CLIMATE MONITORING AT HOME

MONITOROVÁNÍ PROSTŘEDÍ V DOMÁCNOSTI

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR
AUTOR PRÁCE

ANDREJ PILLÁR

SUPERVISOR
VEDOUCÍ PRÁCE

Ing. PETR ŠOUSTEK

BRNO 2020

Specification Bachelor's Thesis

Department:	Institute of Automation and Computer Science
Student:	**Andrej Pillár**
Study programme:	Engineering
Study branch:	Applied Computer Science and Control
Supervisor:	**Ing. Petr Šoustek**
Academic year:	2019/20

Pursuant to Act no. 111/1998 concerning universities and the BUT study and examination rules, you have been assigned the following topic by the institute director Bachelor's Thesis:

Climate Monitoring at Home

Recommended bibliography:

MONK, Simon. Programming Arduino: getting started with sketches. New York: McGraw-Hill, c2012. ISBN 978-0071784221.

Deadline for submission Bachelor's Thesis is given by the Schedule of the Academic year 2019/20

In Brno,

L. S.

doc. Ing. Radomil Matoušek, Ph.D.	doc. Ing. Jaroslav Katolický, Ph.D.
Director of the Institute	FME dean

Abstract
This work is focused on designing, implementing and testing a device which could provide an overview of indoor environmental conditions at a glance. Environmental sensors for temperature, humidity, pressure and CO_2 concentration are used for this device. In addition to providing immediate overview on a built in display, the device is able to communicate the measured data via MQTT; an Internet of Things protocol. A server solution for this purpose is also a part of this work. The server stores the data and provides means of accessing it. Data is saved in an SQLite database and accessible via a JSON API built with a Python web micro framework. A web application built on several web technologies provides an overview of both the latest and historic values. This solution can then be hosted on an http server. The reference implementation runs on the NGINX web server which also facilitates secure communication over TLS for all services. The entire solution is implemented with freedom, modularity and extensibility in mind. Libre and Open Source technologies are leveraged wherever possible.

Abstrakt
Cieľom tejto bakalárskej práce je vyvinúť, zostaviť a otestovať zariadenie na meranie enviromentálnych veličín v interiéri. Zariadenie obsahuje senzory teploty, relatívnej vlhkosti, tlaku a koncentrácie CO_2. Aktuálne hodnoty su zobrazené na vstavanej obrazovke spolu s časom a odosielané pomocou IoT protokolu MQTT na server. Práca obsahuje aj riešenie tohoto serveru. Dáta sú ním spracovávané, ukladané do SQL databázy a ďalej sprístupnené cez JSON API implementované v jazyku Python. Aplikácia vyvinutá pomocou niekoľkých webových technológií ponúka prehľad aktuálnych aj historických hodnôt. Serverové riešenie je hostované na http serveri. Demonštračná verzia riešenia využíva server NGINX ktorý tiež poskytuje zabezpečenie komunikácie so serverom pomocou TLS pre všetky služby. Celá práca je spracovaná s dôrazom na otvorenosť, modularitu a rozšíriteľnosť. V práci sú preferenčne používané Libre a Open Source technológie.

Keywords
IoT, Arduino, MQTT, Python, indoor environment, CO2, sensors, monitoring

Klíčová slova
IoT, Arduino, MQTT, Python, domácnost, prostředí, CO2, senzory, monitorování

PILLÁR, A. *Climate monitoring at home.* Brno: Brno University of Technology, Faculty of Mechanical Engineering, 2020. 52 s. Supervisor Ing. Petr Šoustek.

Rozšířený abstrakt
Obsahom tejto práce je návrh, spracovanie a testovanie zariadenia na meranie enviromentálnych podmienok v interiéri. Merané veličiny sú teplota, relatívna vlhkosť, tlak a koncentrácia CO_2. Zariadnie je založené na platforme Arduino, konkrétne 8-bit mikroprocesore ATmega 2560. Prehľad aktuálnych hodnôt je dostupný na vstavanej e-ink obrazovke a zároveň sú dáta odosielané na server cez IoT protolkol MQTT. Pripojenie na internet je realizované cez Wi-Fi pomocou modulu ESP8266. Arduino a Wi-Fi modul komunikujú cez asynchrónnu sériovú linku. Použité senzory sú Bosch-Sensortec BME280 merajúci teplotu, relatívnu vlhkosť a tlak, a Senseair LP8 merajúci koncentráciu CO_2. Senzor CO_2 je súčasťou integrovaného modulu od firmy Hardwario. Tento modul obsahuje všetky podporné komponenty potrebné na fungovanie senzoru a zároveň uľahčuje komunikáciu s ním. Dodávaný software je ale použiteľný len pre ARM mikroprocesory a teda musel byť upravený pre použitie na platforme Arduino. Senzor BME280 komunikuje s procesorom na zbernici I^2C, obrazovka používa protokol SPI a CO_2 modul je pripojený cez sériovú linku. Arduino pracuje s napätím 5 V zatiaľ čo periférie pracujú s napätím 3.3 V. Bezpečná komunikácia medzi zariadeniami je sprostredkovaná pomocou obojsmerných prevodníkov úrovní.

Spomínaný server na zber dát je tiež súčasťou práce. Pozostáva z niekoľkých služieb: MQTT broker, SQL databáza, JSON API a webová aplikácia poskytujúca prehľad. Ako MQTT broker bola zvolená implementácia Mosquitto, SQL databázu poskytuje knižnica SQLite3 a API je postavené na webovej mikroplatforme Flask v jazyku Python. Webová aplikácia je postavená na technológiách JavaScript a jQuery. Ako vyplýva z funkcie MQTT, prichádzajúce dáta sú ukladané pomocou Python skriptu ktorý dáta prečíta, skontroluje a zapíše do databázy. Taktiež sa stará o oznamovanie stavu meracieho zariadenia. JSON API poskytuje prístup k dátam z databázy. Prístupné je cez GET a POST requesty s jedným argumentom ktorým je počet dátových bodov. Dáta vracia vo formáte JSON. Webová aplikácia využíva práve toto API. Zobrazuje aktuálne hodnoty a graf historických hodnôt s voliteľným rozsahom. Grafovacie funkcie poskytujú knižnice JustGage a Chart.js. Aplikácia taktiež disponuje responzívnym štýlom čo jej umožňuje prispôsobiť svoj obsah veľkosti prehliadačového okna. Serverové riešenie je spustiteľné pod http serverom, v tejto práci je použitý server NGINX ktorý zároveň poskytuje zabezpečenie komunikácie pomocou TLS.

Celé riešenie je vyvinuté a postavené na Open Source technológiách, dostupné s Open Source licenciou a voľne upraviteľné a rozšíriteľné. Samotná konštrukcia zostáva modulárna. Skrinka zariadenia je vyrobená pomocou 3D tlače z PLA plastu. Kompletný obsah riešenia je dostupný v git repozitári.

I hereby declare that the bachelor's thesis *Climate monitoring at home* was prepared as an original author's work under the supervision of Ing. Petr Šoustek. All the relevant information sources which were used during preparation of this thesis, are properly cited and included in the list of references.

Andrej Pillár

Contents

1 Introduction

Comfortable conditions in a living or working space greatly influence people's ability to focus and perform work efficiently in general. Changes in temperature, too low or too high humidity or poor airflow causing an increase in CO_2 concentration have an impact on people's performance and comfort. Therefore it is advisable to keep track of these conditions and act accordingly to keep them at comfortable levels. The aim of this work is to provide a tool, a monitoring device that could serve this purpose. This device is to be built on an 8-bit microcontroller platform. In addition to direct feedback via an integrated screen, it will feature an internet connection to communicate measured data to a server via an IoT protocol. The internet connection will be facilitated by a separate WiFi capable microcontroller. Firmware for both devices will be written in C++ using the Arduino IDE. The environmental conditions monitored are: temperature, humidity, pressure and CO_2 concentration. An RTC module will take care of accurate time keeping.

The solution will include a server for storing and analyzing the gathered data. The server will provide connection facilities for the device, a database backend, a JSON API and a web visualization interface. The core server functionality will be developed on a Python platform. Data from the database will be accessible via a simple JSON API. A web application utilizing several web development libraries will present the most recent values and a plot of historical values.

The entire solution is to be provided under a permissive Open Source license allowing for complete control and ability to customize or extend the solution. Open Source technologies, both software and hardware will be leveraged on every level of the solution.

2 Indoor environmental conditions

Environment of indoor spaces is subject to multiple conditions and variables. Traditionally, the level of comfort is defined by ambient temperature, humidity and “freshness” of air. This chapter describes the relevant measurable parameters, including methods of measurement and their respective sensors.

2.1 Temperature

Temperature is the immediately obvious and probably the most significant factor in indoor environment quality. While dependent on the most external factors such as seasonal conditions, air drafts or occupant activity, regulating it is a straightforward process. In the context of environment quality, the best indicator is the ambient temperature level. Keeping this temperature in a comfortable range is vital to ensuring a quality living or working environment. While the exact value that can be called comfortable differs between people, it is generally recommended to keep the ambient temperature at around 20 degrees Celsius. Taking seasonal conditions into account, the recommended ranges are 20–23.5°C for the winter or heating season and 23–25.5°C for the summer season [21].

There are many available methods and devices for temperature measurement. From simple ones such as a mercury thermometer or a bimetallic strip, through analog devices measuring changes in electrical resistance, to more recent specialist integrated circuits. Due to nature of this work, the focus is on the latter category.

2.2 Relative humidity

Relative humidity is the amount of water vapor in the air relative to its absolute capacity at a given temperature. The level of water vapor in the air has an effect on person’s response to temperature. Different levels of relative humidity affect the comfortable temperature range on both ends. This is because of the effect relative humidity has on the human body’s ability to regulate its temperature through water evaporation. Higher relative humidity makes air of the same temperature feel warmer and at levels above 70 percent even uncomfortable. In addition to decreasing overall comfort, higher relative humidity often results in increased presence of biocontaminants such as molds, fungi or mites. The range of relative humidity for the most comfortable and productive environment was established at 30 to 60 percent with temperature also in the recommended range [22].

A device for measuring humidity is known as a hygrometer. Attempts at measuring air humidity began in the ancient times with primitive methods such as comparing the weight of a highly absorbent material like charcoal before and after exposing it to the measured air. Later, hair tension hygrometers were introduced, leveraging a properly treated and tensioned hygroscopic material such as a human or animal hair. Length of the hair changed with the amount of moisture and this movement driven a needle indicating an approximate value on a dial. This type of hygrometer is dependent on condensation of moisture [3]. Modern hygrometers are either electrical or optical. Electrical hygrometers further fall into two categories: capacitive or resistive. The former measure the effects of humidity on the capacitance of a polymer or metal oxide capacitor while the latter measure changes in resistance of a metal oxide or polymer based resistor [49]. Optical hygrometers function as spectrometers, measuring absorption of a certain wavelength of light by the water contained in the measured air [5]. There is one other type of hygrometer in use today: a volumetric hygrometer. Volumetric hygrometers function by measuring the weight ratio of an air sample and a sample of dessicated air of equal volume [37]. This type of hygrometer is usually used only when calibrating the other less precise hygrometers due to the inconvenience of its use.

2.3 Atmospheric pressure

The pressure of the atmosphere. Decreases with rising altitude from the sea level where it reaches it's peak value standardized at 101325 Pa or 1013.25 hPa as it is usually measured [46]. Besides altitude, variations of atmospheric pressure are most often caused by changing weather conditions. While having a direct effect on human comfort and environment in general, there is little that can be done about it. It is however influenced by temperature and humidity and in turn affects the other measured property, the CO_2 concentration, so it needs to be taken into account in the context of this work.

The device for measuring atmospheric pressure is called a barometer. Its invention is attributed to the Italian physicist Torricelli in the 17th century. The first barometers functioned on the principle of a column of mercury with an open reservoir. The pressure exerted on the exposed liquid would force the liquid up the column and the measurement would be made by reading the height of the column [4]. Later, so called aneroid barometers not containing any liquid but rather a metal cell with a vacuum inside were used. The cell, held by a spring to prevent it from collapsing, would expand or contract with changes in pressure of the surrounding air. This movement would then drive an indicator needle by a series of levers and gears [2]. With the advent of photolitography, the MEMS technology allowed for miniaturization of the sensors and their inclusion in tiny IC packages. The sensor used in this work belongs to the last category.

2.4 CO_2 concentration

Any human activity in an enclosed space causes accumulation of the main byproduct of breathing: carbon dioxide or CO_2. Its concentration is therefore a good indicator of ventilation quality even though it is not considered a primary air contaminant. Poor air quality has arguably the largest impact on human activity. People who spend a prolonged amount of time in spaces where CO_2 concentration exceeds the natural background levels of 300-450 ppm by 2-3 times have been observed to suffer headaches, fatigue, and eye and respiratory tract irritation. At concentrations above 5000 ppm, CO_2 is considered a health hazard. An indoor environment is considered well ventilated if the CO_2 concentration does not exceed values of 700 ppm above background [23].

CO_2 sensors can be split into two main categories based on the principle of measurement: Non Dispersive Infrared or NDIR and heated electrode. The first category in principle functions as a prismless (hence the non-dispersive attribute) spectrometer. The measured air enters a chamber with defined optical properties and infrared light is shined through the chamber in a short, controlled pulse. A photodiode also placed in the measurement chamber captures this flash. The resulting concentration is then determined by the amount of a particular wavelength of light absorbed by the air within the chamber. The other category of sensors, based on metal oxide technology, is not capable of direct measurement of CO_2. The measured value is instead calculated as a CO_2 equivalent by measuring concentrations of other volatile organic compounds (VOC) in the air. Sensors of this technology are also cross-sensitive to humidity and temperature fluctuations. They perform the best in higher concentrations of above 2000 ppm [52]. These sensors are substantially cheaper and more available, however. While also capable of other measurements of potential interest, such sensor was not included due to its limited lifespan and relatively high power consumption. Instead, a more accurate NDIR technology sensor was chosen. The solution's architecture, however, allows for the addition of a VOC sensor either by the user or in further revisions should the need arise.

