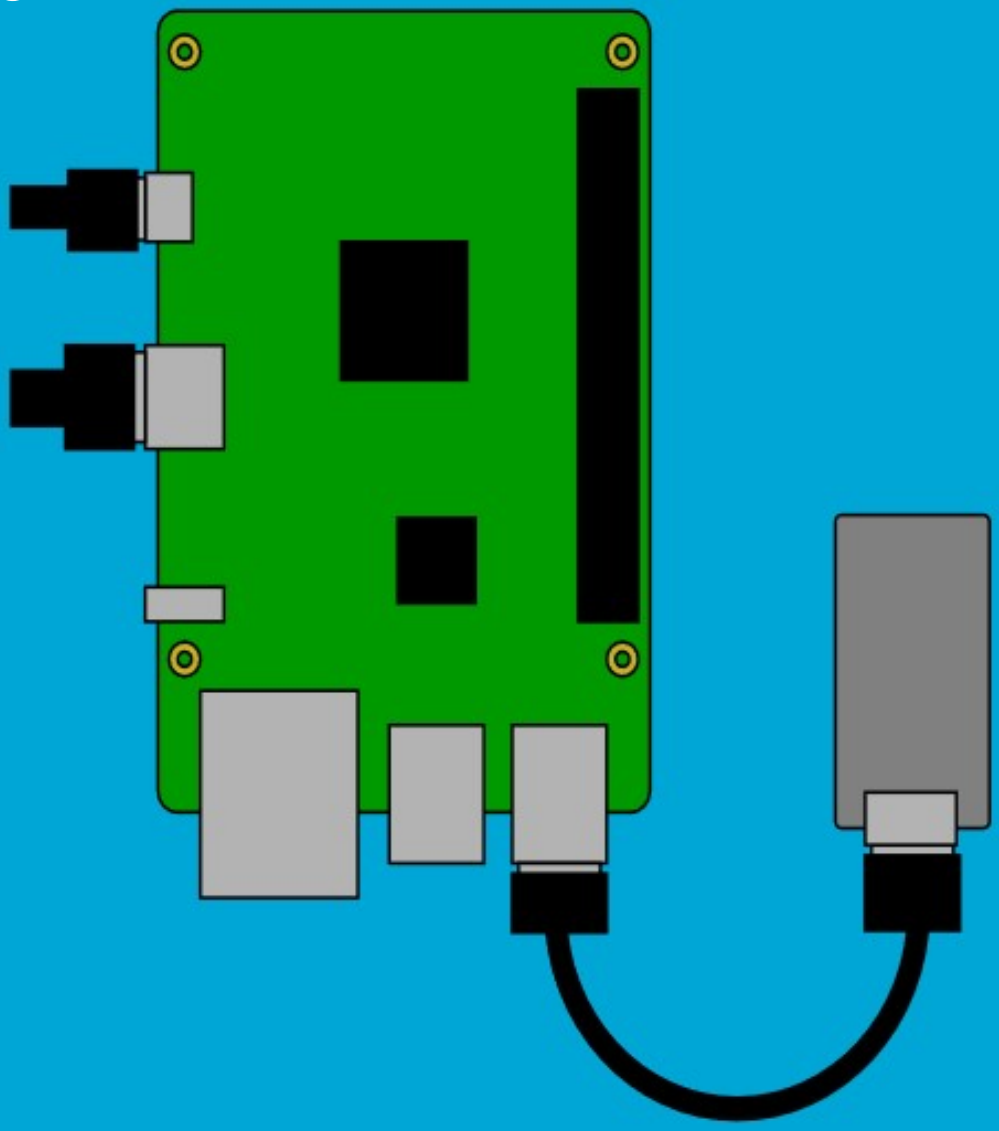
IoT-Based Smart Classroom: Server

BSc Graduation Project

Bas Smeele
Mitchell Vuong



Delft University of Technology

TU Delft

IoT-Based Smart Classroom: Server

BSc Graduation Project

by

Bas Smeele
Mitchell Vuong

Student number: 4895827 (Bas Smeele)
Student number: 4882040 (Mitchell Vuong)
Thesis committee: Dr. B. Abdi, TU Delft, supervisor
Dr. J. Martinez, TU Delft, supervisor

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft

Abstract

In this work, we propose an IoT smart classroom framework which will be the base for smart classroom and hybrid teaching applications. The prototype introduces sensor readout capabilities, with which temperature, humidity, and loudness in a classroom can be monitored. Furthermore, the prototype introduces capabilities to collect and display measurements. Additionally, this prototype will assist an educator in a hybrid teaching setup by offering additional functionality, namely notifying the educator of online questions and giving an overview of environmental measurements in the classroom. With this prototype, the educator is able to control whether the students online are able to hear physically present students and vice versa. The prototype also introduces a way to notify the educator of any questions from students attending online. The goal of this prototype is to make the first steps to improve the hybrid teaching environment by reducing the workload on the educator and by making it easier for online students to interact with the educator. This prototype is split into three theses, focusing on sensor hardware [1], the Bluetooth mesh network [2], and the server respectively. This work will have a higher focus on designing and creating a server for this prototype, which will collect and display sensor measurements.

Preface

This thesis describes the Bachelor Graduation Project of group C. The subject, IoT-based smart classroom, was given by TU Delft to be researched. This research is in relevance for future education for TU Delft, as for other educational instances. The designed system is a foundation for both a hybrid teaching system and a smart classroom system.

During the project we were guided by our supervisors dr. Jorge Martinez and dr. Bahareh Abdi. Therefore we would like to give special thanks to them. Secondly we would like to thank dr.ing. Ioan Lager for coordinating The Bachelor Graduation Project. Finally we are grateful to work with our colleagues: Ciarán Lichtenberg, Tom Goedegebuure, Richard Groenendijk, and Mathijs Binnendijk. Together we showed cooperative teamwork to reach our goal during the project.

Bas Smeele
Mitchell Vuong
Delft, June 2022

Contents

1 Introduction

In the past two years, the Covid-19 virus had a major impact on the world and heavily altered various aspects of society. Specifically, the importance of remote solutions became apparent. In the education sector this resulted in either fully online courses through software such as Zoom or a mix of online and local courses (hybrid). Now that the pandemic has mostly subsided, lectures have returned to normal. However, the pandemic did show the potential of remote teaching solutions to accommodate the needs and preferences of students and educators. The combination of face-to-face lectures with remote teaching, called hybrid teaching, has remained after the pandemic has ended. Unfortunately, this can create a disconnect between the local classroom and the online environment, for which very few solutions are currently available.

Hybrid teaching is a functionality of a smart classroom application. Therefore researching the current trend of hybrid education as well as smart classrooms will form an important context for the design and development of a successful product. Since hybrid teaching is a larger focus point than other smart classroom applications, it will be discussed separately from smart classrooms. Section 1.1 discusses the issues and challenges of hybrid teaching. Section 1.2 discusses the issues and challenges of smart classrooms. These sections are followed by the general problem definition and research of existing products. Section 1.5 describes the solution that will be designed in this thesis. The design of this solution will be subdivided in three subgroups and discussed in 3 different theses as mentioned in section 1.6.

1.1. Hybrid teaching issues and challenges

To understand the problems teachers have using hybrid education and smart classroom technology, a survey was sent to all educators of the faculty of EEMCS at Delft University of Technology. The responses to the survey can be found in appendix A. For hybrid classrooms, a recurring problem was the interaction with the online students. Questions from online students are often missed as the attention of an educator is mostly on those physically present. They find it difficult to split their attention between both groups.

Another survey [3] showed that 62% of the educators feel that hybrid teaching should be adopted. Another 23% of educators want to adopt the hybrid teaching model but are untrained and unsure about it. This shows strong support for hybrid teaching models by educators and the importance of ease of use to convince the 23% that are still unsure. Another important result of the survey is that 73% of the students are willing to adopt the new model. 18% are willing but they don't have the proper resources or skills. This shows a strong support for hybrid teaching models by students as well.
This support for hybrid teaching is justified when looking at the pros [4]:

- **Ease of participation:** Students are able to participate both synchronous and asynchronous during courses allowing for great flexibility.
- **Choice in teaching format:** The ability to choose between classroom and e-learning possibilities allows students to choose their preferred learning style.

- **Cost effectiveness for the institution:** More students making use of hybrid education can reduce the opportunity costs for institutions as well as students. The institution can spend less on classroom space, utilities and upkeep costs. Meanwhile, students can save on travel costs.

The hybrid teaching model does also have cons [4]:

- **Computer literacy:** Not every student is computer literate or has access to the resources needed for e-learning.
- **Course design difficulties:** Designing a course to effectively meet the needs of e-learning can be a sizeable challenge for the educator. Educators can also feel overwhelmed since they are responsible for more aspects such as technology, delays and dividing attention between online and face-to-face [5]. Teaching effectively while taking all these new aspects into account can introduce complexity.
- **Lacking face-to-face time:** Some students do best when physically present or need instructor face-to-face time to fully grasp a subject or to feel engaged in the learning process [6] [7]. Learning online is not the best environment for these students.
- **Teachers non-convinced by new teaching methods:** One of the interesting results based of the survey appendix A is that teachers are often non-convinced by new teaching techniques. One of the arguments presented is that the technology needed for hybrid teaching is not working properly. It also takes a lot of time and effort to understand the new technology. So this decreases the willingness to adapt to new teaching methods [8].

Regarding computer literacy, it is not much of an obstacle with the Netherlands ranking among the top EU countries in terms of digital skills according to the CBS [9]. This statistic could not be possible without overall access to the resources needed for e-learning, especially among young people.

Regarding course design difficulties, designing a course to effectively meet the needs of e-learning is difficult. This issue can be solved through education and experience. The division of students groups both online and offline can increase the amount of tasks teachers have to focus on during lecturing. *Care should always be taken to minimize the workload placed on teachers when hybrid technology is designed.*

Regarding face-to-face time, the fact that some students do better in physical based education can be taken into account within the education program. *It is important to allow for face-to-face teaching for those who prefer this style of education.*

And finally, regarding non-convinced teachers, it is understandable that not all teachers are convinced by hybrid teaching methods. It is important to take the stress and inconvenience introduced by hybrid teaching solutions into account, when they are not functioning as they are supposed to. This brings us to the last important takeaway, *The hybrid technology designed should be user friendly and intuitive.*

It can be concluded that, while hybrid teaching has its disadvantages, it can offer some significant benefits. This is why we would like to introduce a technological solution for a possible hybrid teaching application. This will be further elaborated on in section 1.5.

1.2. Smart classrooms issues and challenges

Within this project, smart classrooms are defined as classrooms equipped with technology providing some sort of benefit in an educational environment. This technology can help in hybrid education but also provide other uses in terms of security, environmental control, data collection, and visualisation. One interesting way of implementing such technologies is making use of the Internet of Things (IoT). IoT allows for an interconnected set of devices like sensors, actuators, and other technologies. All these devices are able to communicate with one another, thereby creating a wide range of possible functionalities. The following list gives some examples to get a general idea of the possibilities:

1. **Security**
   (a) Detecting persons in off-limit areas and notifying personnel.

   (b) Detecting emergencies like fires, and notifying authorities.
   (c) Wireless door locks.

2. **Environmental control**
   (a) Automated HVAC (heating, ventilation, and air-conditioning) control based on temperature and humidity measurements.
   (b) Controlling blinds based on light intensity.
   (c) Controlling lights based on occupancy.
   (d) Air quality measurement and control.
   (e) Automatically switch scenes when a different learning environment is required (like turning off lights when a projector is turned on).

3. **Hybrid teaching**
   (a) Giving educators visual feedback when there is a question online.
   (b) Automated recording of lecturers and students for online reviewing.

4. **Data collection and visualization**
   (a) Attendance tracking of students (for example by using student cards).
   (b) Using sound level measurements and machine vision to measure whether students are still paying attention.

IoT systems within education have a lot of advantages: reduction in cost, enhancement of comfort, saving time, enhancement in safety, exploring personalised learning, and increasing student collaboration [3] [10]. The need for specific applications will differ between institutions. Thus, *an IoT system should be easily customizable to fit the needs of differing institutions.*

IoT based systems also create a number of challenges. One difficulty is that teachers have problems with using the technology or technology is not working all the time. According to our survey (appendix A), educators want smart teaching solutions to work automatically without having to set it up. As classrooms are used by multiple teachers and with multiple groups of students, any smart solution should be plug and play.

Furthermore, researchers agree that applying IoT in education could create a challenge in terms of security and privacy [3] [11]. This is because of an increase in data collection on things like credentials, location, and learning history of students. This is why *security and privacy should be taken into account when designing an IoT system.*

Another obstacle in implementing IoT technology in a classroom setting is cost [11]. Educational institutions could struggle with the budget for implementing a smart classroom IoT system. This is why *the cost should be minimized when designing an IoT system.*

1.3. Problem definition

In the sections above, the advantages and disadvantages of both hybrid teaching and smart classrooms are discussed. After this assessment of the challenges educators face, the problem that will be tackled in this project is defined. In this project a framework will be designed for a smart classroom using IoT solutions. Within this framework a useful application for hybrid teaching will be integrated in the system. The complete problem definition can be stated as followed:

How to design a complete and expandable IoT system that implements a smart classroom framework and improves hybrid education?

This problem encompasses two focus points. The first being the design of a complete and expandable IoT system for a smart classroom. The problem herein solved is the difficulty of using smart solutions in a classroom. The second focus point is improving hybrid education. The main difficulty lecturers face is the interaction between online and physical audiences and not being able to focus on both (appendix A).

1.4. Product research

Hybrid teaching is a concept that has existed for a couple of years, during which multiple hybrid teaching systems have already been designed. However, no system has been able to resolve all identified issues in section 1.3. Some notable companies who have developed hybrid teaching systems are Logitech, Aver, Paramtech, and ViewSonic.

Logitech has designed a conferencing system that could be used for hybrid teaching. The system consists of at most one ultra high definition camera, two speakers, and seven microphone hubs. Since the system is not expandable, it can only be used in relatively small classrooms [12].

Aver also designs systems for conferencing purposes. This system has a person tracking camera, microphone range of 10 meters, and a single loudspeaker. This product is mainly used for conferencing since audio and video range is limited [13].

Paramtech is a company that creates solutions for hybrid teaching applications. Their hybrid teaching system is expandable up to one camera, twenty microphones, and two loudspeakers. They also offer a virtual microphone system that has two microphone bars. The audio range is for spaces up to 9.14 m by 15.24 m. The presenter in the room can walk freely because the camera has a tracking system, and the audio bars cover the whole room. But this system is applicable for a classroom specific. It can not be expanded to larger rooms as stated above. So the system cannot be used in a lecture room [14].

ViewSonic is developing a hybrid teaching system that focuses on both physical and online student engagement. The online students are projected on the wall and the physical students are audio and video recorded. At this moment ViewSonic designs systems for a standard classroom application since the amount of audio/video modules is limited. Due to the operational range of this equipment the product cannot be used for lecture halls [15].

Similar to hybrid teaching, the concept of smart classrooms is not new. With the advancements in technology, smart classroom tools have slowly crept into the education sector. For example, almost all classrooms are equipped with a projector, interactive whiteboard, or both. On another front, digital tools, such as the quizzing tool Kahoot![1], are also being used more often in lectures. Additionally, uploading lecture slides, lecture recordings, and other learning resources through software such as brightspace or blackboard, has become the norm. More recent advancements are products such as ScanMarker[2], a scanner shaped like a marker which can read text and write it to a digital text editor. Alternatively, companies such as Magicard[3] develop ID cards which can track student attendance, improve security through authentication and access control, or help regulate services such as printers or study rooms.

The future of smart classrooms looks promising, but still has a long way to go. Smart classroom products can be expensive, require extensive instructions, or be unreliable according to the responses of the survey in appendix A. All of this combined makes the education sector slow to adopt new technologies and educators hesitant to integrate new tools into their lectures (appendix A). Additionally, most tools work to enhance the already existing classroom environment and rarely try to connect the physical classroom with a remote teaching environment.

So there exist companies that offer hybrid teaching solutions. But these solutions are designed for small conferencing rooms or classrooms. A solution particular for a lecture room is not yet on the market. Secondly there are solutions for hybrid teaching and for smart education. But until now there isn't a system that covers both.

1.5. IoT-based smart classroom

When looking at existing products, see section 1.4, it can be observed that they all are quite limited in terms of functionality and/or expandability. The proposed product will address this problem, as well as the problems mentioned in section 1.3.

[1] Kahoot! is a game-based learning platform, used as educational technology in schools and other educational institutions. For more information: https://kahoot.it/

[2] For more information: https://scanmarker.com/

[3] For more information: https://magicard.com/

The core of the product is an expandable IoT network which will communicate between devices and a server. The IoT product will assist educators with hybrid teaching as well as offering smart classroom features. The smart classroom features include measuring the humidity, temperature, and loudness in a classroom to monitor the learning environment. For the hybrid teaching, a prototype will be created to make interacting in the lecture easier in a hybrid education setting for both the students and educators. See Figure 1.1 for the setup.

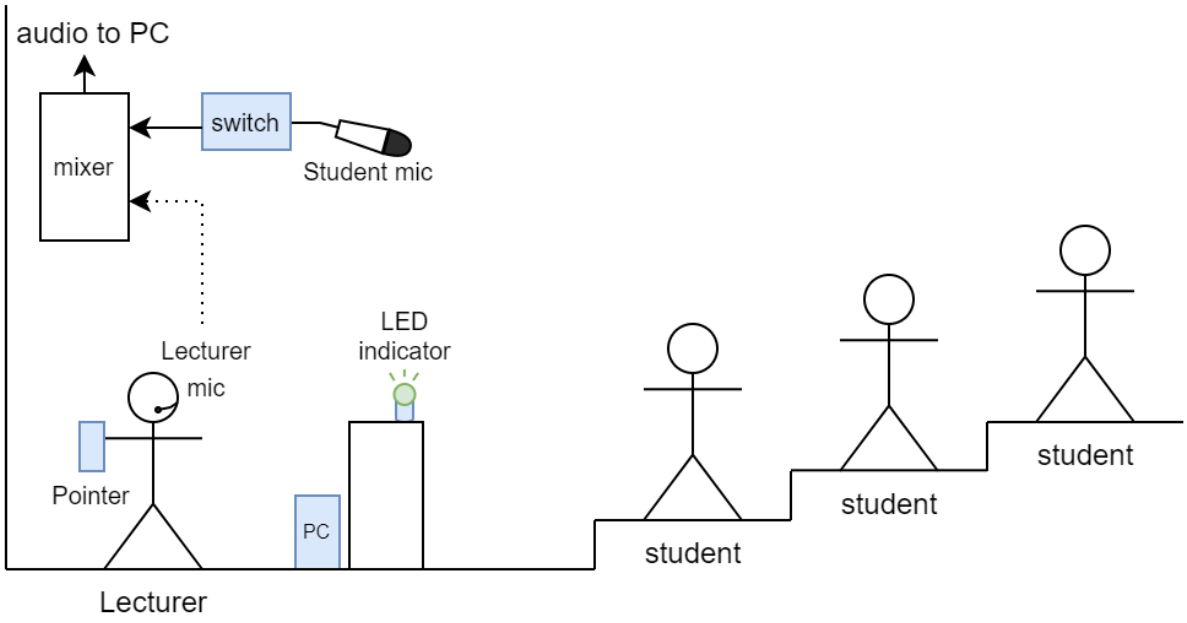


Figure 1.1: Hybrid teaching product setup

The important elements of the proposed product are: a computer running the online meeting software, an LED indicator, a laser pointer, and a student microphone. The computer running the online meeting software will have a program/app running on it, that will see if students online have a question. In turn, the computer will send a signal to indicate that a student has a question, this will light up an LED visible by the teacher. When the lecturer sees the LED indicator change color, they can then choose to unmute the online student(s) with a device attached to a laser pointer so everyone can hear the question in the class. This laser pointer can also be used to turn on the student microphone so that the students online will be able to hear the questions from the students attending the lecture in person. With this, the lecturer won't have to look for the online questions or need to repeat any questions, making the flow of the lecture smoother and decreases the work load on the lecturer.

1.6. Submodules of an IoT-based smart classroom

The project is divided into three subgroups: hardware, network and server subgroup.

1.6.1. Hardware subgroup

The hardware subgroup [1] designs and implements the peripheral devices needed for the system. These devices send include environmental sensors, audio equipment and actuators. The hardware devices communicate their data to an IoT development board.

1.6.2. Networking subgroup

The network subgroup [2] is responsible for the wireless communication between IoT devices. The hardware subgroup will communicate their data with an IoT development board. The networking group will take this data and send it wirelessly between IoT boards. This means the networking group will have to look for a suitable wireless technology. Once a suitable wireless technology is found this technology must be implemented to fulfill the requirements of the prototype.