2.5 Sensors

In this section, several environmental sensors are compared and evaluated on the basis of applicability in the solution. The main evaluated parameters are: measured properties, accuracy, power consumption, the sensor's interface, and availability. Focus is primarily on sensors based around integrated circuits measuring multiple properties as integrating analog devices or multiple sensors would needlessly complicate the solution.

Temperature, humidity and pressure sensors

Most IC based sensors available today are capable of measuring at least two of the mentioned properties. Of the widely available hobbyist offerings the following environmental sensors were considered: AM2320, AM2302, Bosch BMP280 and Bosch BME280. A brief comparison of their properties is presented in Table 2.1.

Table 2.1: Comparison of available environmental sensors [6–9, 12, 19, 20, 25].

Model	AM2320	AM2303	BMP280	BME280
Manufacturer	Aosong	Aosong	Bosch Sensortec	Bosch Sensortec
Operating voltage [V DC]	3.3–5.5	3.3–6	1.8–3.3	1.8–5
Operating current [µA]	950	1500	3.4	3.6
Quiescent current [µA]	10	N/A	0.1	0.1
Sampling rate [Hz]	0.5	0.5	157	182
Temperature range [°C] (Full accuracy range)	-40–80	-40–80	-40–85 (0-65)	-40–85 (0–65)
Temperature accuracy [±°C]	0.5	0.5	0.5	1
Temperature resolution [°C]	0.1	0.1	0.01	0.01
Humidity range [% RH]	0–100	0–100	N/A	0–100
Humidity accuracy [±% RH]	3	5	N/A	3
Humidity resolution [% RH]	0.1	0.1	N/A	0.008
Pressure range [hPa]	N/A	N/A	300–1100	300–1100
Pressure accuracy [±hPa] (absolute)	N/A	N/A	0.12 (1)	0.12 (1)
Interface	I^2C	1-wire	I^2C	I^2C or SPI
Price [CZK]	97	147	62	199

The chosen sensor is the Bosch Sensortec BME280. While not the most accurate, the fact that the sensor combines all necessary measuring facilities in one low power package was the deciding factor. A combination of a temperature + humidity sensor and a separate pressure sensor would be possible but highly impractical due to the increased space and power requirements. Not to mention the increased code complexity due to the need to include separate libraries for the sensors.

CO_2 sensors

The other important environmental sensor for this work is the CO_2 concentration sensor. Compared to the previous sensor category, these sensors are not nearly as common. This basically ruled out any selection criteria based on price, the focus here was primarily on availability and reasonable accuracy. An overview of the considered sensors is presented in Table 2.2.

Table 2.2: CO_2 sensor comparison [1, 31, 50, 53, 54, 62].

Model	MH-Z19	SCD30	LP8
Manufacturer	Winsen	Sensirion	Senseair
Operating voltage [V DC]	3.6–5.5	3.3–5.5	2.9–5.5
Operating current [mA] @ $\frac{1}{60}$Hz sampling rate	18	19	0.66
Sampling rate [s] minimum	60	2	16
Range [ppm] (extended range)	0–2000 (0–5000)	0–10000 (0–40000)	0–2000 (0–10000)
Accuracy [±ppm +% reading] (extended range)	50+3	30+3	50+3(10% reading)
Interface	serial, pwm	serial, I^2C	serial (modbus)
Price	18.6 US$	59.95 US$	111.70 EUR (module)

The LP8 sensor was chosen because of its immediate availability as an integrated module. The SCD30 would be a preferred choice for its temperature and humidity measurement capabilities that would make it an all-in-one solution. The sensor was however unavailable to the author as of the time of writing. The Chinese MH-Z19 sensor would also be a usable alternative if it were available.

Other sensors

An air particulate sensor Sharp GP2Y1010AU0F was also sourced for this work. The sensor was however found to be not sensitive enough for typical indoor use during testing. This sensor is therefore not included in the final implementation of the solution.

3 IoT connectivity protocols

The devices comprising the ever evolving landscape of the Internet of Things need to communicate in order to be useful. This communication is facilitated by either a wireless or wired network and a messaging protocol. Those are not unlike the messaging protocols used in communication programs for people, such as XMPP, only they are used between machines. Over the years, device manufacturers and designers either implemented their own or re-purposed an existing protocol. These protocols are usually employed in applications with limited resources and are therefore optimized for low bandwidth use. A multitude of such more or less featured application layer protocols is available. For the purpose of this work, two pub/sub messaging protocols were considered: *MQTT* and *OPC UA*. The following sections contain a closer look at the protocols.

3.1 MQTT

One of the most prominent protocols in use today for consumer IoT devices is MQTT. MQTT stands for MQ Telemetry Transport, though it is sometimes incorrectly referred to as Message Queuing Telemetry Transport [39]. The name was historically derived from the "MQ" line of IBM messaging middleware products. The protocol runs over TCP/IP, but any lossless bidirectional connection can support it [14]. The protocol provides publish-subscribe messaging but no queuing despite the often used name. The MQTT protocol was originally created by IBM in the year 1999 for the purpose of monitoring an oil line running through the desert using little resources [58]. The entire design is focused on low bandwidth high latency applications. In 2013, the protocol was submitted to the *Organization for the Advancement of Structured Information Standards* (OASIS) for standardization. It is also standardized by *International Standards Organization* (ISO) as ISO/IEC 20922 [33]. A separate specification called MQTT-SN meaning MQTT Sensor Network aimed at non-TCP/IP networks also exists. In this version the protocol can be used over other transports such as UDP, Bluetooth or zigbee [55]. The MQTT protocol was standardized in version 3.11 and revised in 2019 to version 5.0 [14], which extended the functionality with better error reporting and the addition of message metadata among other things [15, 57].

The MQTT protocol is used for communication between two network entities: a broker and an arbitrary number of clients. A broker is a server that receives all messages from connected clients and routes the messages to the appropriate destination clients. A client is any device that has an MQTT library available and connects to an MQTT broker [56]. Clients and brokers can be a wide range of devices; from a microcontroller or a single board computer to a workstation or server. The messages are organized by topics. Topics can be freely available to any connected client or secured with a set of credentials. Clients can subscribe to one or several topics. When a client publishes data to a topic, it sends a message to the connected broker and the broker in turn distributes the message to any clients currently subscribed to the topic [59]. This way none of the clients need to have any information about the number or location of the other clients and the broker does not need to be configured in any way in order to receive data from the publishers. A network may contain multiple brokers allowing for broker load balancing. If no clients are subscribed to a topic, any messages published to the topic will simply be discarded. The protocol specifies an optional parameter allowing the message to be retained until a client subscribes but the broker only stores a single most recent message per topic. This is useful for topics with multiple subscribers with intermittent connections as it allows them to receive the latest data without the need to wait for the publisher [60]. MQTT also defines three levels of Quality of Service (QoS), which control how hard the broker tries to deliver a message. The higher the QoS level, the more latency is introduced and more bandwidth required. The three levels are defined as follows:

- QoS 0: deliver once, no confirmation (fire and forget)
- QoS 1: deliver at least once, confirmation required
- QoS 2: deliver exactly once, using a handshake [38]

Since the data communicated from the measuring device is of no vital importance, only QoS 0 messaging is used in this work. An MQTT message can be as small as two bytes and a maximum message size can be communicated between the client and broker using a control message [16]. There are 14 defined control messages for connecting to and disconnecting from the broker, data publishing and acknowledging and connection supervision [17]. MQTT sends all data including connection credentials in plain text. Transport security can be however added in the underlying TCP protocol. This is known as MQTT over TLS or MQTTS. In this work, this was achieved using the reverse proxy feature of the NGINX web server and a Let's Encrypt TLS certificate [34].

3.2 OPC UA

Another machine to machine messaging protocol considered for this work is OPC UA. Developed by the OPC foundation, it is aimed primarily at industrial applications for device communication and data collection [43]. The name stands for *Open Platform Communications Unified Architecture*. While developed by the same organization, OPC UA is significantly different from the previous OPC protocol [40]. The main differences are the openness of the protocol and its ability to be used on multiple platforms and even scaled down to very low power devices [41]. This is an improvement from the previous MS Windows only protocol, implementations of which were often incomplete owing to its black box nature. The new protocol is built as a Service Oriented Architecture and offers security functionality for authentication and data integrity [45]. The data security features include redundancy support, data acknowledgments and buffering meaning that a lost datagram can be reconstructed. The specification was released in the year 2008 [41] and the latest version 1.04, in 2017. The latest major revision from 2018 includes a pub/sub model in addition to the classic client/server [44]. The specification of the protocol is very complex, comprising of 14 core documents and over 1200 pages total [42]. APIs for several languages including Python are available. Due to the project's far reaching coverage of automation needs and therefore its inherent complexity, the existing implementations are usually focused on a certain task. This collection of focused implementations due to the possibility to selectively implement a functionality e.g. only the pub/sub functionality, means a diverse environment that is not easy to navigate. While also possible to deploy on a variety of platforms and devices utilizing either free or commercial toolkits, the protocol's complexity makes it an inferior choice for a simple application such as the one presented in this work.

4 Implementation

This chapter describes the reference implementation of the device and related components. It is split into four sections. The first is dedicated to the hardware comprising the monitoring device itself, the next describes the device firmware, the third section is concerned with the server implementation and the last with the device assembly. The device is based on an Arduino platform, in particular the Mega 2560. An ESP8266 chip provides WiFi connectivity. The sensors and other peripherals are connected using multiple protocols including serial, I^2C and SPI. The device is also equipped with a 2.9" e-ink screen to provide information to the user at a glance. The server provides an MQTT broker, an SQL database, a JSON API and a web front-end for data visualization. A block diagram in Figure 4.1 showing the hardware, services and protocols provides an overview of the solution.

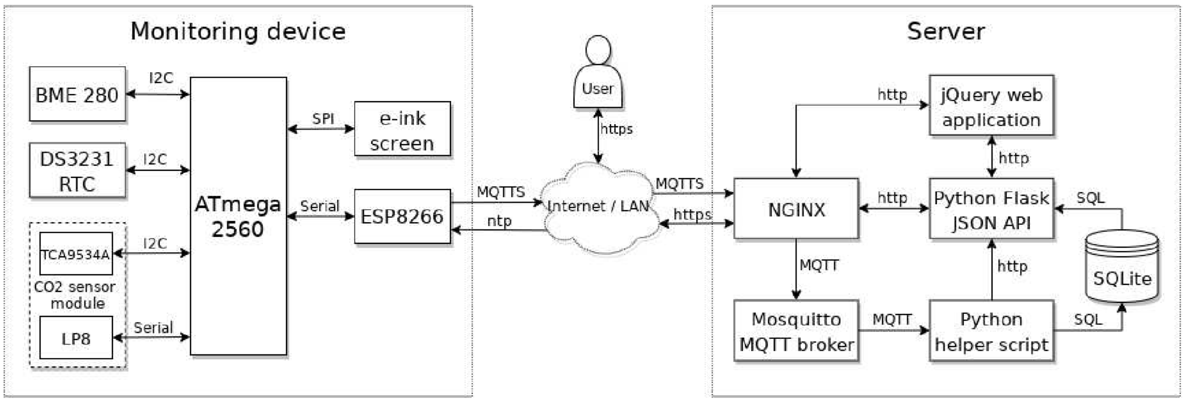


Figure 4.1: Reference implementation block diagram.

4.1 Hardware

4.1.1 Arduino Mega 2560

The Arduino platform provides a wide range of ready to use accesible development boards primarily based on AVR 8-bit microcontrollers. For this work, an ATmega2560-based board was chosen. This decision was primarily driven by the chip's broad array of communication facilities. In particular the 4 hardware serial lines, which are required for communicating with the ESP chip, the CO_2 sensor and, optionally, with the user's computer. The ability to connect to the board via serial also proved useful during development and debugging stages. The next driving factor was the chip's large SRAM of 8 KB, which is beneficial to the system's stability. Particularly when handling display drawing routines and processing the relatively large amount of data from the sensors. The original board can be seen in Figure 4.2. The board's technical specification as per the manufacturer is presented in Table 4.1 below. A customized miniature version of the board, shown in Figure 4.3, is used in the final product in order to save space while retaining full functionality. Specifications of the custom board differing from the original are also written in Table 4.1 in parentheses. The board's firmware is written in C++ using the freely available Arduino IDE. More on the firmware in the following sections.

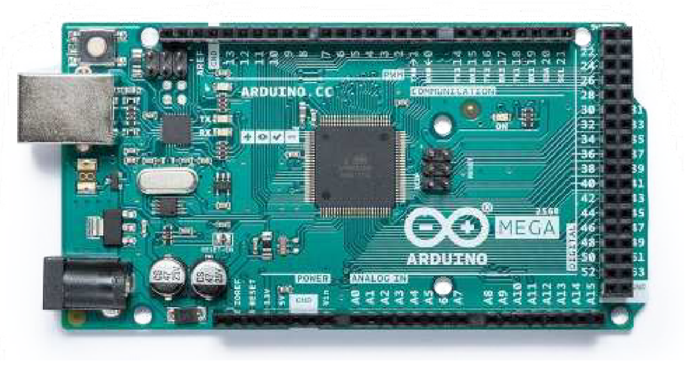


Figure 4.2: Original Arduino Mega 2560 board [13].

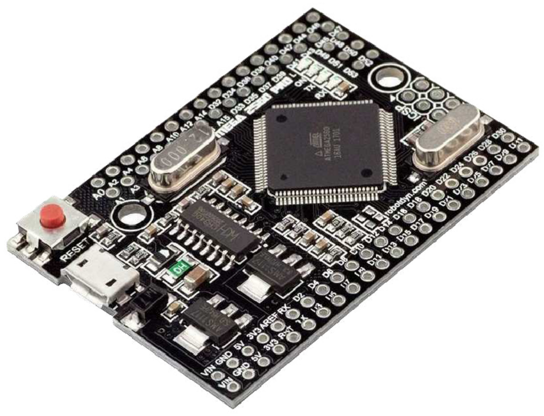


Figure 4.3: Custom miniature board by RobotDyn [48].