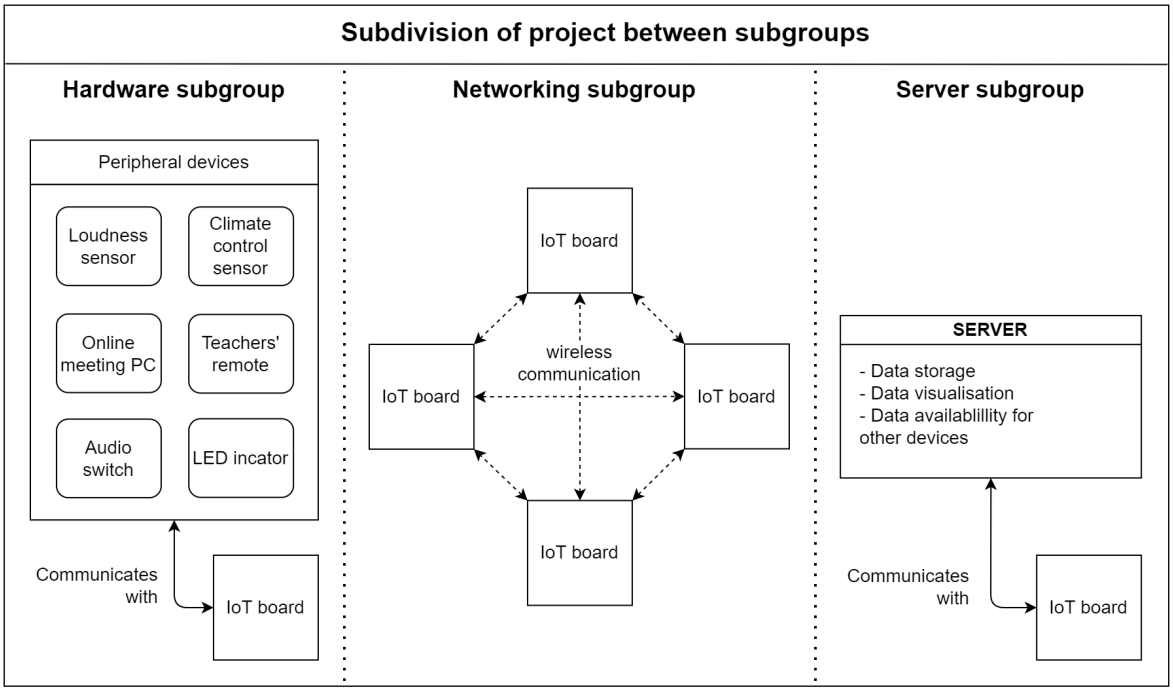


Figure 1.2: The subdivision between the subgroups

1.6.3. Server subgroup

The server subgroup designs a server which will store the acquired sensor data that will be received from the network and also provides the data that is requested to external and internal devices. This means that the server group will need to decide on a hardware platform to create the prototype on, on the software implementation of the server to make sure that the server is able to store and retrieve data, and on how the requirements of the prototype will be implemented. A further introduction to the server will come in the following section.

1.7. The server of the product

This report will go more in depth about the server that has been designed as part of the IoT prototype described in section 1.5. In order for the server to work, it needs to be able to receive, store, process, and retrieve the messages (data) it receives from the IoT network. The report will start with explaining the Program of Requirements of the prototype in chapter 2. This will be followed by the design process in chapter 3, where the considered options are presented and the final choices are explained. The implementation details and results will be presented in chapter 4. Our findings and process will be discussed in chapter 5, this chapter will also contain the conclusion of the project, some final thoughts, and future recommendations.

2

Program of requirements

In this chapter, the requirements will be set, in order to develop our IoT product for hybrid education. First, the main takeaways will be discussed, followed by the Program of Requirements (PoR) of the product, and then the PoR of the subgroup will be defined. The main takeaways and PoR are labelled with TA for takeaways, M for mandatory requirements, T for trade-off requirements, and B for boundary conditions. The server specific requirements use the same labels proceeded by an S for server.

2.1. Main PoR takeaways

The context research in chapter 1 yielded some important points when making the PoR:

TA1 Care should be taken to minimize the workload placed on teachers when hybrid technology is designed.
TA2 It is important to allow for face-to-face teaching for those who prefer this style of education.
TA3 The hybrid technology designed should be user friendly and intuitive.
TA4 An IoT system should be easily customizable to fit the needs of differing institutions.
TA5 Security and privacy should be taken into account when designing an IoT system.
TA6 The cost should be minimized when designing an IoT system.

2.2. General PoR

The mandatory requirements, which were chosen in order to create a good working product, can be found below. The Trade-off requirements are chosen to improve the product as well. The boundary conditions are chosen to determine what will be left outside the scope of the project, and what will be included. The numbers following some of the requirements are references to the main PoR takeaways discussed in section 2.1.

Mandatory requirements

General

M1 The data measured from sensors must be readable through a GUI (Graphical User Interface). [**TA1**, **TA3**]
M2 The collected data must be available for external devices. [**TA4**]
M3 The system must be able to log and save data.

IoT

M4 The network must be able to accept new IoT devices while in operation. [**TA3**]
M5 IoT devices must be able to send and receive data via a wireless network.
M6 The network should support senors applicable for a classroom. [**TA4**]
M7 The coverage of the network must be expandable. [**TA4**]
M8 Losing an IoT device should not completely stop the operation of the network. [**TA3**]

Hybrid

M9 The teacher must be notified of any questions from online students. [**TA1**, **TA3**]
M10 The teacher must be able to switch between mute and unmute of the audience microphone and the students online. [**TA1**, **TA2**]
M11 The online students must be able to hear the physically attending students. [**TA2**]

Trade-off requirements

T1 The system should be affordable. [**TA6**]
T2 The system should be customizable. [**TA4**]
T3 The system should operate without extensive technical knowledge. [**TA1**, **TA3**]
T4 The system hardware should have a small form factor.
T5 The system should have basic event-based decision-making capabilities.

Boundary conditions

B1 The security of the system will not be a main focus. [**TA5**]
B2 The privacy of the system will not be a main focus. [**TA5**]
B3 Only IoT devices applicable to a classroom will be discussed.

While security and privacy are important considerations, these topics are quite extensive. Therefore they are taken into consideration, but within the allotted time of this project, further research and design into this area is left outside of the scope of this thesis.

2.3. Server specific PoR

For the server of the project, there are also some additional requirements in addition to the general PoR, these are stated below.

Mandatory requirements

SM1 The server must be connected to the network.
SM2 The server must receive and interpret messages.
SM3 The server must not miss any new messages.
SM4 Storage and computation must be done locally.

Trade-off requirements

ST1 The software should be light weight.
ST2 The software should be easily configurable.

3

Design

This chapter will discuss the design choices made for the server. The PoR in chapter 2 has been taken into account when designing the server. We start with our decisions on designing a prototype server in section 3.1. Next, the decision on the database/middleware will be discussed in section 3.2. The visualization with the GUI will be explained in section 3.3, and section 3.4 will discuss the web server. Finally, the software design will be discussed in section 3.5.

3.1. Hardware

For the hardware used for the server, the following popular hardware platforms have been compared in [16]:

- Raspberry Pi
- Arduino
- BeagleBone Black
- Phidgets
- Udoo

The article that compared these single board computers had it's main attention on the functionality, performance, and constraints of these platforms. The comparison was done, based on: size & cost, power & memory, flexibility, communication, and operating systems & programming languages. Since the hardware is only used to create a working prototype, the following criteria were taken into account: size & cost and power & memory. A quick overview can be seen in Table 3.1. The values that are shown in **bold**, are the most desirable values for considering a hardware platform.

Table 3.1: Comparison overview between popular hardware platforms used for an IoT server, based on [16]

Name	Size (mm)	Weight (g)	Cost per node (US$)	RAM	Power
Raspberry Pi	85.6 x 52.98 x 17	45	**25-35**	256-512 MB	**5 V/USB**
Arduino	**75 x 53 x 15**	**~30**	30	16-32 KB	7-12 V/USB
BeagleBone Black	86.3 x 53.3	39.68	45	512 MB	5 V
Phidgets	81.3 x 53.3	60	50-200	64 MB	6-15 V
Udoo	110 x 85	120-170	99-135	**1 GB**	6-15 V

As shown in Table 3.1, it shows that the Raspberry Pi is not the smallest, lightest option, or has the biggest RAM, but it is comparably cheaper and uses a small amount of power. An IoT device must be energy efficient and thus requires a low power solution [17]. Using a Raspberry Pi would be one of the most efficient options. From these hardware platforms, the Raspberry Pi is also the cheapest, this means that a lot of Raspberry Pis can be used for a relative small price. This comes favorable with

main PoR takeaway **TA6** from section 2.1. The Raspberry Pi is the second smallest and the second lightest model, right after the Arduino. Small and light devices are more practical in more locations and scenarios [16]. Not having the biggest memory won't be a main issue either, as Raspberry Pi is also compatible with Secure Digital (SD) Cards [16] [18]. Moreover, the Raspberry Pi is shown to be an affordable single board computer which can support a large number of input and output peripherals as well [16]. In conclusion, considering the advantages of raspberry pi over other platforms and the goal of the project, it is the best option for our prototype server.

Figure 3.1: Image of a Raspberry Pi 3 model B

3.2. Database

In order to meet requirement **M3** and **SM4**, a database will be needed in the server. For this, some IoT Middleware solutions have been taken into consideration.
Middleware is the software that connects different components or applications in order to run multiple processes on one or multiple machines to interact across a network. IoT applications fall into 2 general categories [19]:

1. Ambient data collection and analytics
2. Real-time reactive applications

The first category of the applications involves collecting sensor data from a variety of sensors, the data will be processed to create a model and make predictions for new data to be collected from the sensor in the future. The second category involves real-time reactive systems such as autonomous vehicles or manufacturing processes where the system make real-time decisions.
The following Middleware solutions have been considered [20]:

Mires

An event-based message-oriented Middleware that implements a traditional publish/subscribe

solution, which allows query and extract data from the network and communication between Middleware services.

DPWS
Includes specifications for executing remote procedure calls, security, authorization and service discovery. The main advantage is, it uses standard internet protocols with the option of having decentralized service discovery.

TinyDB
A framework based on TinyOS that lets its users see a WSN as a database. The queries on the sensors results in data acquired by the network. It supports aggregation, energy-aware query constraints and continuously running queries [21]

SensorsMW
It provides a service oriented architecture with the aim of integrating WSN into the internet. One of the main features is that there is a possibility to configure a WSN according to the needs of client applications

MQTT-S
It uses publish/subscribe concept with broker nodes network. The extension is Middleware targeted at low-end battery powered design principle aim at minimising network bandwidth and device resource requirements whilst targeting reliability.

Linksmart
It provides tools for the in-clusion of heterogeneous devices into IoT application. Updates in ebbits are focused on an extension of the events processing capabilities. As well as on scalability and performance improvements semantic repositories

VITRO
A VSN (virtual sensor network) is composed of sensors or sensor clusters that might be located in different regional or administrative domains. In this case VITRO is used

SENSEI
A heterogeneous wireless Sensor and Actuator networks (WS&AN) are integrated into a common framework of global scale. It is accessible by services and applications through universal service interfaces based on eXtensible Markup Language (XML)

LooCI
It has a Lightweight runtime re-configurable component model. The features are loosely-coupled, event-based binding model. all the information in the network is carried as events.

After comparing all these Middleware solutions, we decided to use TinyDB as Middleware considering the requirements of this project. It is a database written in pure Python, and it is perfect for making a simple database that works without lots of configuration [21]. This would also fit with requirement **ST2**. However, if more advanced features, or a higher performance is needed, other Middleware solutions are more optimal.

3.3. GUI

To satisfy requirement **M1**, a GUI must be designed to make the collected data readable. By doing this, it will make it easier for the user to understand what kind of data is collected and what it means. As a result, the user will have an easier time to use the IoT product.
In order to easily inform the user of the current data recorded in the room, the GUI will be updated with the most recent data, in real time. The data that will be presented on the GUI will be the current temperature, humidity, and loudness.
To create the GUI, the Tkinter package from Python has been used. Tkinter is a widely used Python interface to the GUI toolkit [22]. Tkinter is convenient to use, due to it being an object oriented interface. This helps to make an attractive graphical interface that makes it easy to interact with for the end user [23].
The GUI will be discussed in more detail in section 4.1.4.

3.4. Web server

To make the data accessible for external devices (requirement **M2**), a web server is required.
The web server is used to request and view the data from the server. The idea is that other people or applications can connect to the web server, collect the data that the server has stored from its sensors, and process them if needed. This way, the product is easily expandable, by making it easier to make use of other IoT applications that are not initially part of the original product and without needing to connect to the Bluetooth network.
In section 4.1.5, the web server will be discussed in further detail.

3.5. Software Design

Following from requirement **M7**, to allow the functionality of the server to be easily expandable, the software was developed using a modular design approach. In this approach different functionalities are designed as separate modules with a focus on reducing dependency between models. This allows for the creation and addition of new modules without needing in depth knowledge of the other modules. When running multiple modules concurrently, one might drastically slow down the execution of other modules. This can result in missing certain events or other time-sensitive operations. Running the modules in parallel would solve this issue, allowing modules to run while other modules are in an active or semi-active state. Achieving such parallel processing can be done in multiple ways:

Synchronous execution
: Processes/programs are run on threads. In the case of synchronous execution, a process/program runs on a single thread. Usually this is sufficient, but when making calls to outside functions that take time to return (also called i/o bound), this halts the execution of the process even though it is not executing anything. Synchronous execution does not support parallel behaviour and is thus not sufficient for this application.

Multithreading
: Instead of running on one thread, a process can start another thread. These threads can run in parallel, thus executing different functions at the same time while sharing the same resources. Unfortunately this can lead to some difficult issues, such as race conditions when multiple threads try to access the same resource. Therefore it is very important to keep a multithreading process thread safe, which can be very difficult. Additionally, python uses a global interpreter lock, which allows only one thread to execute at the time. Multithreading does bring the required parallel processing, but can be very difficult to work with [24].

Asynchronous execution
: Instead of using multiple threads, asynchronous execution achieves parallel execution in a single thread. Asynchronous execution uses an event loop on which it schedules functions/tasks. When making an input/output bound function call some other part of the process can be executed while waiting for the function to return. This return is then scheduled on the event loop to execute as soon as possible. If another part of the process is dependent on the function that is awaiting a return, it is also paused until the original function can continue. Since this is all running on a single thread, asynchronous execution is not actual parallel execution. However, for this application this is not an issue [25].

Multiprocessing
: Instead of using multiple threads, multiprocessing creates multiple processes, each with their own resources. This achieves parallel execution without the difficulties of multithreading, namely sharing resources and the global interpreter lock. However, starting a new process is slower and uses more memory. Multiprocessing would also be sufficient solution for this application, but asynchronous execution is more suitable [26].

Thus, the asynchronous execution was used, to achieve parallel processing. As mentioned above, asynchronous execution achieves parallel execution, unlike synchronous execution, it does not bring the difficulties presented by multithreading, and it is faster and uses less memory than multiprocessing. For a server that will need to store a lot of data and receives a lot of messages, asynchronous execution seems to be the best fit. This also helps with achieving requirement **SM3**, by not missing any new messages.

4

Implementation

This chapter will discuss implementation details based on the design choices described in chapter 3. Section 4.1 describes the software architecture and its components. Section 4.2 elaborates more on the physical aspects of the prototype. Finally, section 4.3 will present the testing of the prototype and its results. All the code written for the server can be found in appendix B.

4.1. Software architecture

The software of the server is split into different modules each with different functionality. Each module can run separately and communicates with the other modules when needed. Figure 4.1 shows the relations between all modules and the methods of each module.

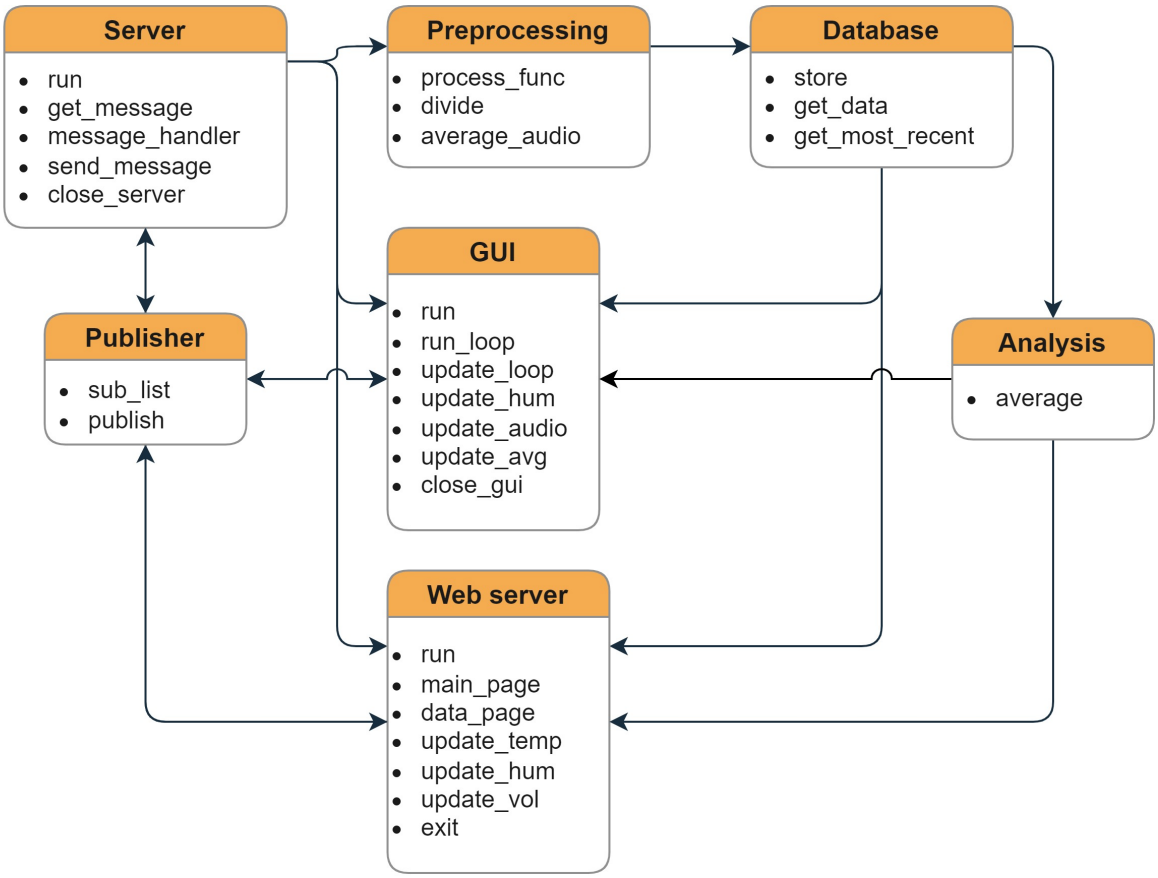


Figure 4.1: Software architecture

4.1.1. Asynchronous execution

In order to assure cooperation between the modules, they all share a single event loop implemented through the AsyncIO library [25]. Within an asynchronous program functions are defined as coroutines through the async keyword. Coroutines are awaitable through the await keyword. While a coroutine is awaiting some return, control is given back to the event loop, which can start executing other coroutines. Additionally, coroutines can be added to the event loop as tasks to run as soon as possible or at a later time. The event loop manages all these tasks and coroutines. It continously checks if any awaiting coroutines have returned, continuing with other operations otherwise. This allows the modules to run concurrently in a single thread by scheduling tasks, which are executed when other modules have some downtime. All running or pending tasks are added to a to-do list, which is shared between the modules. This to-do list is managed by the server module, which removes any completed tasks and ensures all tasks are either completed or properly cancelled before shutdown. Additionally, modules can communicate with the publisher module to notify other modules of certain events. The publisher module keeps track of a subscriber list for each event and schedules the appropriate coroutines when a new event is received.

4.1.2. Server module

The server module is the main module and initializes the other modules. It also keeps track of all pending tasks on a to-do list and ensures all pending tasks are resolved if the server is closed. Finally the server module is the gateway between the Bluetooth Mesh and the other modules. It handles incoming messages, instructs other modules accordingly, and sends its own messages to the mesh if necessary. In order to streamline development, the server module can be run either in test mode or in operation mode. In operation mode, the server gets incoming messages from the network. In test mode, the server uses its get_message method to simulate receiving messages by generating randomized messages at randomized intervals.
When a new message is received, the server module schedules its message_handler method to interpret new messages. This method checks the message type and the sender of the message. From there it decides what to do with the message and instructs other modules accordingly. The close_server method reacts to a shut down event to ensure a graceful shut down.

4.1.3. Database module

The database module is the primary location for storing and accessing data. The received data is stored in a json file managed by the TinyDB library [21]. Each new entry is stored in a separate document based on the received time. If there already is a document with that time, the entry is added to that document. This way any data can be queried based on requested time periods, or data can individually be queried through the TinyDB API. Finally, the database module implements a method that gets the most recent entry of a specified data type. This method looks through all documents from newest to oldest and returns once it finds a document which contains the requested data type.