Table 4.1: Arduino Mega 2560 - technical specifications [13] [48].

Microcontroller	ATmega2560
Operating Voltage	5 V
Input Voltage (limit)	6-20 V
Digital I/O Pins	54 of which 15 provide PWM output
Analog Input Pins	16
DC Current per I/O Pin	20 mA
Flash Memory	256 KB
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz
Physical dimensions	
Length	101.52 mm (54 mm)
Width	53.3 mm (38 mm)
Weight	37 g (21 g)

4.1.2 ESP-01

Another important piece of hardware is the by now ubiquitous ESP8266 WiFi capable chip by the Chinese company Espressif. It is available on a wide variety of boards in many different form factors. The ESP-01 variant, shown in Figure 4.4, was chosen for this work. It is a small form factor board with a printed antenna and an 8-pin 2x4 DIL header. Since the board is utilized here for its wireless connectivity only, the limited I/O does not pose a problem. In addition to power connections, the header exposes a hardware serial line, two GPIOs that are also used to select boot modes, a reset pin and a chip enable pin. The reset pin is connected in series with the Arduino's so that both boards can be reset at the same time with a single button. The module is programmed in the same Arduino IDE as the Mega thanks to the ESP8266 Arduino Core [28].

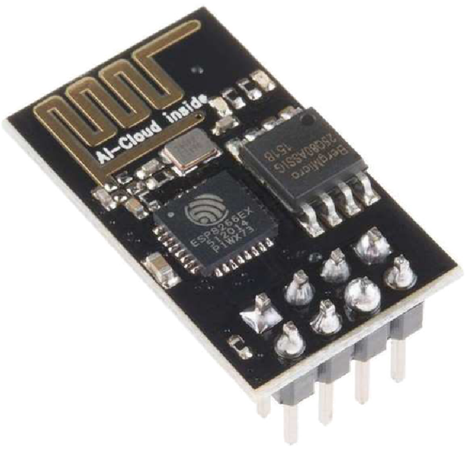


Figure 4.4: The ESP-01 module [10].

Table 4.2: ESP-01 - technical specification [29].

Processor	Tensilica L106 32-bit RISC core
Operating voltage	2.5 - 3.6 V
Flash memory	1 MB external QSPI
Operating memory	32 KB instruction
	80KB user data
Clock speed	80 MHz
Physical dimensions	
Length	24.8 mm
Width	14.3 mm

4.1.3 Sensors and peripherals

Bosch-Sensortec BME280

This integrated circuit provides the device with the ability to measure temperature, humidity and pressure. The sensor communicates on I^2C bus and the chosen module, shown in Figure 4.5, allows operating voltages between 1.8 and 5 V by including an LDO on the breakout board. As mentioned earlier in Section 2.5, the sensor features a low operating and quiescent current, a relatively low measuring cycle of 1 second full accuracy and an accuracy of ±1 °C, ±3 % RH, and ±100 Pa as per Table 2.1.

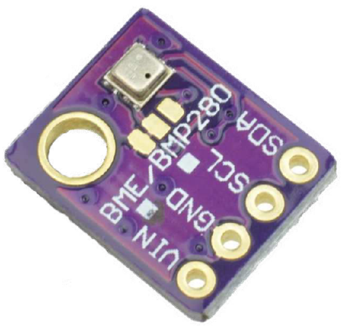


Figure 4.5: A BME280 sensor on a breakout board [8].

Hardwario CO_2 sensor module (Senseair LP8)

The sensor of choice for CO_2 measurement is the LP8 NDIR sensor by the Swedish company Senseair. As utilized in the work, the sensor is part of a module. The module, depicted in Figure 4.6, is designed and manufactured by the Czech company Hardwario. It includes all the necessary support components for the sensor to operate and several extra ICs to enable integration into their IoT module lineup. The module and all associated firmware is released under a permissive MIT Open Source license.

The sensor requires a constant current source with a supercapacitor for the IR lamp as well as switching the supply voltages on and off at specific times during the measurement period so having all of these facilities available on a module is very helpful. The module includes a TCA9534A I^2C IO expander and an SC16IS750 I^2C serial bridge. Since the mega microcontroller includes a hardware serial, connection to the sensor is made directly, bypassing the serial bridge. Thankfully, the module's layout includes such option and the serial pins are even broken out in a pin header [18]. Only a slight modification to the board by replacing two links is required for the direct connection. This is illustrated in Figure 4.7 where the two 0 Ohm links are moved to the new position. Also evident in the figure is the factory bodge to add pullup resistors to the serial line. This was fixed in a later revision of the board layout. While the mega has enough available I/O pins to handle the voltage rail switching and pin polling, the onboard I^2C IO expander was used instead as it allowed for a much neater implementation, not to mention the hardware modification required for bypassing it would have to be rather extensive. The sensor's manufacturer provides a very detailed datasheet and an application guide that proved extremely useful while writing a measurement routine for the sensor.

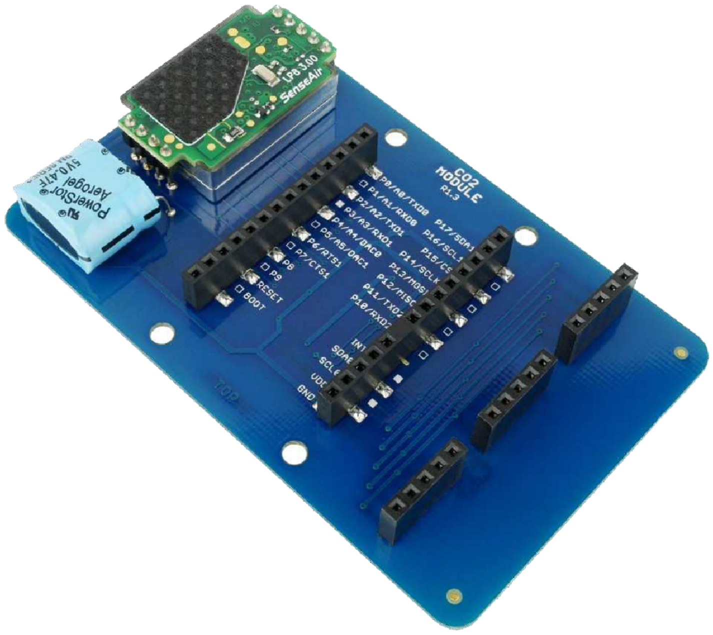


Figure 4.6: The Hardwario CO_2 sensor module [31].

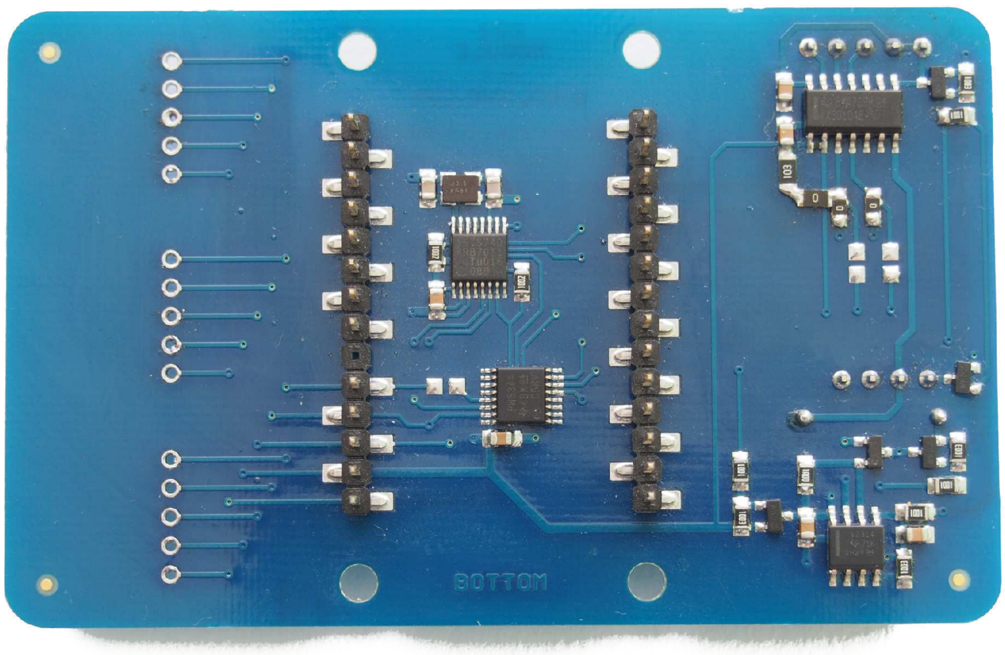


Figure 4.7: Bottom side of the CO_2 sensor module.

DS3231 real time clock

For the purpose of keeping accurate time even while powered off or used off-line, an RTC module is used. The chosen module, shown in Figure 4.8, is based around a DS3231 chip. Besides basic time and date keeping, the chip offers the ability to count days in months with less than 31 days, a leap year correction, two configurable alarms and both 12 and 24 hour time formats. The chip provides excellent time accuracy and stability thanks to a robust package and its ability to measure temperature and therefore compensate for the temperature drift of its oscillator [36]. The module communicates on the I^2C bus and accepts voltage levels between 3.3 and 5 V, which makes it easy to integrate. Also present is a 32K EEPROM and pins for square wave and 32 kHz oscillator output but they are not utilized in this work. The module includes a power loss backup by means of a button cell battery and facilities to recharge said battery. While practical, this arrangement calls for a rechargeable LIR2032 lithium cell. Since such cell was unavailable to the author at the time of writing, a simple modification to the module by removing a charging resistor allowed the safe use of a standard non rechargeable CR2032 battery. No difference in module functionality was observed throughout the testing period.

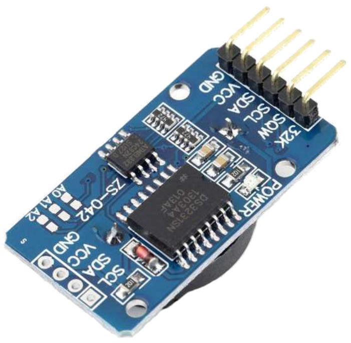


Figure 4.8: The DS3231 RTC module [11].

Waveshare 2.9” e-ink screen

The device includes a screen to display time and current measurement values. The chosen display is based on an e-ink technology. The display was chosen for its very low power consumption. It only consumes power when redrawing its contents and very little power at that. Another driving factor was the absence of any sort of backlight. This greatly lowers visual disturbance caused by the device while providing much more detail than what would be possible with a segmented LCD. One slight downside is that the screen retains its contents even while powered down, which might cause confusion around the state of the device. The power LED on the MCU board however remains visible and can serve as a status indicator. The display operates on 3.3 V and is connected using SPI. The display can be seen showing a value overview and debugging symbols still as a part of the testing setup in Figure 4.9 below.

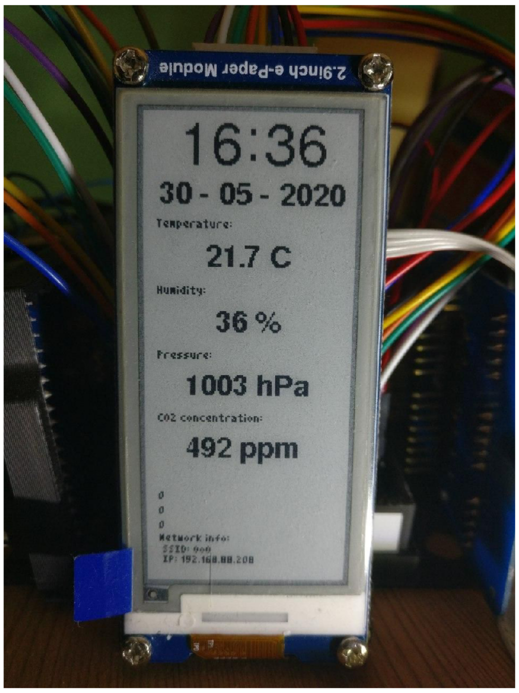


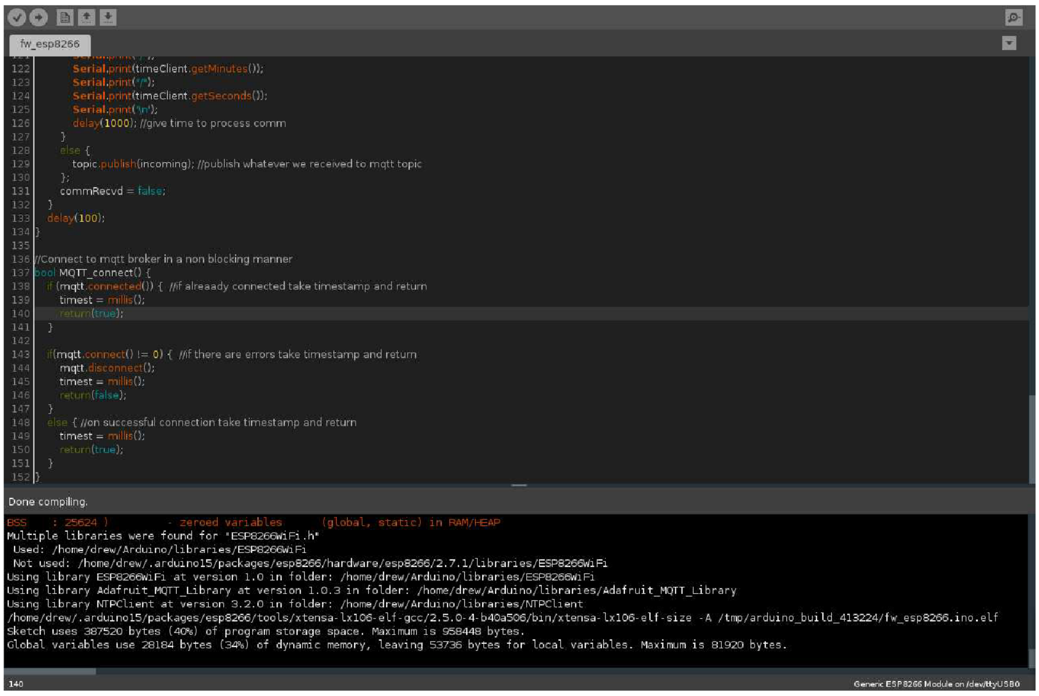
Figure 4.9: The e-ink display.