4.1.4. Graphical User Interface

The GUI module is the gateway between a local user and the other modules. It creates a new window through the TKinter library [22], on which it displays the current temperature, humidity, audio volume, and a graph displaying the temperature and humidity changes over the last 10 minutes. Additionally it presents an exit button to shut down the entire server. Figure 4.2 Shows an image of the described GUI. Traditionally TKinter manages a separate event loop for the GUI, however to ensure proper cooperation between the models the window is updated through the update method and schedules itself to update again 10 times per second. Similarly, instead of querying the most recent entries every time the GUI is updated, it keeps track of them in its own attributes and tells the publisher module to notify it whenever a new entry of that datatype is added. For the graph it similarly keeps an array with averages, calculated by the analysis module, and replaces outdated values according to the specified time interval. After updating the array of averages, it schedules to update the array again in a minute.

4.1.5. Web Server

The web server module presents an interface through a web page with the quart library [27]. Similarly to the GUI it displays the current temperature, humidity, and audio volume. Additionally it presents a

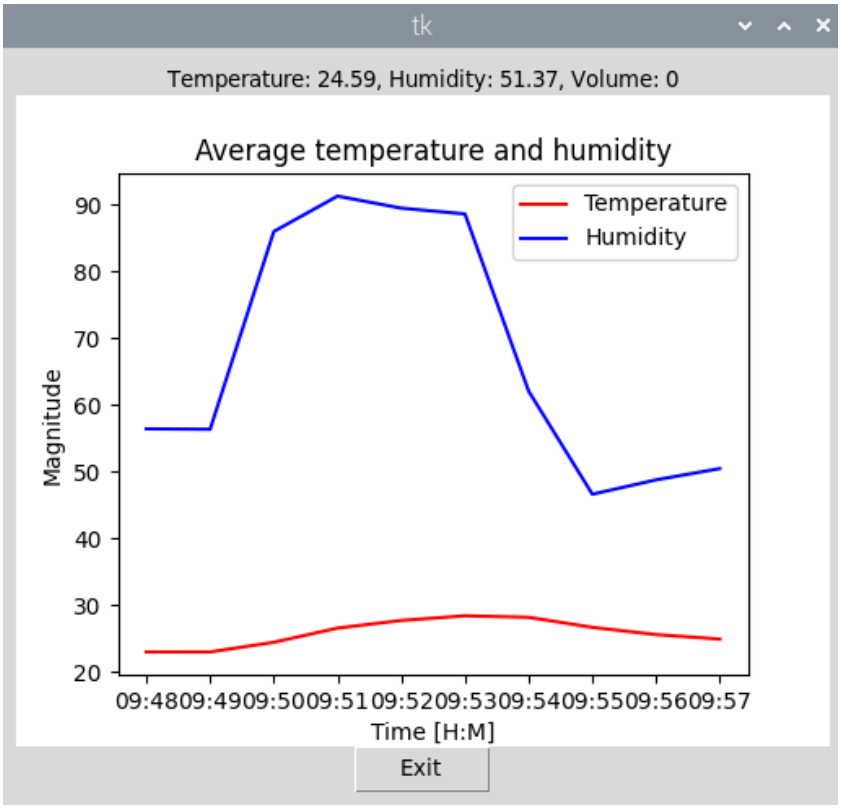
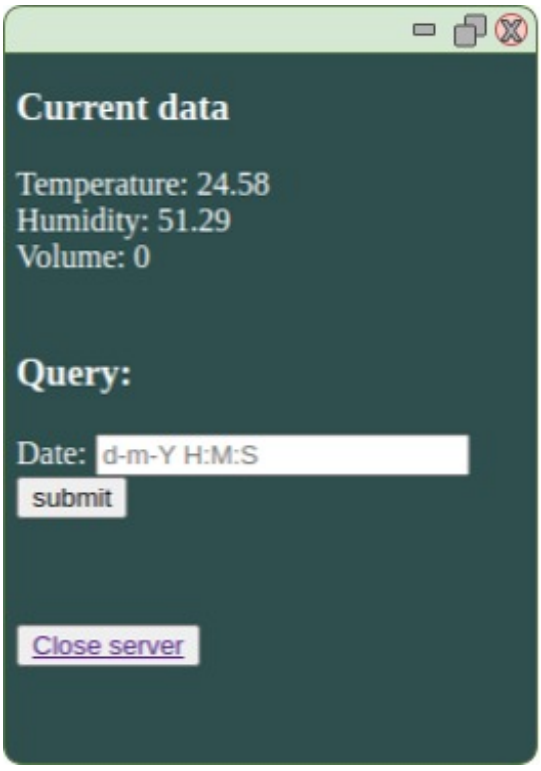
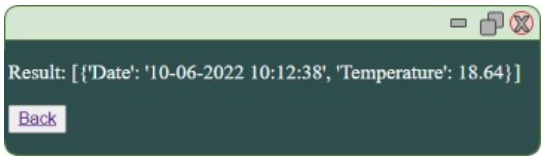


Figure 4.2: GUI



(a) Web page



(b) Web page query return

Figure 4.3: The GUI and web page

query field to allow external users and devices to request data from the database. On request, the web server returns a page with the desired data. Finally the web page has an exit button to close the server remotely. Figure 4.3a shows an image of the described web interface. Similarly to the GUI module, the web server subscribes to new data entries and updates its stored attributes accordingly.

4.1.6. Data Pipeline

Sensor data sent on the Bluetooth Mesh network is received by the server as a message with the DATA tag. This message contains the property ID and a data field, among other attributes. When the server receives a DATA message, the message handler translates the property ID to its associated data type (temperature, humidity, audio volume, etc.) and passes the data type and the data to the preprocessing module. The preprocessing module recognizes the data type and passes the data through the appropriate processing function. Finally the data and the data type is passed to the database and a new data event is published. The GUI and the web server can access this data by either directly querying the database module, or through the analysis module. The analysis model, in its current implementation, only supports an average method. This method takes a duration and a datatype, and returns the average of that data type over the specified duration. More analysis functionality can be added, however was not necessary for this prototype. Figure 4.4 shows a diagram of the described data pipeline

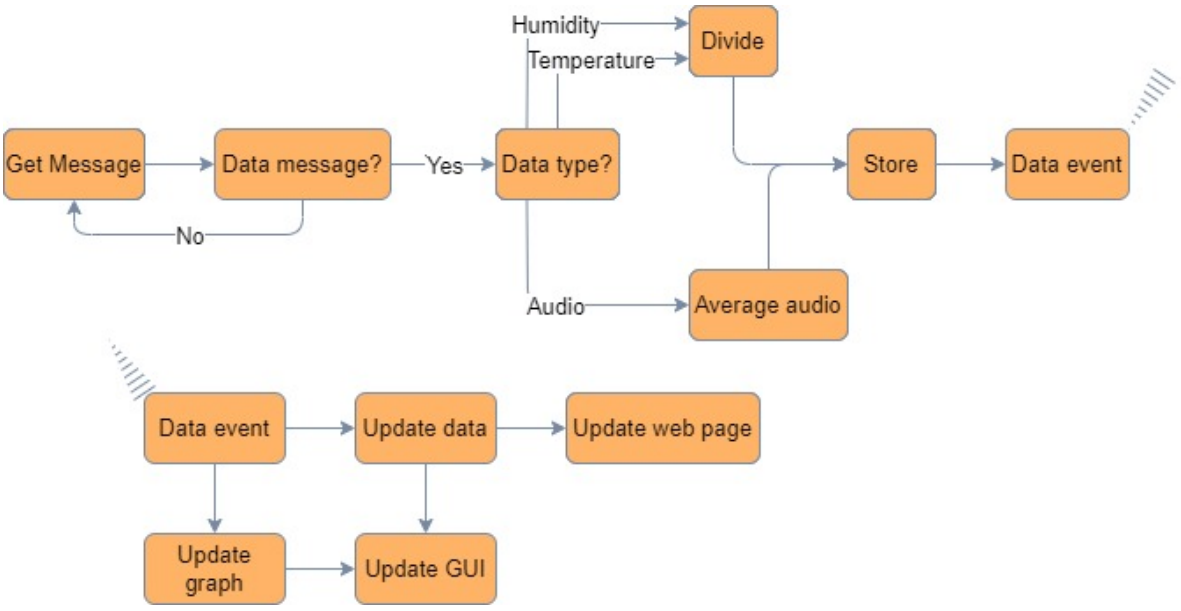


Figure 4.4: Data pipeline

4.2. Prototype Implementation

The prototype implements the software described in section 4.1 on a Raspberry Pi 3B. The Raspberry Pi is connected through USB to an M5Stamp C3 development board [28], which is the provisioner node for the Bluetooth Mesh network [2]. Any message this node receives can be read by the Raspberry Pi through the serial link. Whenever the Raspberry Pi receives a new message, the server module adds the message handler with the received message to the to-do list. This way the Raspberry Pi can start looking for the next message while other parts of the software processes the newly received message. The GUI continuously updates to reflect the current sensor measurements. Figure 4.2 shows the implemented GUI. Similarly, the web server also updates the most recent measurements when requested. Additionally, the web server returns a web page with the requested data when a query is submitted in the query field. This query has to be a date and time, formatted as: [day-month-year hour:minute:seconds]. If this is not the case, or if the submitted date and time has no recorded data, the web page returns an empty array. Figure 4.3a shows the implemented web page and Figure 4.3b shows the returned page after submitting a query.
Due to modular design of the software, functionality can be altered or added within its respective module

and only has to interact with what it actually needs.

4.3. Testing and Results

The server prototype can either be run in testing mode or operation mode, making it possible to test independently of the other subgroups. As mentioned in section 4.1.2, in testing mode the server generates random messages similar to how they would be received from the Bluetooth mesh network. Initially, testing was done with the described testing mode. Later on testing shifted to a mix of using test mode and operation mode.
In the final test setup of the completed prototype, the Raspberry Pi is connected to an M5Stamp C3 development board [28] and a temperature and humidity sensor send their data on the bluetooth mesh network. For the first two minutes the sensors record the environmental variables of the room. After two minutes, both sensors are covered with a hand. And after 5 minutes the hand is removed again. The graph in Figure 4.2 shows an appropriate change in temperature and humidity during the mentioned times. Similarly, the web interface returns the appropriate data when requested. Data can be queried through the web interface, returning an empty array when the requested data does not exist. Finally, both the GUI and the web interface can gracefully shut down the server when necessary. Figure 4.5 shows a diagram of the used test setup.