Breakout board

Since the mega microcontroller operates on a 5 V logic level and most other peripherals use 3.3 V, level shifters had to be utilized. What's more, the 3.3 V supply available on the Arduino board is only able to source 50 mA of current, which is not sufficient for the ESP8266 and other peripherals to operate. To solve this problem, a custom break out board was created. Besides having a space for connecting the level shiter boards and breaking out connections to peripherals into pin headers for modularity, the board houses a 3.3 V LDO capable of sourcing 1 A of current. This is more than a generous amount for all of the peripherals to run. The development version of the board also provided a micro-usb port passed through to a full-size connector and a switch for setting boot modes for the ESP8266 chip. The board was used throughout the development and testing period, the final construction of the device is different as described in Section 4.4

4.2 Firmware

As mentioned above, firmware for both the Arduino and ESP boards was written in C++ using the free and open source Arduino IDE. There are many useful libraries available for use in the Arduino environment covering a wide variety of sensors and other peripherals. Some of these libraries are utilized in this work while others, like in the case of CO_2 module, had to be written from scratch. Figure 4.10 shows a screenshot of the IDE window.



```
fw_esp8266
```

```
        Serial.print(timeClient.getMinutes());
        Serial.print("/");
        Serial.print(timeClient.getSeconds());
        Serial.print('\n');
        delay(1000); //give time to process comm
      }
      else {
        topic.publish(incoming); //publish whatever we received to mqtt topic
      };
      commRecvd = false;
    }
    delay(100);
}

//Connect to mqtt broker in a non blocking manner
bool MQTT_connect() {
  if (mqtt.connected()) {  //if alreaady connected take timestamp and return
    timest = millis();
    return(true);
  }

  if(mqtt.connect() != 0) {  //if there are errors take timestamp and return
    mqtt.disconnect();
    timest = millis();
    return(false);
  }
  else { //on successful connection take timestamp and return
    timest = millis();
    return(true);
  }
}
```

```
Done compiling.
BSS    : 25624 )         - zeroed variables      (global, static) in RAM/HEAP
Multiple libraries were found for "ESP8266WiFi.h"
 Used: /home/drew/Arduino/libraries/ESP8266WiFi
 Not used: /home/drew/.arduino15/packages/esp8266/hardware/esp8266/2.7.1/libraries/ESP8266WiFi
Using library ESP8266WiFi at version 1.0 in folder: /home/drew/Arduino/libraries/ESP8266WiFi
Using library Adafruit_MQTT_Library at version 1.0.3 in folder: /home/drew/Arduino/libraries/Adafruit_MQTT_Library
Using library NTPClient at version 3.2.0 in folder: /home/drew/Arduino/libraries/NTPClient
/home/drew/.arduino15/packages/esp8266/tools/xtensa-lx106-elf-gcc/2.5.0-4-b40a506/bin/xtensa-lx106-elf-size -A /tmp/arduino_build_413224/fw_esp8266.ino.elf
Sketch uses 387520 bytes (40%) of program storage space. Maximum is 958448 bytes.
Global variables use 28184 bytes (34%) of dynamic memory, leaving 53736 bytes for local variables. Maximum is 81920 bytes.
```

```
140                                                  Generic ESP8266 Module on /dev/ttyUSB0
```

Figure 4.10: The Arduino IDE.

4.2.1 Mega 2560

The ATmega mcu is in charge of polling the sensors, drawing on the e-ink screen and sending measurement data to the ESP chip for publishing. At its core, the firmware is relatively simple; consisting of a main loop that executes measurement, communication and drawing tasks. These tasks are triggered every minute by polling the RTC module for time change. The loop services a couple of other tasks such as periodic full screen refresh, date redrawing on midnight and periodic time synchronization. Apart from the Arduino native libraries for handling serial and I^2C communication, there are several 3rd party ones in use in this firmware: GxEPD2 [63] for the SPI e-ink screen drawing methods, MD_DS3231 [35] for controlling the RTC module and a library for configuring interfacing with the BME280 environmental sensor [30]. Methods provided by libraries are then usually wrapped in functions. The driver for the LP8 CO_2 sensor and associated circuitry was rewritten by the author for use within the Arduino environment. More details in Section 4.2.2.

The setup routine initializes the three serial lines required for interfacing as well as the I^2C system. Next it waits for the ESP chip to finish booting. Receiving network connection information (associated AP SSID and IP address), indicates that the ESP is ready to operate. The setup function then requests an ntp time update and processes the received information accordingly. Wait loop for the connection information times out after approximately 40 seconds and the MCU continues functioning normally without a network connection. Another initialization loop waits for a response from the BME280 sensor. If the sensor is not responsive, an error message is printed on the display. The device will finish booting anyway but as it cannot function properly without the sensor, the main loop will only update the clock. After processing connection and time information, the data is shown on the display. When the printing routine exits, the MCU is ready to execute its main loop, performing the functions described at the start of the section.

The following is a closer look at the measurement task. As soon as the time changes the measurements are taken. A function wraps the calls to measuring routines from an appropriate library and stores the results in a global struct. The struct in question can be seen in Listing 4.1. This is to simplify access to the measured data for the publishing function. An example of a measuring function is shown in Listing 4.2. Minimum sampling period for the BME280 sensor is one second so in order to balance out speed and accurate readings, a value is read every second for five seconds. To add at least some statistical value to the data, the final value is then calculated as an arithmetic average of the five samples. The float - integer conversions save a few clock cycles since the ATmega2560 does not incorporate a dedicated FPU.

Listing 4.1: Dataset structure

```
struct data {
    float temp;
    float hum;
    int rhum;
    float pres;
    unsigned int rpres;
    int16_t CO2;
};
```

Listing 4.2: Temperature measurement function

```
float measureTemp() {
    BME280::TempUnit tempUnit(BME280::TempUnit_Celsius);
    int temps[5];

    //light led while measuring
    digitalWrite(13, HIGH);
    Serial.print(F("Measuring temperature"));
    for(char i = 0; i < 5; i++){
        temps[i] = bme.temp() * 100;
        Serial.print(F("."));
        delay(1000);
    }
    float temp;
    temp = (temps[0] + temps[1] + temps[2] + temps[3] + temps[4]) / 5;
    Serial.print(F("done: "));
    Serial.println(temp / 100);
    //turn off the led
    digitalWrite(13, LOW);
    return(temp / 100);
}
```

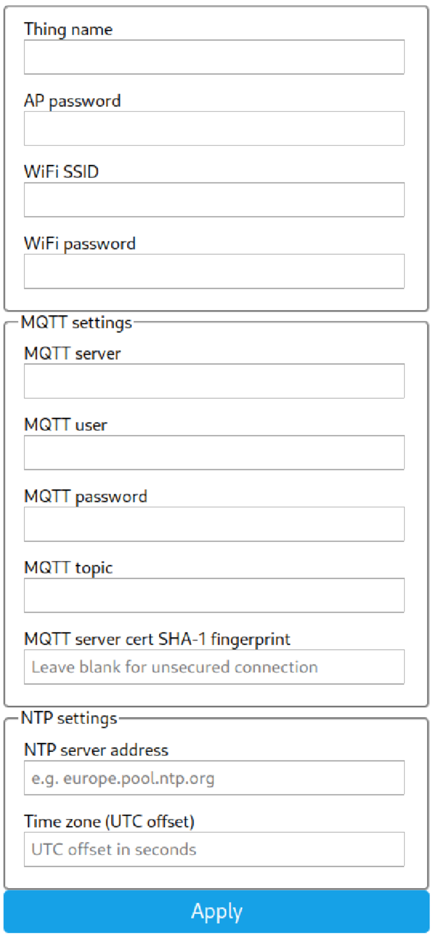
The CO_2 sensor has a sampling period of at least 16 seconds as per the manufacturer's application notes [51], so the CO_2 value is sampled only once per measurement cycle. This value however, should be accurate without averaging multiple samples thanks to built-in progressive filtering, compensation and adjustment algorithms. After each measurement cycle, the data is read from the struct by the publishing routine, formatted into a string and sent over to the ESP8266 chip via serial.

4.2.2 The CO_2 sensor driver

As mentioned before, the library for the CO_2 sensor was put together in part by porting over chunks of the open source ARM driver provided for the Hardwario module and in part by following the official manufacturer's documentation for the sensor. The sensor is connected on a serial line speaking the modbus protocol. Interfacing with it is fairly straightforward but its built-in filtering and continuous correction algorithms require state writebacks complicating the implementation somewhat. The sensor also requires a constant current source and voltage rail switching during its measurement cycle. These facilities are provided on the module. Since the module's intended use is with a smaller MCU with limited I/O, the relevant connections are routed to an I^2C I/O expander IC. Even though the ATmega MCU used in this device has plenty of I/O, implementing the functionality using the I^2C I/O expander chip proved a much neater solution. Also due to the limited I/O capacity of the intended microcontroller, an I^2C serial bridge is present on the module. The module's design, however, allows to break out the serial line to a pin header by moving a couple of links so this simple hardware modification was performed. It allowed the use of one of the hardware serials on the ATmega chip directly without the need to write routines for communicating with the serial bridge.

4.2.3 The ESP-01 module

The ESP-01 module provides the monitoring device with connectivity to the local network and all Internet related functions such as ntp time sync and MQTT publishing. Its firmware is built with IotWebConf library, which provides means to configure and store parameters such as wireless network credentials and MQTT server settings via a local http server [47]. This allows the device to be reconfigured without the need to re-flash the firmware. Upon powering the device up, a wireless access point is created. Devices that connect to this access point are then redirected to a captive portal providing the configuration options as depicted Figure 4.11. After setting up access to the local network and filling out MQTT server address and credentials, the monitoring device should be power cycled. The options are saved to non volatile memory and applied on every subsequent boot. Successful connection to a local network disables the access point. The configuration interface however remains accessible. Users can access it from any computer on the same network as the monitoring device by visiting the device's address from a web browser. When accessed this way, the configuration interface requires authentication with the password set up for the access point. The access point also reactivates for a period of 30 seconds after every reboot. Connecting to the AP halts the boot process so changes can be made to the configuration. The IotWebConf library also provides facilities for resetting the firmware in case of a forgotten password by means of boot time pin configuration but since the ESP-01 module used in this work does not have any pins usable for this purpose, the firmware must be reset by re-flashing. After connecting to the local wireless network and opening a connection to the MQTT server, the ESP begins its main loop. The loop reads the serial buffer, responds to http configuration server requests and periodically checks the network and MQTT server connections, attempting reconnection whenever necessary. The buffer reading routine is very simple: if the buffer contents are a request for time update, an ntp sync is performed and since no other serial communication is programmed, anything else



Thing name

AP password

WiFi SSID

WiFi password

MQTT settings

MQTT server

MQTT user

MQTT password

MQTT topic

MQTT server cert SHA-1 fingerprint

Leave blank for unsecured connection

NTP settings

NTP server address

e.g. europe.pool.ntp.org

Time zone (UTC offset)

UTC offset in seconds

Apply

Figure 4.11: The configuration interface as it would appear on a mobile device.

is published to the set up MQTT topic. The http server and network connection routines provided by the IotWebConf library are non-blocking so in case of an unavailable network or unreachable MQTT server, the rest of the functionality is not disrupted.

4.3 Server

A server for data gathering and visualization is also part of the solution. The server, same as the rest of the solution, is built on free and Open Source technologies. It is designed so that it can be hosted entirely locally within a private network or deployed to a VPS for example. The functional core is built with Python. The functionality implemented is as follows: An MQTT broker (Mosquitto) is responsible for routing topics and messages from connected device(s), a Python helper script using the Eclipse Paho MQTT library [27] listens to set topics and takes care of validating the incoming messages and writing them to an SQL database of choice. The reference implementation is using a simple SQLite database. Next, a Python Flask JSON API is available to query the database for its contents. This API is used mainly by the jQuery visualization web app but can be made available externally should the user desire to use the stored data for other purposes. More on these parts in later sections. This collection of programs can run behind any web server or reverse proxy on any platform as long as the appropriate MQTT and SQL libraries are available. The demo implementation is running on an Arch GNU/Linux based server behind an NGINX web server/reverse proxy. The server stack was also successfully tested on a Raspberry Pi 2 B single board computer running Arch GNU/Linux ARM. The NGINX web server also provides TLS with Let's Encrypt certificates for all running services.

4.3.1 Processing MQTT data

The MQTT broker of choice in the reference implementation is Mosquitto [26]. It was chosen because it is a well supported Open Source implementation of the full protocol and a multitude of resources is available for working with this broker. Since the protocol works with a client-broker model, capturing the incoming data requires a client listening to the appropriate topic. This is achieved by means of a python helper script. This script continuously listens on the set up topic and any incoming messages trigger a processing routine. The routine first reads the message payload, which contains the actual data in the form of a comma delimited string `"temperature,humidity,pressure,CO`$_2$`"`. The payload string is subsequently parsed and checked for out of range values. The processed values are stored internally in an array. Next, an SQL query is formulated with values from the payload array and the data is committed into the database. An example SQL formulation can be seen in Listing 4.3.

Listing 4.3: SQL command creation

```
sql = "INSERT INTO envdata0 (timestamp, temp, hum, pres, co2) VALUES (\"%s
    \", %s, %s, %s, %s)" % (datetime.datetime.now().strftime(f), dec_msg[0],
     dec_msg[1], dec_msq[2], dec_msg[3])
write_db(sql)
```

4.3.2 The database

The incoming data is stored in an SQL database. A library called SQLite is used to provide this functionality in the reference implementation. This library allows storing the data in a single database file without the need to run an SQL server. Thanks to the library's availability for many different languages and platforms, the server solution is able to be deployed on a variety of devices from traditional servers to ARM based single board computers. The solution is also able to use an existing SQL server with minimal modification as long as a Python library exists to interface with it.

4.3.3 The JSON API

Another core functionality of the server part is the JSON API. Its main purpose is to allow access to the collected data but it is also used internally for monitoring device state reporting. The API is written in Python using the Flask web service micro framework. The following is a brief description of the available methods, with a sample output presented in Listing 4.4.

The data retrieval method, by default available at `/api/getdata`, responds to both GET and POST requests. There is one optional argument `samples`, which allows the user to specify a number of samples to retrieve. Without this argument the sample count defaults to 120 data points (minutes). The samples are counted from the last available sample and timestamps with missing data points get a `NaN` value. The returned mime-type is obviously JSON and besides separate datasets for each temperature, humidity, pressure, and CO_2 concentration, an information about the state of the monitoring device is included. This can be used to identify whether the data is "fresh" i.e. no older than 120 seconds without having to parse any timestamps. An example output of a GET request `/api/getdata?samples=5` is shown in Listing 4.4 below.

Listing 4.4: An example output of the data fetching API method

```
{"status": "alive",
 "temp": {"2020-05-27 16:06:21": "23.2",
          "2020-05-27 16:07:21": "23.24",
          "2020-05-27 16:08:21": "23.24",
          "2020-05-27 16:09:21": "23.2",
          "2020-05-27 16:10:21": "23.22"},
 "hum": {"2020-05-27 16:06:21": "39.03",
         "2020-05-27 16:07:21": "38.43",
         "2020-05-27 16:08:21": "38.43",
         "2020-05-27 16:09:21": "38.47",
         "2020-05-27 16:10:21": "38.67"},
 "pres": {"2020-05-27 16:06:21": "1015",
          "2020-05-27 16:07:21": "1014",
          "2020-05-27 16:08:21": "1014",
          "2020-05-27 16:09:21": "1014",
          "2020-05-27 16:10:21": "1014"},
 "co2": {"2020-05-27 16:06:21": "537",
         "2020-05-27 16:07:21": "541",
         "2020-05-27 16:08:21": "539",
         "2020-05-27 16:09:21": "538",
         "2020-05-27 16:10:21": "537"}
}
```

The other method, available at `/api/heartbeat`, is mainly used internally for monitor state reporting. Since the device is unreachable from outside the local network it is attached to and therefore cannot respond to pings, a heartbeat is sent every time an MQTT message arrives from the device. This functionality is also covered by the helper script. After receiving a message, the script sends a POST request with the message time stamp. The POST request must contain a previously set up secret to protect and prevent erroneous data from being injected. The time stamp is then stored and whenever a device status is requested, a time delta between the time of request and the stored time stamp of the last message is calculated. The monitor is set up to send messages every minute, so if the delta is larger than 120 seconds, one can safely assume that the device is dead. A GET request to this method returns a JSON containing the device status and the last message time stamp in UNIX format as can be seen in Listing 4.5.

Listing 4.5: An example output of the heartbeat method

```
{"status": "alive",
 "last_msg": 1590656302.2858639}
```

4.3.4 The web application

To give the user an easy access to the gathered data, a web visualization application was created. This application is built with JavaScript and jQuery, the plots are drawn using the Chart.js library [24] and the main gauges with JustGage [61]. A screenshot of the application can be seen in Figure 4.12. A detail of the leftmost column is shown in Figure 4.13. This column contains gauges illustrating the latest values as well as a simple interface for changing the plot timebase. A detail of plots showing an overview of a full day can be seen in Figures 4.14 and 4.15. The application features a "responsive" design, which enables it to adapt to the browser viewport size to efficiently display its contents.
The application fetches the data by querying the aforementioned JSON API. This is done asynchronously using ajax. After the initial load, a timer refreshes the contents every 90 seconds by requesting the latest data point from the API. In case of a device outage the site stops refreshing its contents and instead polls the API for monitor status. Upon monitor state change from dead to alive, the whole dataset is reloaded and the visualization continues operating as usual. Any missing data points are then drawn in the plot as an empty space. At the bottom of the web page is a status bar of sorts indicating the current mode of operation (live data or latest values presented statically).

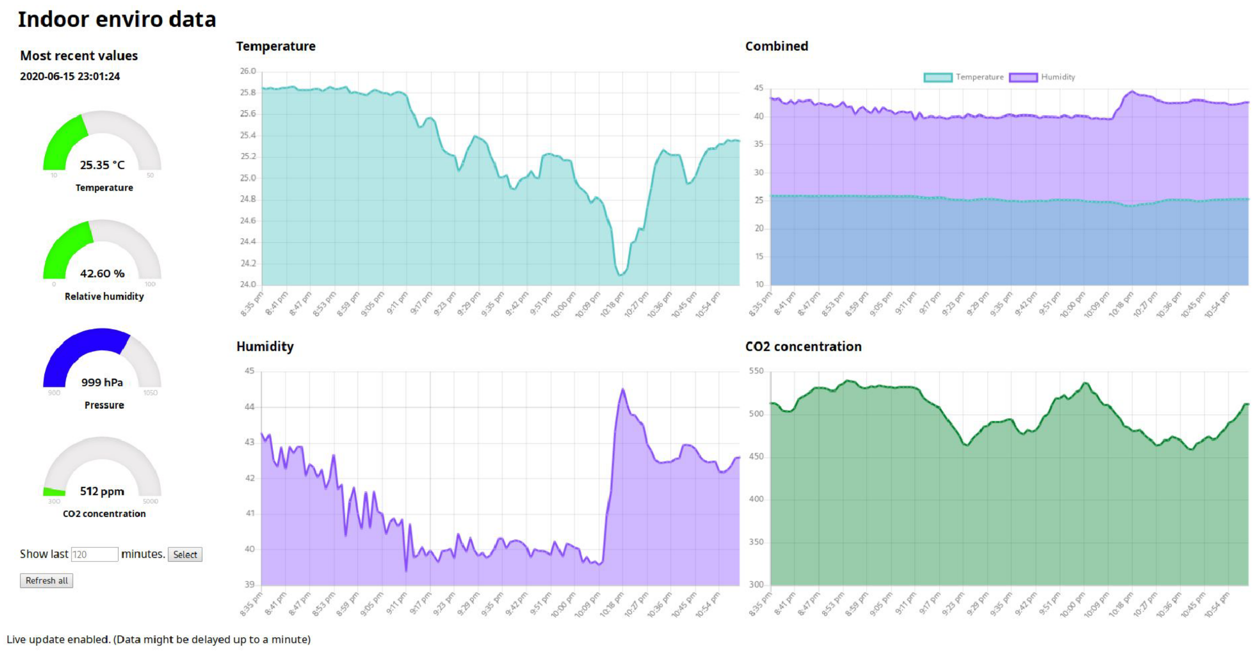
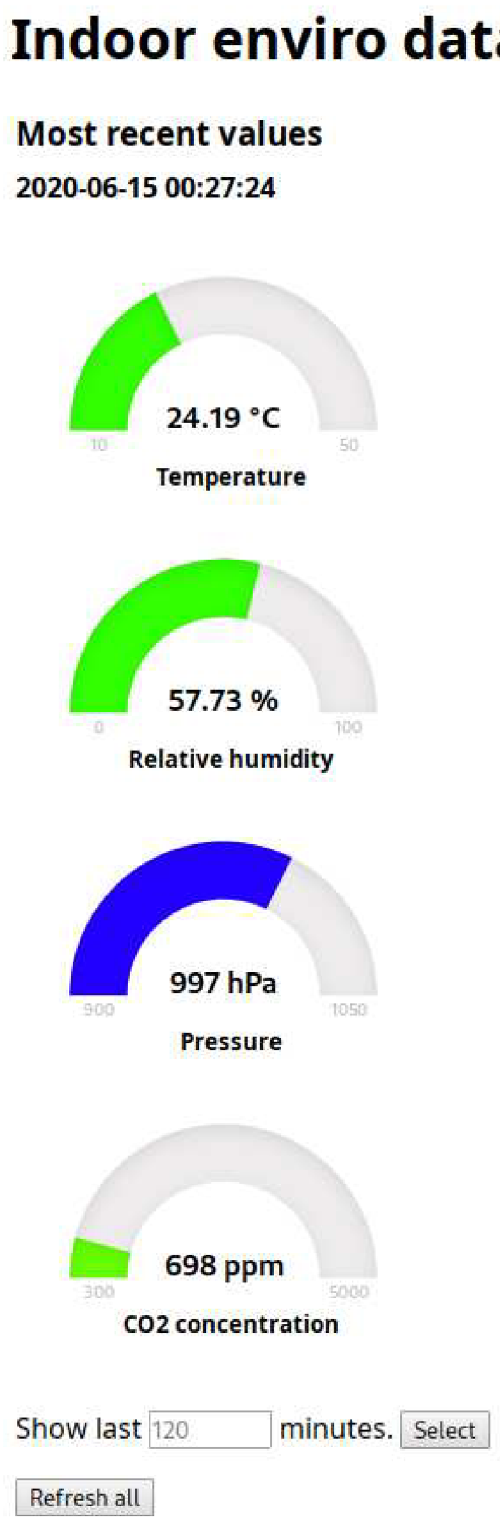


Figure 4.12: The overview web application.


Indoor enviro data

Most recent values

2020-06-15 00:27:24

Figure 4.13: Detail of the web application interface.

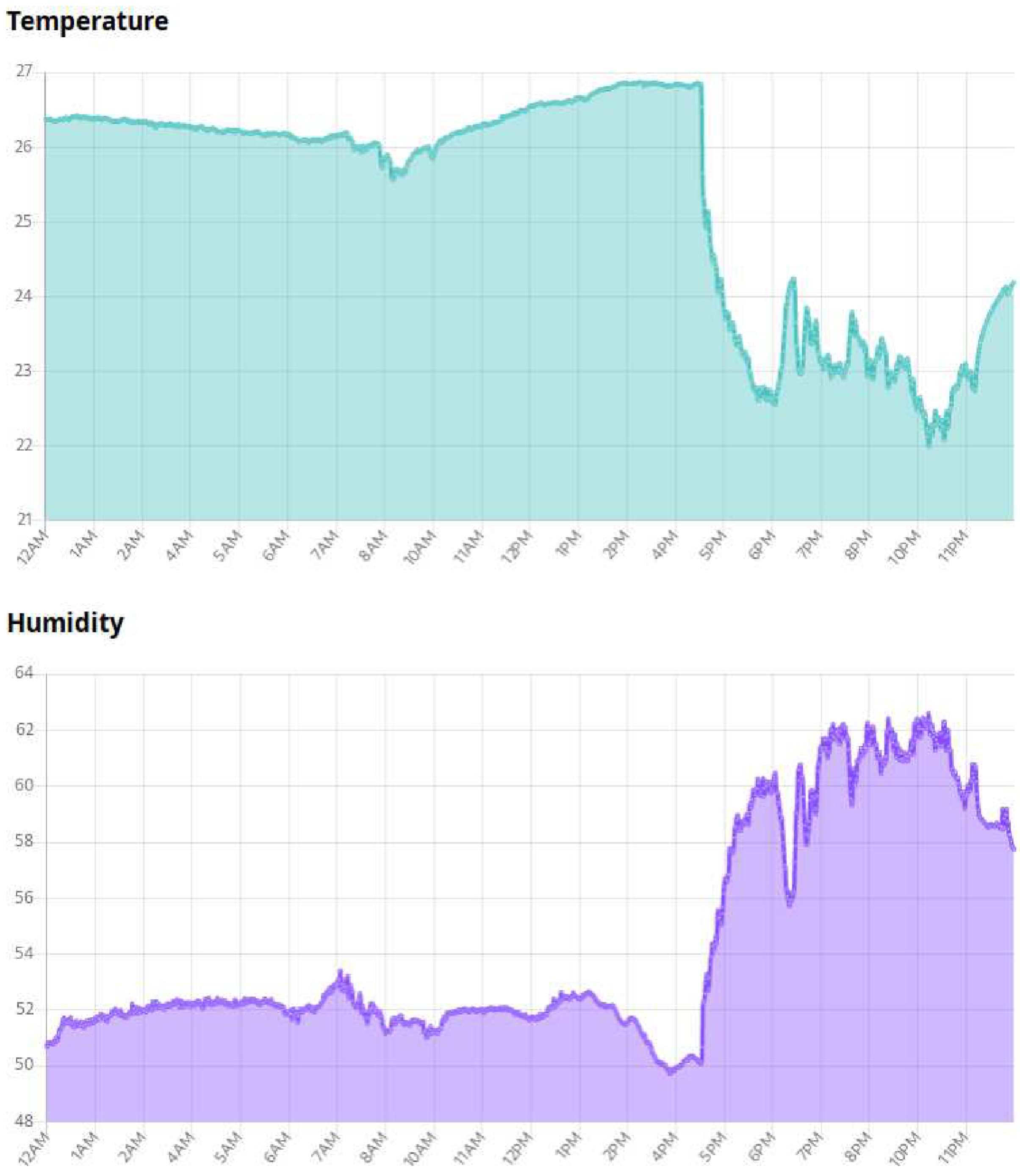
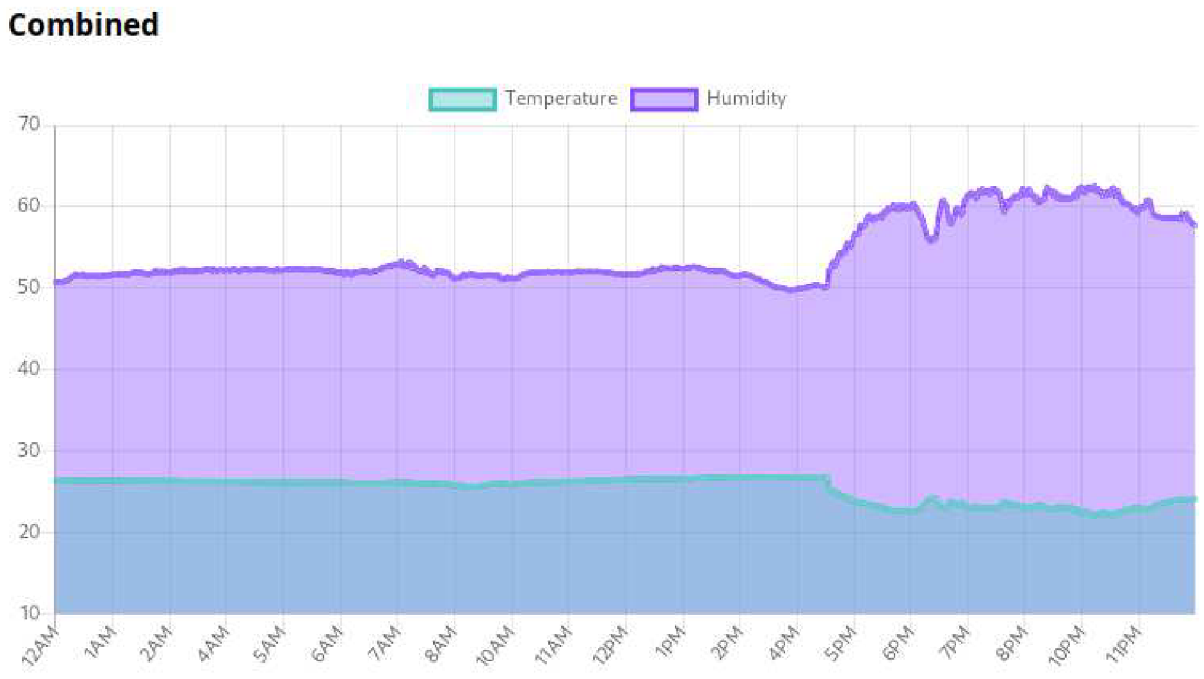


Figure 4.14: A full day overview (center column).