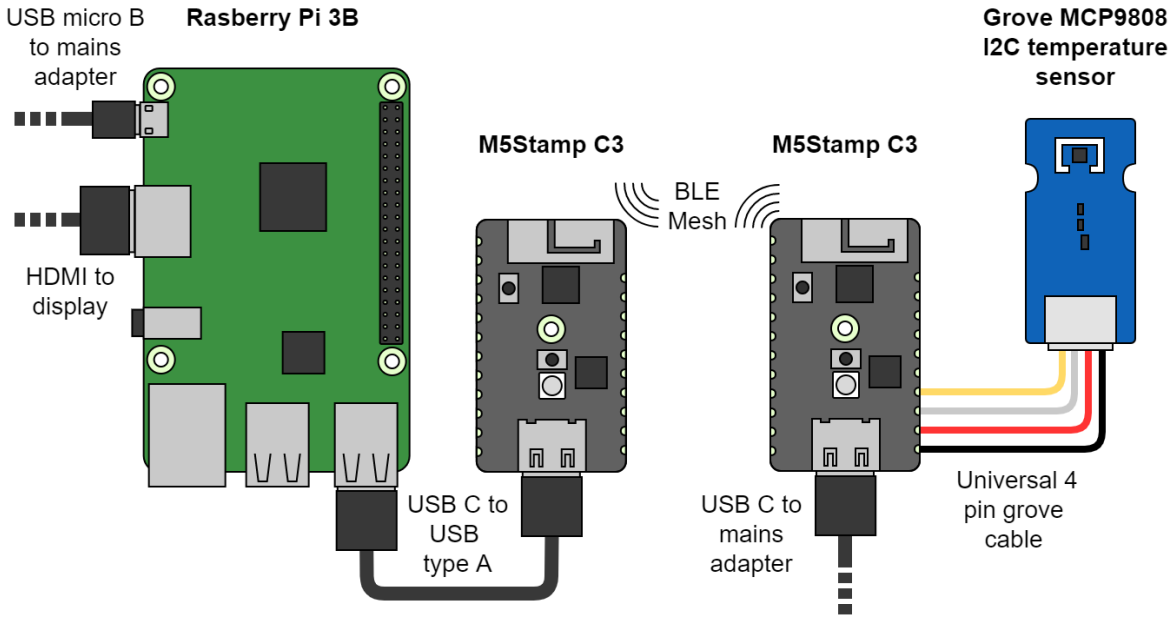


Figure 4.5: Server prototype test setup

5

Conclusion & Future Work

In the final test the completed prototype behaved as expected; It receives incoming data, stores it in a database after being processed, and displays the appropriate information on the GUI and web interface.

5.1. Conclusion

Coming back to the problem definition, stated in section 1.3: “How to design a complete and expandable IoT system that implements a smart classroom framework and improves hybrid education?”. This report mainly focused on designing and making a server for this expandable IoT system. As of now, a prototype has been created that meets the requirements. The software implemented on the Raspberry Pi makes the prototype relatively cheap. Furthermore, the modular design of the software and the use of asynchronous execution makes it flexible and creates a framework that can be expanded upon.
The server, in combination with the other components designed by the other subgroups [1] [2], can be used as a minimum working prototype for the described IoT system in section 1.5. This minimum system is able to achieve the smart classroom functionalities like sensing and logging the data about the classroom’s temperature, humidity, and loudness. Additionally, it would be relatively easy to let the server handle new data, when more hybrid teaching functionalities are implemented.

5.2. Revisiting the PoR

When comparing the prototype with all the requirements mentioned in chapter 2, it is clear that the server meets the general requirements of the complete system (section 2.2) and the server specific requirements (section 2.3).

General PoR

When looking at the general PoR, some of the requirements that have been achieved are:

- The server has a working GUI (**M1**).
- The server can host a web server, on which the data can be requested with the use of queries (**M2**).
- The server makes use of TinyDB to log and save the received data (**M3**).

Some other requirements that have been achieved, but could be improved are:

- The server is able to receive all kinds of data over the IoT network. However, it is only able to send data to a web page and not yet over the IoT network via Bluetooth (**M6**).
- The server is relatively cheap, since it is all programmed on a Raspberry Pi (**T1**).
- The server is customizable and the prototype offers some program abstraction. However, knowledge about the server and the IoT system is still necessary (**T2**).
- The server can be used without a lot of knowledge of the system/software, thanks to the GUI. However, the GUI could be expanded with additional functionality and customizability (**T3**).
- The server is able to recognize and process events that are received over the IoT network. However, this is still simple since more advanced events are not necessary in the prototype (**T5**).

The remaining requirements in the general PoR do not apply to the server, or were out of the scope of the project (like the boundary conditions).

Server specific PoR

When looking at the server specific PoR, the requirements that have been achieved are:

- The server is connected to the Bluetooth mesh network (**SM1**).
- The messages that have been received over the mesh network, are correctly processed (**SM2**).
- With the use of asynchronous operation, we can receive all messages and execute when the system has time to do so (**SM3**).
- The server does not need any external help to store and compute the data (**SM4**).

Looking at the trade-off requirements:

- The software runs in a single thread, with the use of asyncio, in a single process, and it is able to run on a Raspberry Pi. So for the use of the system, the software can be considered light weight (**ST1**).
- The software is somewhat configurable due to the modular design. However, this was not a main focused, thus coding knowledge is required to be able to add functionality or change configuration. A future improvement could be the addition of a graphical configuration tool (**ST2**).

5.3. Prototype Limitations and Future Directions

However, while this prototype does implement all necessary functionality, it has some limitations.

Web Page

The web page is quite primitive, only allowing one query at a time based on date and time. Different applications use data in different formats, or might only need the temperature. In the current prototype implementation, these kind of requests would be very inefficient. Furthermore, there are no restrictions on who can access the web page, thus anyone could request any data or shut down the server. Requiring authentication before being able to access certain data or functionality would be preferable. Finally, the web page could be extended with more IoT and hybrid classroom functionality. For example, being able to instruct an HVAC system remotely.

GUI

In a similar vein, the GUI is also still in an early stage. It adequately displays some of the key points of the collected data, but could be expanded to offer more interaction with other devices on the Bluetooth network. The ability to instruct an HVAC system would be well suited for a GUI. Another good improvement is to give customization options. That way an educator can alter the layout and displayed information according to their needs.

Serial Communication

Another limitation is that the Raspberry Pi receives its message through serial communication with a M5Stamp C3 board [28]. A preferable implementation would be to have the Raspberry Pi connected to the network with Bluetooth and be able to provision newly added devices. This would make it easier to add new devices in the Bluetooth network and simplify the deployment of the system.

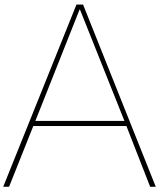
Security and Privacy

While little additional efforts were made towards security, as mentioned in requirement **B1**, it is a major flaw in the prototype. All the collected data is stored in plain text, thus offers little resistance to malicious actors. Additionally, the web page makes it all accessible without authentication. Although little sensitive data is stored in this prototype, a layer of encryption and authentication would be preferable when handling more sensitive data and functionality. On the other hand, computation and storage are done locally, making the system more resistant to internet malfunctions and more secure to cyber attacks when compared to cloud based alternatives.

Hybrid teaching integration

Finally, the server part of the IoT prototype acts mostly as an endpoint. The server only collects

data and has little further interaction with the other parts of the IoT system. Previously discussed limitations already brought up this issue, mainly the web page and GUI lack the functionality to instruct other IoT devices. Furthermore, the server does not respond to queries made through Bluetooth, only through the web page.



Survey results

What problems do you encounter with hybrid teaching?

1. That both groups expect to be the number 1 audience, while as teacher it is impossible to serve both groups equally. As an effect both groups are to some extend disappointed with the experience. It works fine if one audience is the target and the other group is either watching in or is 'audience'.
2. Chalkboards are not readable because of the light coming in. Often the entire board is filmed which makes text too small to read. The camera does not follow the teacher well.
3. Complications setting up cameras, smartboards and meetings and keeping an eye on the online participants during the lecture.
4. Lack of feedback, questions or participation from students connected online.
5. Students online do not engage. Focus is on students physically present.
6. Focus is to 1 group only, in most cases the ones that are present
7. Making the online people feel connected to the onsite group.
8. Hard to keep track of questions both in the room and online.
9. Serving two audiences at the same time.
10. Dividing my attention.

In what way could technology assist with or solve these problems?

1. A digital board that is streamed directly or writing on a tablet. A solution for following the teacher is unknown to me. I do not know if it is great to have the teacher stay in a fixed spot.
2. A different screen for projecting the students camera that join online, a way to share the white board with the students online.
3. Work for one of the target groups only and improve the experience of the secondary group using technology.
4. Perhaps having a large screen (tv sized) in the audience to see the students at home could help.
5. If VR would be good enough that people online has the same perspective as in the room.
6. Less technology would be preferable (i.e. a proper blackboard) but a quality camera.
7. An up-voting mechanism for important questions.

What could be improved about hybrid teaching?

1. The quality of the sound and video streaming and setting it up. When managed from outside (for instance in aula) this works very well, but the quality is much worse when the lecturer has to set it up themselves.
2. Large smartboards which can be streamed directly (instead of via a camera) or good writing tablets of which the screen gets streamed and projected.
3. No smart boards, but traditional smart boards so you only need one technology (online meeting+camera) to function.
4. Balanced presence and attention of online and physically present students.
5. Participation of students joining online.
6. Connection with those online.
7. Interaction with each other.

In an ideal world with endless possibilities, what would be your vision on a hybrid classroom?

1. Either TV show setting where you do your class for the remote participants and the co-located ones are audience - as such they even have a lesser experience than the people at home, so we should give them all the tools and add ons that the people at home have too. OR we make co-located session where the people remote join in in such a way that all the things they miss from the co-location are compensated. I like the Getmibo townhall setting as example where remote environments bring co-located features.
2. A camera which follows the teacher automatically and a smartboard as big as an old school chalkboard with a second screen of the same size so you can shove the empty screen in front of the full screen.
3. All students turn on the cameras, a different screen for showing the students connected online so we 'feel' that also students joining online are actually there.
4. Just very high quality video and audio (but I still prefer the offline interaction, so I'd still prioritise those in the room).
5. Have the online people in the same line of vision as the onsite people. Like the holodeck.
6. If VR would be good enough that people online has the same perspective as in the room.
7. Blackboard+good camera+screen for feedback with online students.

What problems do you encounter in a smart classroom?

1. Every system is different and I use it not often. As such the UI always makes you wonder how it works.
2. Sometines setting up the systems is really difficult.
3. Smart boards not working
4. Technology not working.

In what way could technology assist with or solve these problems?

1. Replace smartboards with proper blackboards
2. Have less malfunctions.
3. Voice response

What could be improved about smart classrooms?

1. Often the controls are fixed to the desk and the desk as such needs a fixed spot. Before you know it the desk becomes THE central place in front, which it is not but cannot moved away.