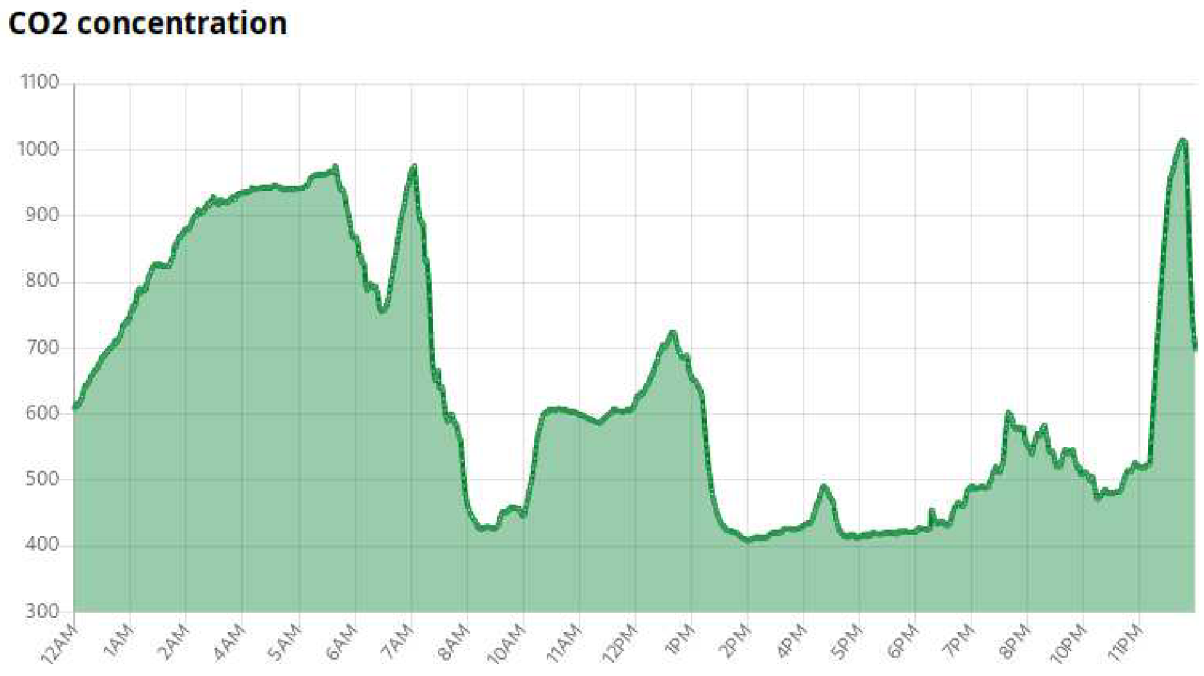


Figure 4.15: A full day overview (right column).

4.4 The completed device

Construction of the device was developed alongside the firmware and also progressively evolved. One difference from the development board is the absence of a dedicated 3.3 V LDO on the finished device. The alternative version of the MCU board used for the final product contains an appropriate 3.3 V regulator so the need for an external one was obsoleted. While the device was intended to remain modular from the start, another major change is the main board. A custom double sided breakout PCB for the modules was laid out but due to circumstances outside of author's control, custom PCB manufacturing was unavailable at the time so the board is instead hand-wired on a single-sided prototyping board. This posed somewhat of a challenge since the board requires pin headers broken out on both sides. The finished product is functional as intended albeit slightly larger and not very aesthetically pleasing. Figures 4.16 and 4.17 provide a look at the populated board. Immediately apparent is the modification performed to the CO_2 module. The RTC module and the BME sensor are attached to the otherwise unused connections on the module. This solution saves a significant amount of space and simplifies the breakout board further.

Figures 4.18 and 4.19 show renders of the basic PCB for comparison. An alternative PCB design also provides footprints for two 2 mm pin pitch JST connectors for other power sources. A render of this slightly taller alternative design is presented in Figure 4.20. One connector is for powering the device with a 7-12 V DC supply and one with 5 V DC directly either from a lithium battery with a step-up converter or other 5V source. Only one connector should be used at a time. Powering the board through these connectors will result in a neater appearance since the cable can be routed through the back panel, but the serial output will be unavailable.

The device's housing is 3D printed out of PLA plastic. The enclosure measures 65 mm wide, 75 mm deep and 110 mm tall for the PCB version and 67 mm wide, 75 mm deep and 115 mm tall for the prototyping board version. The model was created using an OpenSCAD script for generating custom enclosures, written by the user Heartman on Thingiverse [32] and modified by the author for use with the device. An OpenSCAD render of the enclosure can be seen in Figure 4.21. The front panel contains a cutout for the e-ink display and the back panel has some additional ventilation holes in the vicinity of the CO_2 sensor. The back panel also includes a cutout for a power cable in case the user elects to power the board through the optional DC input. The bottom part of the enclosure has enough space to house a battery and related circuitry. The board is placed vertically into rails. The mounted board can be seen in Figure 4.22. A cutout on the side of the enclosure allows connection to the MCU board's micro USB port. The reset button is also accessible just below the connector but power cycling is the preferred method of resetting the device and should be used instead. The assembled device is shown in Figure 4.23.

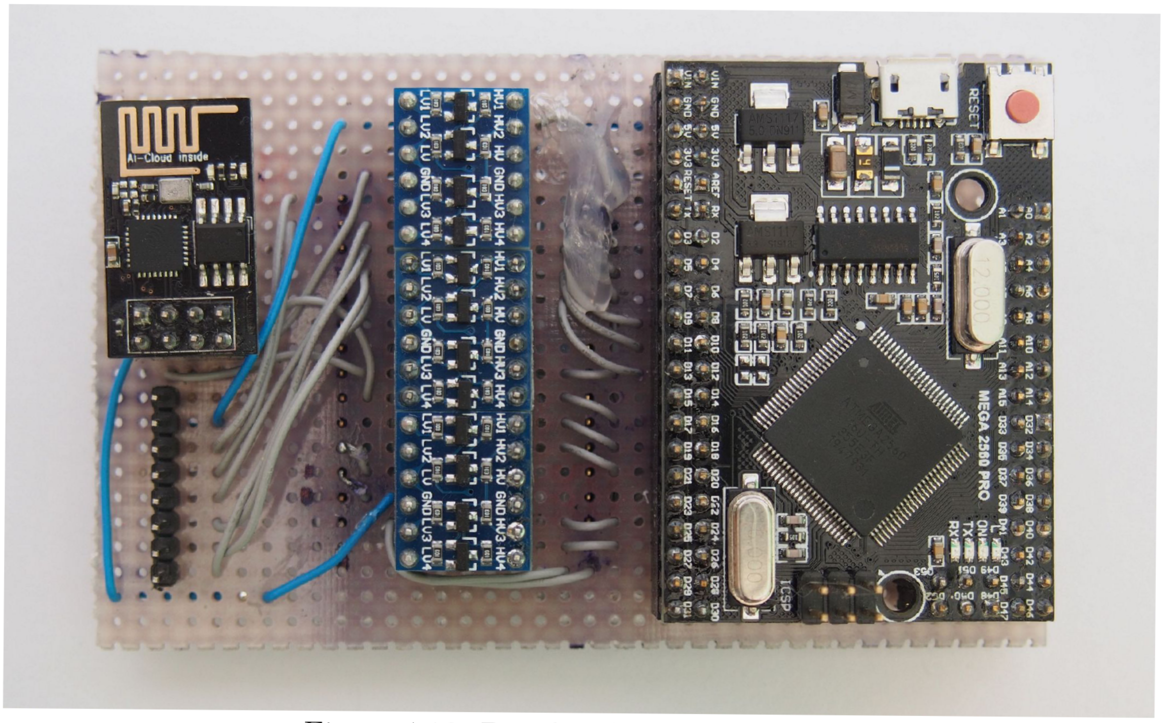


Figure 4.16: Populated board, front side.

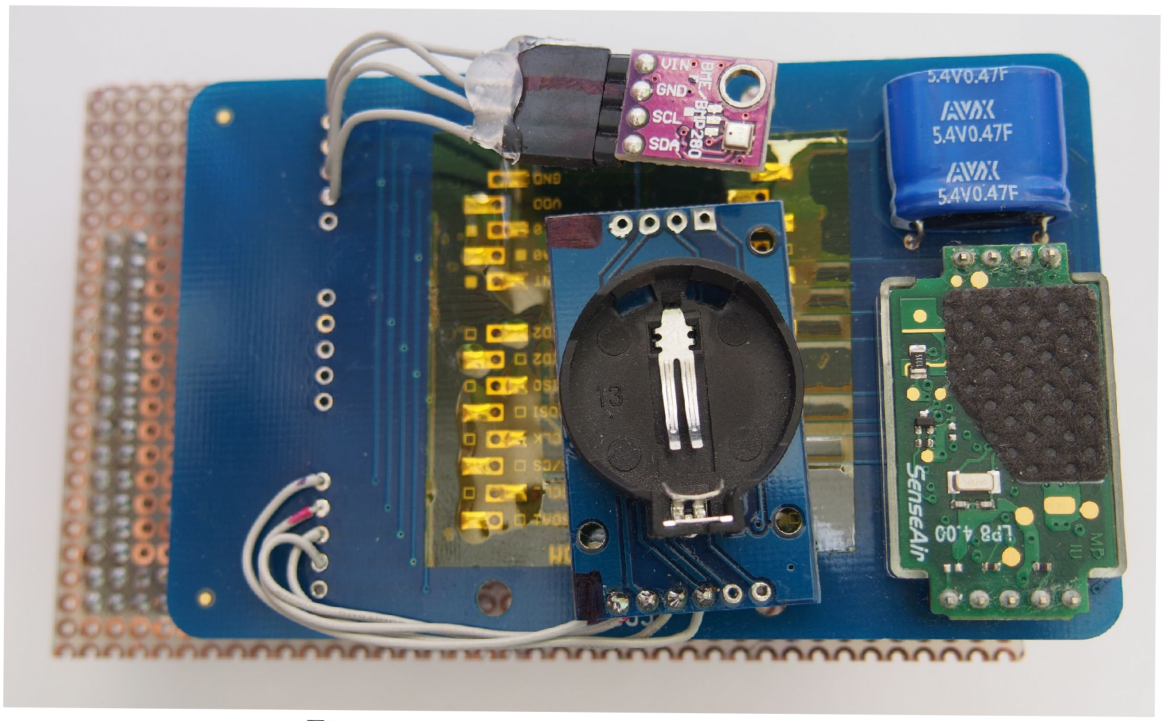


Figure 4.17: Populated board, back side.

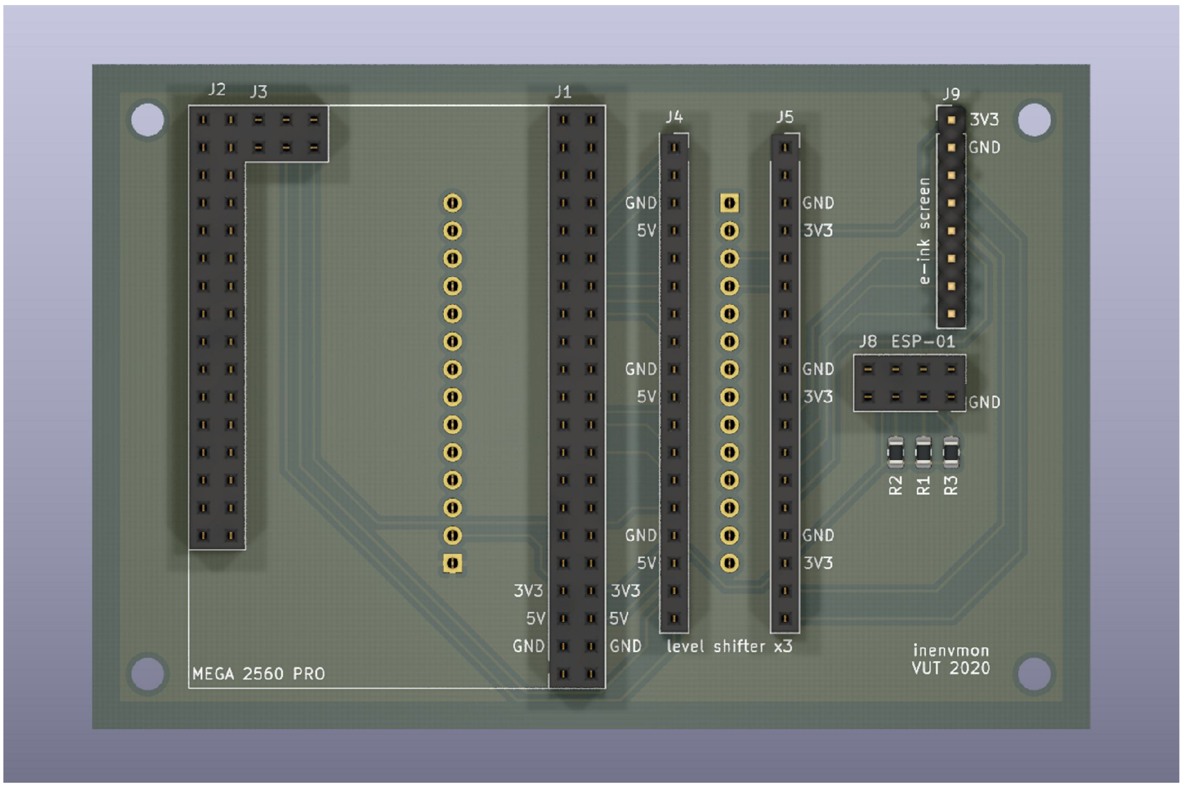


Figure 4.18: PCB render, front side.