In an ideal world with endless possibilities, what would be your vision on a smart classroom?

1. Fully automatice - no display - maybe voice controlled
2. It is easier to set up all the systems.

B

Code

B.1. Python code

B.1.1. analysis.py

```python
from datetime import datetime, timedelta


class Analysis:
    def __init__(self, db):
        self.db = db

    def average(self, datatype: str, period: int):
        total = 0
        count = 0

        time = (datetime.today()-timedelta(seconds=period)).strftime('%d-%m-%Y %H:%M:%S')
        for entry in self.db.db.search(self.db.User.Date >= time):
            if datatype in entry:
                total += float(entry[datatype])
                count += 1

        if count != 0:
            return total/count
        return None
```

B.1.2. database.py

```python
from tinydb import TinyDB, Query
from datetime import datetime


class Database:
    def __init__(self):
        self.db = TinyDB('data/testdb.json')
        self.db.truncate()
        self.User = Query()

    def store(self, data_type, data):
        date = datetime.today().strftime('%d-%m-%Y %H:%M:%S')
        self.db.upsert({'Date': date, data_type: data}, self.User.Date == date)
        print(date, ' - ', data_type, ': ', data)

    def get_data(self, datatype=None):
        """
        input:
        datatype = what data it wants
        Needs a datatype to return most recently logged data
        """
        d_id = self.db.all()[-1].doc_id
```

```
23          requested_data = self.db.get(doc_id=d_id)
24          if datatype:
25              try:
26                  r_data = requested_data[datatype]
27                  return r_data
28              except Exception as e:
29                  print(e)
30                  return 0
31          else:
32              return None
33
34      def get_most_recent(self, datatype=None):
35          last_id = len(self.db)
36          for i in range(last_id):
37              if datatype in self.db.get(doc_id=(last_id-i)):
38                  return self.db.get(doc_id=last_id-i)[datatype]
39          return None
```

B.1.3. GUI.py

```
1  from datetime import datetime
2  from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
3  from matplotlib.figure import Figure
4  from tkinter import *
5  from tkinter import ttk
6  import asyncio
7
8  from analysis import Analysis
9
10
11 class GUI:
12     def __init__(self, db, publisher, todo):
13         self.db = db
14         self.publisher = publisher
15         self.todo = todo
16         self.gui_tasks = [None, None]
17         self.analysis = Analysis(self.db)
18
19         self.root = Tk()
20         self.root.protocol('WM_DELETE_WINDOW', self.close_gui)
21
22         self.frm = ttk.Frame(self.root, padding=10)
23         self.frm.grid()
24
25         ttk.Label(self.frm, text='Temperature: ..., Humidity: ..., Volume: ...').grid(
               column=0, row=0)
26         ttk.Button(self.frm, text='Exit', command=self.close_gui).grid(column=0, row=2)
27
28         self.fig = Figure(figsize=(5, 4), dpi=100)
29
30         self.avg_plot = self.fig.add_subplot(111)
31         self.avg_plot.title.set_text('Average temperature and humidity')
32         self.avg_plot.set_xlabel('Time [H:M]')
33         self.avg_plot.set_ylabel('Magnitude')
34         self.avg_plot.plot()
35
36         self.canvas = FigureCanvasTkAgg(self.fig, self.frm)
37         self.canvas.draw()
38         self.canvas.get_tk_widget().grid(column=0, row=1)
39
40         self.temp = [0]
41         self.temp_time = ['0']
42
43         self.hum = [0]
44         self.hum_time = ['0']
45
46         self.audio = 0
47
48         self.avg_temp = [0]
49         self.avg_hum = [0]
```

```
50          self.avg_time = ['0']
51
52          self.question = 0
53
54          self.publisher.sub_list['Temperature'].append(self.update_temp)
55          self.publisher.sub_list['Humidity'].append(self.update_hum)
56          self.publisher.sub_list['Audio'].append(self.update_audio)
57          self.publisher.sub_list['Close'].append(self.exit)
58          self.publisher.sub_list['Question'].append(self.update_question)
59
60      async def run(self):
61          self.gui_tasks[0] = asyncio.create_task(self.run_loop(), name='gui_loop')
62          self.todo.add(self.gui_tasks[0])
63
64          self.gui_tasks[1] = asyncio.create_task(self.update_avg(), name='avg_graph_loop')
65          self.todo.add(self.gui_tasks[1])
66
67      async def run_loop(self):
68          ttk.Label(self.frm, text=f'Temperature: {self.temp[-1]}, '
69                                   f'Humidity: {self.hum[-1]}, '
70                                   f'Volume: {self.audio}').grid(column=0, row=0)
71
72          self.avg_plot.clear()
73          self.avg_plot.plot(self.avg_time, self.avg_temp, color='r', label='Temperature')
74          self.avg_plot.plot(self.avg_time, self.avg_hum, color='b', label='Humidity')
75          self.avg_plot.set_xlabel('Time [H:M]')
76          self.avg_plot.set_ylabel('Magnitude')
77          self.avg_plot.title.set_text('Average temperature and humidity')
78          self.avg_plot.legend()
79
80          self.canvas.draw()
81          self.root.update()
82
83          await asyncio.sleep(0.1)
84
85          self.gui_tasks[0] = asyncio.create_task(self.run_loop(), name='gui_loop')
86          self.todo.add(self.gui_tasks[0])
87
88      async def update_temp(self):
89          if len(self.temp) >= 10:
90              self.temp.pop(0)
91          self.temp.append(self.db.get_most_recent('Temperature'))
92
93          if len(self.temp_time) >= 10:
94              self.temp_time.pop(0)
95          self.temp_time.append(datetime.today().strftime('%M:%S'))
96
97      async def update_hum(self):
98          if len(self.hum) >= 10:
99              self.hum.pop(0)
100         self.hum.append(self.db.get_most_recent('Humidity'))
101
102         if len(self.hum_time) >= 10:
103             self.hum_time.pop(0)
104         self.hum_time.append(datetime.today().strftime('%M:%S'))
105
106     async def update_audio(self):
107         self.audio = self.db.get_most_recent('Audio')
108
109     async def update_question(self):
110         self.question = 1
111
112     async def update_avg(self):
113         if len(self.avg_temp) >= 10:
114             self.avg_temp.pop(0)
115         self.avg_temp.append(self.analysis.average('Temperature', 60))
116
117         if len(self.avg_hum) >= 10:
118             self.avg_hum.pop(0)
119         self.avg_hum.append(self.analysis.average('Humidity', 60))
120
```

```
121         if len(self.avg_time) >= 10:
122             self.avg_time.pop(0)
123         self.avg_time.append(datetime.today().strftime('%H:%M'))
124 
125         await asyncio.sleep(60)
126         self.gui_tasks[1] = asyncio.create_task(self.update_avg(), name='avg_graph_loop')
127         self.todo.add(self.gui_tasks[1])
128 
129     def close_gui(self):
130         self.publisher.publish('Close')
131 
132     async def exit(self):
133         for task in self.gui_tasks:
134             task.cancel()
135         self.root.destroy()
```

B.1.4. preprocessing.py

```
1 
2 class Preprocessing:
3     def __init__(self):
4         self.process_func = {'Temperature': self.divide, 'Humidity': self.divide, 'Audio'
              : self.average_audio}
5         self.audio_total = 0
6         self.audio_samples = 0
7 
8     def divide(self, data):
9         data = float(data)/100
10        return data
11 
12    def average_audio(self, data):
13        self.audio_total += int(data)
14        self.audio_samples += 1
15        if self.audio_samples >= 10:
16            audio_average = self.audio_total/self.audio_samples
17            self.audio_total = 0
18            self.audio_samples = 0
19            return audio_average
```

B.1.5. pubsub.py

```
1 import asyncio
2 
3 
4 class Publisher:
5     def __init__(self, todo):
6         self.sub_list = {'Temperature': [], 'Humidity': [], 'Audio': [], 'Question': [],
              'Close': []}
7 
8         self.todo = todo
9 
10    def publish(self, topic):
11        if topic not in self.sub_list:
12            print('Unknown topic')
13            return
14        for sub in self.sub_list[topic]:
15            task = asyncio.create_task(sub(), name=f'{topic}_events')
16            self.todo.add(task)
```

B.1.6. Server.py

```
1 import asyncio
2 import serial
3 import random
4 try:
5     import serial_asyncio
6     operation_mode = None
7 except ModuleNotFoundError:
8     print('Module serial_asyncio not found, continuing in test mode...')
```

```
    operation_mode = False

from GUI import GUI
from database import Database
from pubsub import Publisher
from preprocessing import Preprocessing
from webapp import WebApp


class Server:
    def __init__(self, operation_mode):
        self.loop = True
        self.todo = set()
        self.prop_id_translation = {'0x75': 'Temperature', '0xa7': 'Humidity', '0x79': '
            Audio'}

        self.reader = None
        self.writer = None
        self.operation_mode = operation_mode

        self.db = Database()
        self.publisher = Publisher(self.todo)
        self.preprocessing = Preprocessing()
        self.gui = GUI(self.db, self.publisher, self.todo)
        self.webapp = WebApp('webapp', self.db, self.publisher, self.todo)

        self.publisher.sub_list['Close'].append(self.close_server)

    # simulates waiting for a message with a random wait time of 0, 1, or 2 seconds
    async def get_message(self):
        wait_time = random.randint(0, 3)
        print('Message wait time: ', wait_time)
        await asyncio.sleep(wait_time)

        # generate a random message within the format
        msg = f'color clock DATA: pubaddr receivedaddr {random.choice(["0x75", "0xa7", "0
            x79"])} ' \
              f'{random.randint(100, 3000)} END'
        print(msg)
        return msg

    async def send_message(self):
        self.writer.write(b'message send')

    # identify message type and pass to appropriate function
    async def message_handler(self, msg):
        parsed_msg = msg.split()

        try:
            if parsed_msg[2] == 'DATA:':
                data_type = self.prop_id_translation.get(parsed_msg[5])
                data = self.preprocessing.process_func.get(data_type, lambda x: x)(
                    parsed_msg[6])
                if data is not None:
                    self.db.store(data_type, data)
                    self.publisher.publish(data_type)

            elif parsed_msg[0][0:2] == '011':
                self.publisher.publish('Question')

            else:
                print('message not supported')

        except IndexError:
            print('message too short')

    async def run(self):
        if self.operation_mode:
            try:
                self.reader, self.writer = await serial_asyncio.open_serial_connection(
                    url='/dev/ttyACM0', baudrate=115200)
```

```
76              except serial.SerialException:
77                  print('No USB connection found, continuing in test mode...')
78                  self.operation_mode = False
79
80          await self.gui.run()
81          await self.webapp.run()
82          i = 0
83
84          while self.loop:
85              # wait for a new message
86              if self.operation_mode:
87                  msg = str(await self.reader.readline())
88              else:
89                  msg = await self.get_message()
90
91              print(msg)
92
93              # add the message to a list of task to be completed
94              self.todo.add(asyncio.create_task(self.message_handler(msg), name=f'{i}'))
95
96              # check if any tasks have been completed
97              pending = asyncio.all_tasks()
98              # and remove those tasks from the list
99              self.todo.intersection_update(pending)
100             print('Pending tasks: ', len(self.todo))
101
102             i += 1
103
104         # wait for all tasks to be completed before exiting the program
105         while len(self.todo):
106             print('Waiting for all tasks to finish...')
107             done, _pending = await asyncio.wait(self.todo, timeout=1)
108             self.todo.difference_update(done)
109             msg_ids = (t.get_name() for t in self.todo)
110             print(f'{len(self.todo)}: ' + ', '.join(sorted(msg_ids)))
111
112     async def close_server(self):
113         self.loop = False
114
115
116 if __name__ == '__main__':
117     while operation_mode is None:
118         print('Test or operation mode? t/o')
119         toinput = input()
120         if toinput == 't':
121             operation_mode = False
122             print('Continuing in test mode...')
123             break
124         elif toinput == 'o':
125             operation_mode = True
126             print('Continuing in operation mode...')
127             break
128         print('Unrecognized mode')
129     server = Server(operation_mode)
130     asyncio.run(server.run())
```

B.1.7. webapp.py

```
1  import asyncio
2  from quart import Quart, render_template, request
3  import os
4
5
6  class WebApp:
7      app = None
8      def __init__(self, name, db, publisher, todo):
9          self.app = Quart(name)
10         self.db = db
11         self.publisher = publisher
12         self.todo = todo
```

```
13
14          self.app_task = None
15
16          self.temp = 0
17          self.hum = 0
18          self.vol = 0
19
20          publisher.sub_list['Temperature'].append(self.update_temp)
21          publisher.sub_list['Humidity'].append(self.update_hum)
22          publisher.sub_list['Audio'].append(self.update_vol)
23
24          self.publisher.sub_list['Close'].append(self.exit)
25
26
27      async def run(self):
28          self.app_task = asyncio.create_task(self.app.run_task())
29          self.todo.add(self.app_task)
30          self.app.add_url_rule('/', 'main_page', self.main_page)
31          self.app.add_url_rule('/exit', 'close_server', self.close_server)
32          self.app.add_url_rule('/data_page', 'data_page', self.data_page, methods=['POST'
                ])
33
34      async def main_page(self):
35          templatedata = {'title': 'Main page', 'temp': self.temp, 'hum': self.hum, 'vol':
                self.vol}
36          return await render_template('server.html', **templatedata)
37
38      async def data_page(self):
39          date = (await request.form)['date']
40          templatedata = {'title': 'Data page', 'res': self.db.db.search(self.db.User['Date
                '] == date)}
41          return await render_template('datapage.html', **templatedata)
42
43      async def close_server(self):
44          self.publisher.publish('Close')
45          templatedata = {'title': 'Exit page'}
46          return await render_template('exitpage.html', **templatedata)
47
48      async def update_temp(self):
49          self.temp = self.db.get_most_recent('Temperature')
50
51      async def update_hum(self):
52          self.hum = self.db.get_most_recent('Humidity')
53
54      async def update_vol(self):
55          self.vol = self.db.get_most_recent('Audio')
56
57      async def exit(self):
58          self.app_task.cancel()
```

B.2. HTML

B.2.1. datapage.html

```
1  <!DOCTYPE html>
2     <head>
3        <title>{{ title }}</title>
4        <link rel="stylesheet" href="../static/style.css/">
5        <link rel="">
6     </head>
7  <body>
8      <p>
9          Result: {{res}} <br/>
10     </p>
11     <p>
12         <button> <a href="/">Back</a> </button>
13     </p>
14 </body>
15 </html>
```

B.2.2. exitpage.html

```
<!DOCTYPE html>
   <head>
      <title>{{ title }}</title>
      <link rel="stylesheet" href="../static/style.css/">
      <link rel="">
   </head>
<body>
    <p>
        Server closed <br/>
        <button> <a href="/">Back</a> </button>
    </p>
</body>
</html>
```

B.2.3. server.html

```
<!DOCTYPE html>
   <head>
      <title>{{ title }}</title>
      <link rel="stylesheet" type="text/css" href="../static/style.css">
      <link rel="">
   </head>
   <body>
   <h3>Current data</h3>
   <p>
      Temperature: {{ temp }} <br />
      Humidity: {{ hum }} <br />
      Volume: {{ vol }} <br />
      <br />
   <p>
      <h3>Query:</h3>
      <form action="/data_page" method="post">
         Date: <input type="text" name="date" placeholder="d-m-Y H:M:S"> <br />
         <input type="submit" name="form" value="submit" /> <br />
      </form>
      <br /> <br />
   </p>

   <p>
      <button> <a href="/exit">Close server</a> </button>
   </p>
   </body>
</html>
```

B.3. CSS

B.3.1. style.css

```
body {
   background: darkslategrey;
   color: white;
}
```

Bibliography

[1] T. Goedegebuure R. Groenendijk. “IoT-Based Smart Classroom Subgroup: Hardware”. In: (2022).

[2] Ciarán Lichtenberg Mathijs van Binnendijk. “IoT based smart classroom: networking subgroup”. In: (2022).

[3] Kshitij Shinghal et al. “IOT Based Modified Hybrid Blended Learning Model for Education”. In: *2020 International Conference on Advances in Computing, Communication Materials (ICACCM)*. 2020, pp. 229–232. DOI: 10.1109/ICACCM50413.2020.9213049.

[4] La Vonne Fedynich. “Teaching beyond the classroom walls: The pros and cons of cyber learning.” In: *Journal of Instructional Pedagogies* 13 (2013).

[5] V Giampietro M. Detyna R Sanchez−Pizani and E.J. Dommett. “Hybrid flexible (HyFlex) teaching and learning: climbing the mountain of implementation challenges for synchronous online and face−to−face seminars during a pandemic”. In: (2022). URL: https://doi.org/10.1007/s10984-022-09408-y.

[6] P.C. Blumenfeld J.A. Fredricks and A.H. Paris. “School Engagement: Potential of the Concept, State of the Evidence.” In: (2004). URL: https://doi.org/10.3102/00346543074001059.

[7] Annelies Raes. “Exploring Student and Teacher Experiences in Hybrid Learning Environments: Does Presence Matter?” In: (2021). URL: https://link.springer.com/content/pdf/10.1007/s42438-021-00274-0.pdf.

[8] Jo Ann Lee & Patricia Ellen Busch. “Factors Related to Instructors Willingness to Participate in Distance Education.” In: *The Journal of Educational Research* (2005). URL: https://doi.org/10.3200/JOER.99.2.109-115.

[9] *The Netherlands ranks among the EU top in Digital Skills*. Feb. 2020. URL: https://www.cbs.nl/en-gb/news/2020/07/the-netherlands-ranks-among-the-eu-top-in-digital-skills.

[10] Dosheela Devi Ramlowat and Binod Kumar Pattanayak. “Exploring the Internet of Things (IoT) in Education: A Review”. In: *Information Systems Design and Intelligent Applications*. Ed. by Suresh Chandra Satapathy et al. Singapore: Springer Singapore, 2019, pp. 245–255. ISBN: 978-981-13-3338-5.

[11] Sciforce. “Internet of Things for the Classroom”. In: (2019). URL: https://www.iotforall.com/internet-of-things-classroom.

[12] *Logitech Rally Kit(960-001218) Datasheet*. https://cdn.webshopapp.com/shops/91456/files/319561589/logitech-rally-product-data-sheet.pdf. Accessed: 30/05/2022.

[13] *vc520pro2 datasheet*. https://www.averusa.com/business/downloads/vc520pro2-datasheet.pdf. Accessed: 30/05/2022.

[14] *A BRIEF GUIDE TO AUDIO CONFERENCING IN HIGHER ED CLASSROOMS*. https://www.parmetech.com/wp-content/uploads/2021/09/Guide-to-Audio-Conferencing_-Classrooms_-Education_-Cobranded.pdf. Accessed: 30/05/2022.

[15] *Chung Yuan Christian University Establishes World’s First ViewSonic Hybrid Teaching Classroom*. https://www.viewsonic.com/nl/education/pdf/Case_Study_Chung_Yuan_Christian_University_Establishes_Worlds_First_ViewSonic_Hybrid_Teaching_Classroom.pdf. Accessed: 30/05/2022.

[16] Mirjana Maksimovic et al. “Raspberry Pi as Internet of Things hardware: Performances and Constraints”. In: June 2014.

[17] Maurizio Capra et al. “Edge Computing: A Survey On the Hardware Requirements in the Internet of Things World”. In: *Future Internet* 11.4 (2019). ISSN: 1999-5903. DOI: 10.3390/fi11040100. URL: https://www.mdpi.com/1999-5903/11/4/100.

[18] *Raspberry Pi 3 Model B Datasheet*. URL: https://www.alliedelec.com/m/d/4252b1ecd92888dbb9d8a39b536e7bf2.pdf.

[19] Anne H. Ngu et al. “IoT Middleware: A Survey on Issues and Enabling Technologies”. In: *IEEE Internet of Things Journal* 4.1 (2017), pp. 1–20. DOI: 10.1109/JIOT.2016.2615180.

[20] Andrea Azzarà et al. “Middleware solutions in WSN: The IoT oriented approach in the ICSI project”. In: *2013 21st International Conference on Software, Telecommunications and Computer Networks - (SoftCOM 2013)*. 2013, pp. 1–6. DOI: 10.1109/SoftCOM.2013.6671886.

[21] *TinyDB Documentation*. URL: https://tinydb.readthedocs.io/en/latest/index.html.

[22] *Tkinter Documentation*. URL: https://docs.python.org/3/library/tkinter.html.

[23] *Python Tkinter*. https://www.educba.com/python-tkinter/. Accessed: 03/06/2022.

[24] *Threading Documentation*. URL: https://docs.python.org/3/library/threading.html.

[25] *Asyncio Documentation*. URL: https://docs.python.org/3/library/asyncio.html.

[26] *Multiprocessing Documentation*. URL: https://docs.python.org/3/library/multiprocessing.html.

[27] *Quart Documentation*. URL: https://pgjones.gitlab.io/quart/.

[28] *M5Stack Stamp-C3 board specifications*. URL: https://docs.m5stack.com/en/core/stamp_c3.