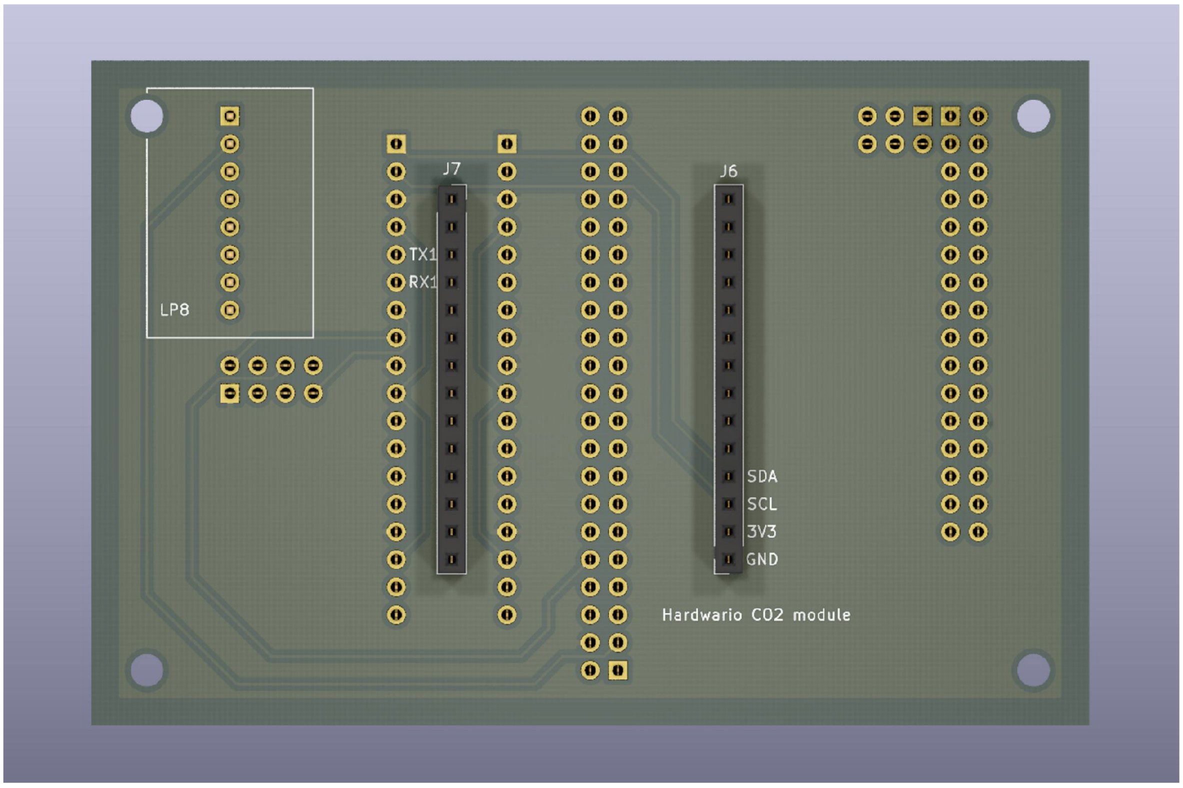


Figure 4.19: PCB render, back side.

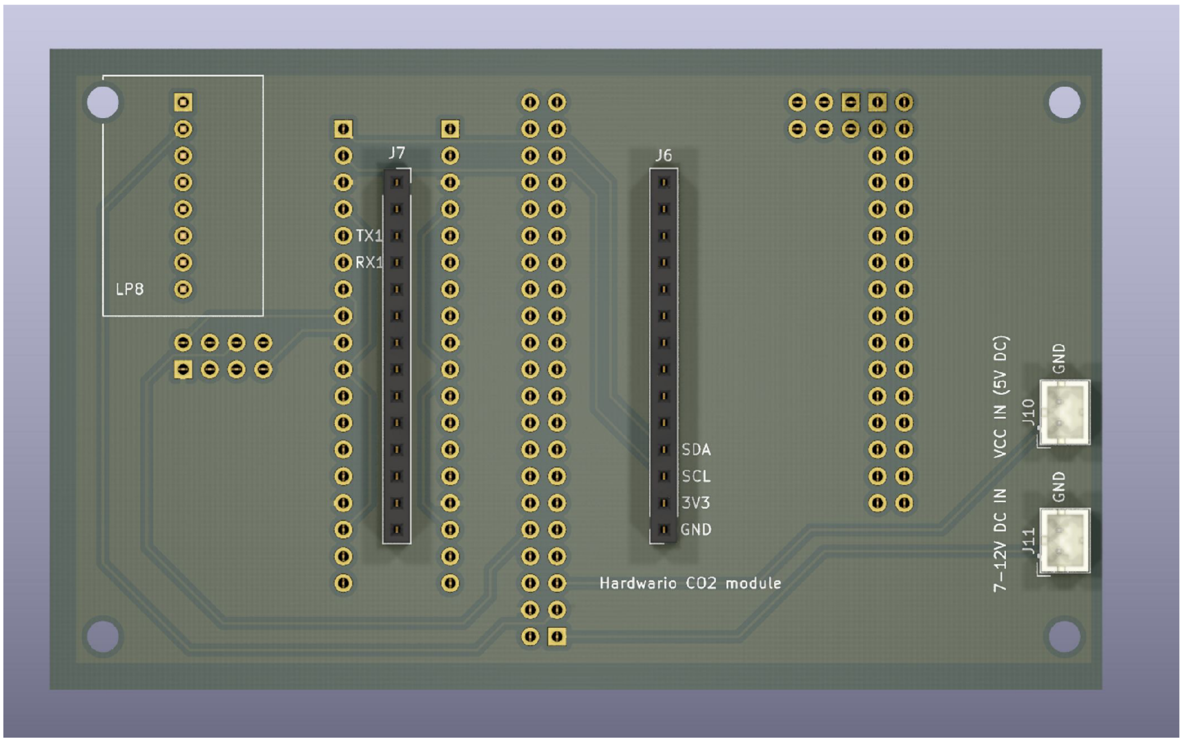
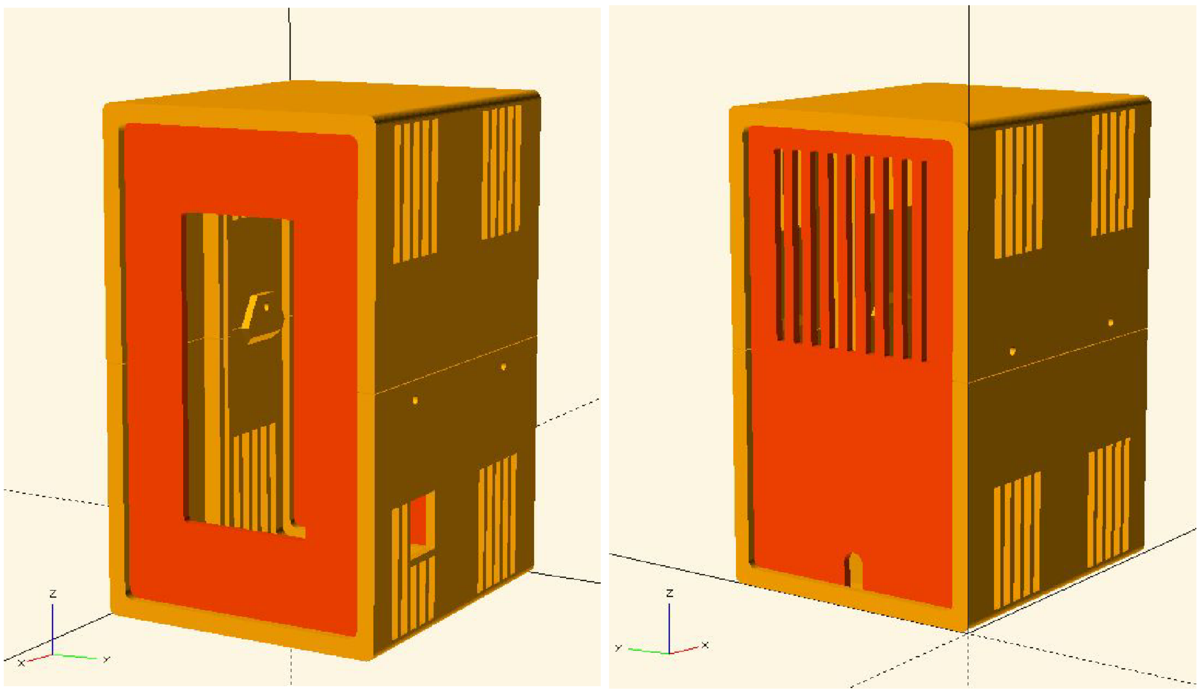


Figure 4.20: Render of the alternative PCB with JST connectors.



(a) View of the front

(b) View of the back

Figure 4.21: OpenSCAD renders of the enclosure.

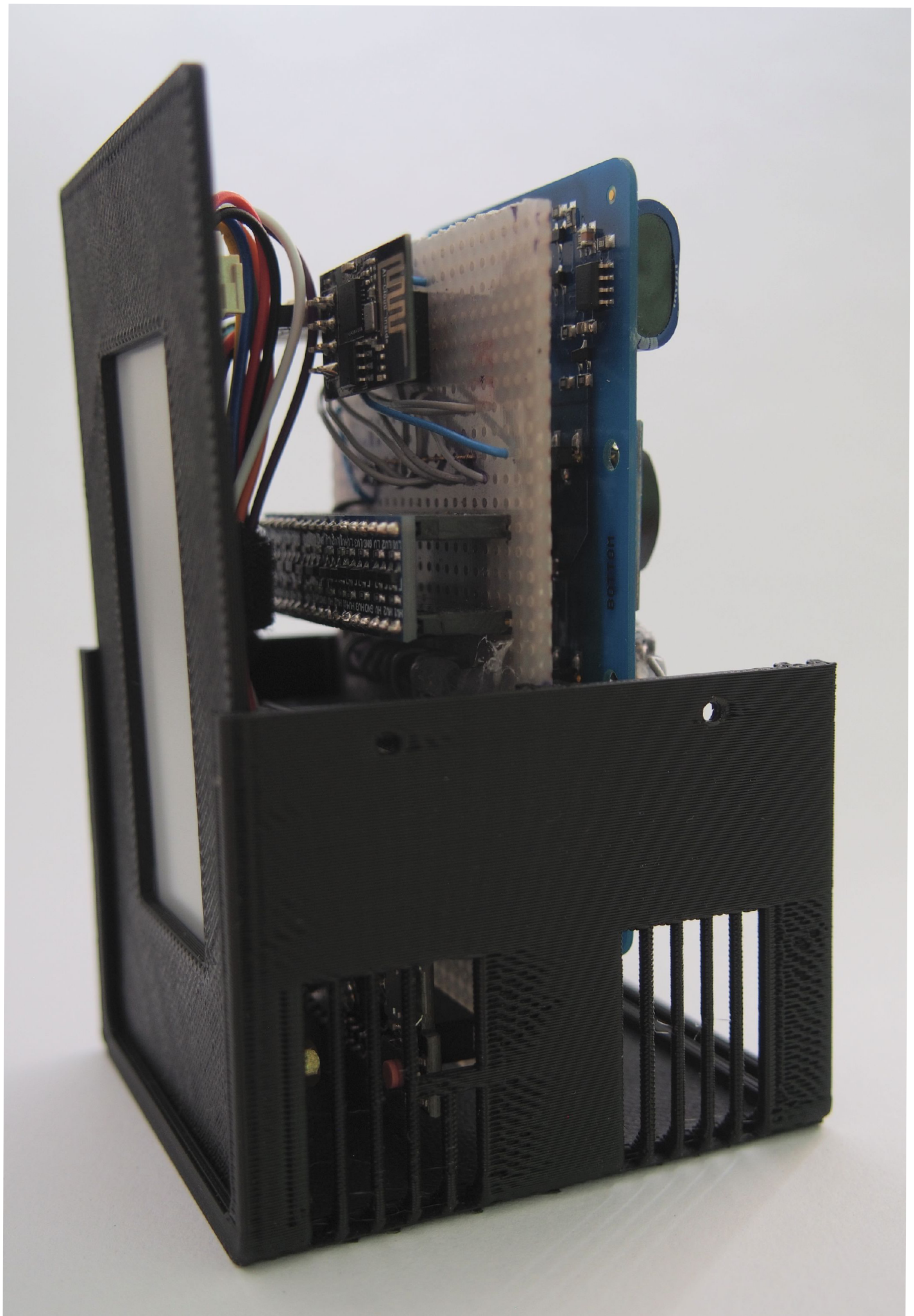


Figure 4.22: Detail of the board mounting.

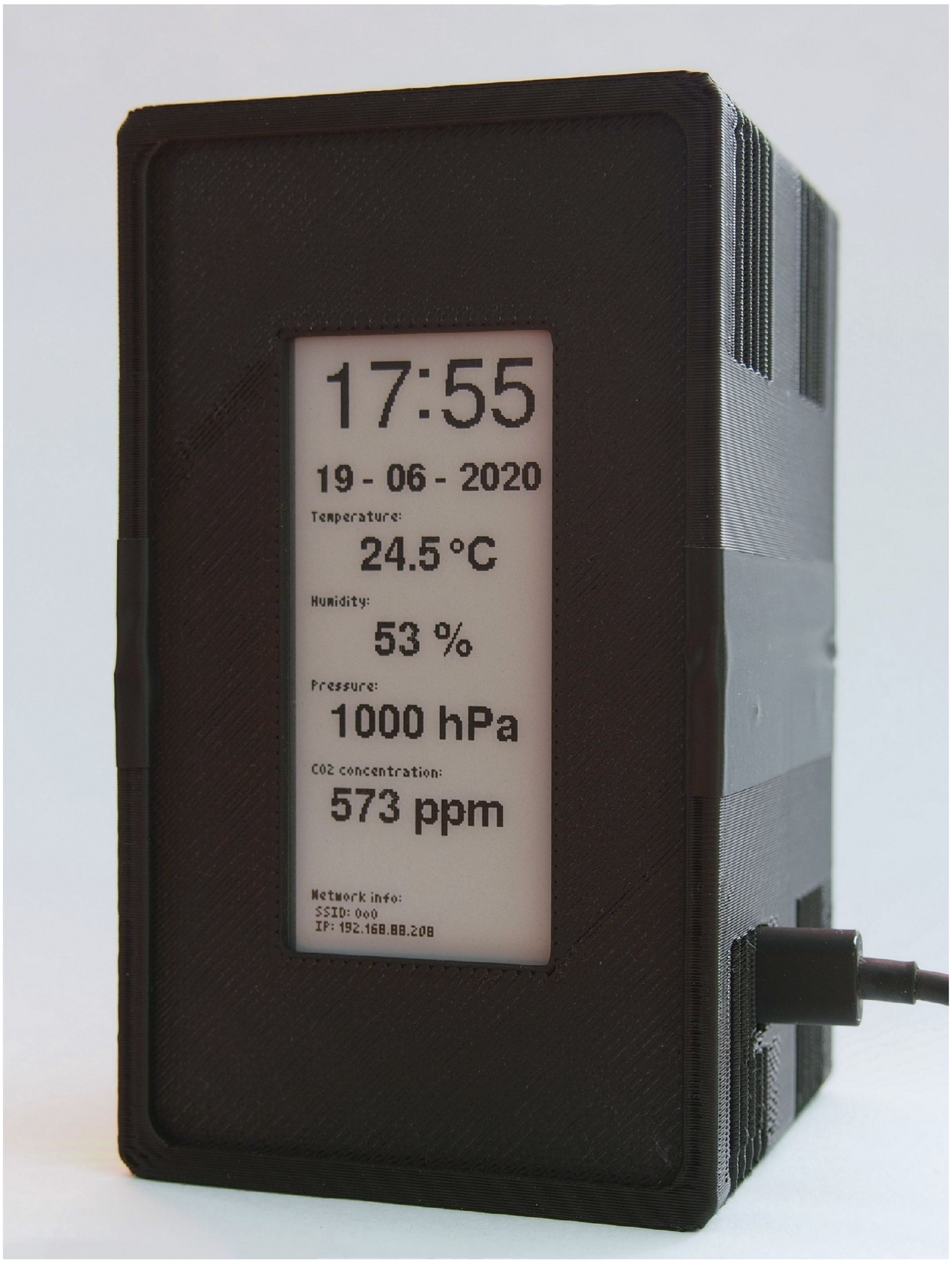


Figure 4.23: The finished device.

5 Conclusion

The goal of this bachelor's thesis was to implement an indoor climate monitoring device based on the Arduino platform. This was achieved by using an 8-bit ATmega2560 MCU and a WiFi capable ESP8266 module. The monitoring device was extensively tested over a period of four months and proven reliable. The remote data gathering is facilitated by the MQTT protocol. To supplement the device's functionality, a web server comprising of a JSON API and a visualization frontend for easy data access was implemented. The API provides a method for retrieving data points using both GET and POST requests. A JavaScript visualization application uses this API to display plots of the measured data. The entire solution is built with freedom and extensibility in mind. Free Open Source technologies and hardware were leveraged wherever possible. Another result of this work is a port of a Hardwario CO_2 sensor module driver for use with the Arduino platform.

The monitoring device and the auxiliary server are tested and fully functional. The device is further extensible by means of additional sensors. The entire service is also extensible to multiple monitoring devices. There is however still space for improvement mainly in the area of user experience and the power efficiency of the monitoring device which were outside the original scope of this work.

The entirety of this project is available under an Open Source license in a git repository git.dotya.ml/thesis_project. A demonstration of the web server and API is available at env.astora.gq.

Bibliography

[1] Aliexpress.com. *MH Z19B Infrared CO2 Sensor for CO2 Monitor MH-Z19B 5000PPM MH-Z19B NDIR Gas Sensor CO2 gas sensor MH-Z19*. [online]. [cit. 2020-02-11]. Available from: https://www.aliexpress.com/item/32672336586.html.

[2] American Meteorological Society. *Aneroid barometer*. Meteorology Glossary. [online]. Apr. 2012, ©2020 [cit. 2020-04-20]. Available from: http://glossary.ametsoc.org/wiki/Aneroid_barometer.

[3] American Meteorological Society. *Hair hygrometer*. Meteorology Glossary. [online]. Feb. 2012, ©2020 [cit. 2020-04-20]. Available from: http://glossary.ametsoc.org/wiki/Hair_hygrometer.

[4] American Meteorological Society. *Mercury barometer*. Meteorology Glossary. [online]. Apr. 2012, ©2020 [cit. 2020-04-20]. Available from: http://glossary.ametsoc.org/wiki/Mercury_barometer.

[5] American Meteorological Society. *Spectral hygrometer*. Meteorology Glossary. [online]. Apr. 2012, ©2020 [cit. 2020-04-20]. Available from: http://glossary.ametsoc.org/wiki/Spectral_hygrometer.

[6] AOSONG. *Digital Temperature and Humidity Sensor: AM2320 product manual*. [online]. [cit. 2020-02-11]. Available from: https://cdn-shop.adafruit.com/product-files/3721/AM2320.pdf.

[7] Arduino-shop.cz. *Arduino DHT22 teploměr a vlhkoměr digitální*. [online]. [cit. 2020-02-11]. Available from: https://arduino-shop.cz/arduino/1188-arduino-dht22-teplomer-a-vlhkomer-digitalni.html.

[8] Arduino-shop.cz. *BME280 Modul Měření Teploty Vlhkosti a Barometrického Tlaku Precizní*. [online]. [cit. 2020-02-11]. Available from: https://arduino-shop.cz/arduino/1361-bme280-modul-mereni-teploty-vlhkosti-a-barometrickeho-tlaku-precizni.html.

[9] Arduino-shop.cz. *IIC I2C Senzor Tlaku a Teploty BMP280 3,3V*. [online]. [cit. 2020-02-11]. Available from: https://arduino-shop.cz/arduino/1488-iic-i2c-senzor-tlaku-a-teploty-bmp280-3-3v.html.

[10] Arduino-shop.cz. *Internet věcí je tady! TCP/IP WIFI ESP8266 ESP-01*. [online]. [cit. 2020-05-19]. Available from: https://arduino-shop.cz/arduino/911-internet-veci-je-tady-tcp-ip-wifi-esp8266-esp-01.html.

[11] Arduino-shop.cz. *RTC Hodiny reálného času DS3231*. [online]. [cit. 2020-05-19]. Available from: https://arduino-shop.cz/arduino/1261-rtc-hodiny-realneho-casu-ds3231-at24c32-iic-pametovy-modul-pro-arduino.html.

[12] Arduino-shop.cz. *Teploměr a vlhkoměr AM2320 digitální*. [online]. [cit. 2020-02-11]. Available from: https://arduino-shop.cz/arduino/2023-teplomer-a-vlhkomer-am2320-digitalni.html.

[13] Arduino Store. *Arduino Mega 2560 rev.3*. [online]. [cit. 2020-05-19]. Available from: https://store.arduino.cc/arduino-mega-2560-rev3.

[14] Banks, A., Briggs, E., Borgendale, K., and Gupta, R. *MQTT Version 5.0*. OASIS Standard. [online]. Mar. 2019 [cit. 2020-05-19]. Available from: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html.

[15] Banks, A., Briggs, E., Borgendale, K., and Gupta, R. *MQTT Version 5.0*. p. 20. OASIS Standard. [online]. Mar. 2019 [cit. 2020-05-19]. Available from: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html.

[16] Banks, A., Briggs, E., Borgendale, K., and Gupta, R. *MQTT Version 5.0*. p. 36. OASIS Standard. [online]. Mar. 2019 [cit. 2020-05-19]. Available from: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html.

[17] Banks, A., Briggs, E., Borgendale, K., and Gupta, R. *MQTT Version 5.0*. pp. 21-22. OASIS Standard. [online]. Mar. 2019 [cit. 2020-05-19]. Available from: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html.

[18] bigclownlabs, HARDWARIO Hardware (Schematic and Assembly Drawings). *bc-module-co2-rev-1-4-sch.pdf*. GitHub. [online]. [cit. 2020-05-29]. Available from: https://github.com/bigclownlabs/bc-hardware/blob/master/out/bc-module-co2/bc-module-co2-rev-1-4-sch.pdf.

[19] Bosch-sensortec. *BME280 Combined humidity and pressure sensor*. [online]. Rev. 1.6, ©Sep. 2018 [cit. 2020-02-11]. Available from: https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bme280-ds002.pdf.

[20] Bosch-sensortec. *BMP280 Digital Pressure Sensor*. [online]. Rev. 1.19, ©2018 [cit. 2020-02-11]. Available from: https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bmp280-ds001.pdf.

[21] Burroughs, H. and Hansen, S. J. *Managing Indoor Air Quality*. pp. 149 – 151. Fairmont Press, Lilburn (Georgia). fifth edn.. 2011. ISBN 978-0-88173-661-8.

[22] Burroughs, H. and Hansen, S. J. *Managing Indoor Air Quality*. pp. 151 – 155. Fairmont Press, Lilburn (Georgia). fifth edn.. 2011. ISBN 978-0-88173-661-8.

[23] Burroughs, H. and Hansen, S. J. *Managing Indoor Air Quality*. pp. 92 – 96. Fairmont Press, Lilburn (Georgia). fifth edn.. 2011. ISBN 978-0-88173-661-8.

[24] Chartjs. *Chart.js, Simple HTML5 Charts using the <canvas> tag*. GitHub. [online]. [cit. 2020-06-07]. Available from: https://github.com/chartjs/Chart.js.

[25] *Digital relative humidity & temperature sensor AM2302/DHT22*. [online]. [cit. 2020-02-11]. Available from: https://cdn-shop.adafruit.com/datasheets/Digital+humidity+and+temperature+sensor+AM2302.pdf.

[26] Eclipse Foundation. *Eclipse Mosquitto MQTT broker*. [online]. Eclipse Foundation. [cit. 2020-04-20]. Available from: http://mosquitto.org/.

[27] Eclipse Foundation. *Eclipse Paho MQTT library.* [online]. [cit. 2020-06-07]. Available from: https://www.eclipse.org/paho/.

[28] Esp8266 community. *ESP8266 core for Arduino.* GitHub. [online]. [cit. 2020-05-19]. Available from: https://github.com/esp8266/Arduino.

[29] Espressif. *ESP8266EX: datasheet.* [online]. ©2020 [cit. 2020-05-19]. Available from: https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf.

[30] Finitespace. *BME280, Arduino library for reading and interpreting Bosch BME280 data over I2C, SPI or Sw SPI.* GitHub. [online]. [cit. 2020-04-20]. Available from: https://github.com/finitespace/BME280.

[31] Hardwario. *Hardwario shop: CO_2 module.* [online]. [cit. 2020-05-19]. Available from: https://shop.hardwario.com/co2-module/.

[32] Heartman. *The Ultimate box maker.* Thingiverse. [online]. Heartman, 2016 [cit. 2020-06-07]. Available from: https://www.thingiverse.com/thing:1264391.

[33] International Standards Organization. *ISO/IEC 20922:2016 Message Queuing Telemetry Transport (MQTT) v3.1.1.* [online]. Jun. 2016 [cit. 2020-04-20]. Available from: https://www.iso.org/standard/69466.html.

[34] Let's Encrypt – A nonprofit Certificate Authority. *Frequently Asked Questions.* [online]. [cit. 2020-04-20]. Available from: https://letsencrypt.org/docs/faq/.

[35] MajicDesigns. *MD_DS3231 RTC library.* GitHub. [online]. [cit. 2020-04-20]. Available from: https://github.com/MajicDesigns/MD_DS3231.

[36] Maxim Integrated Products. *DS3231 Extremely Accurate I2C-Integrated RTC/TCXO/Crystal.* [online]. ©2015 [cit. 2020-04-20]. Available from: https://datasheets.maximintegrated.com/en/ds/DS3231.pdf.

[37] Meyer, C. W. *The second-generation NIST standard hygrometer.* *Metrologia.* 47(3), pp. 192 – 207. 2010. doi:10.1088/0026-1394/47/3/010.

[38] Mosquitto. *MQTT man page.* [online]. [cit. 2020-04-20]. Available from: http://mosquitto.org/man/mqtt-7.html.

[39] MQTT. *Frequently Asked Questions.* In: mqtt.org. [online]. [cit. 2020-04-20]. Available from: https://mqtt.org/faq.

[40] OPC Foundation. *OPC Foundation technologies: OPC Classic.* [online]. ©2020 [cit. 2020-06-10]. Available from: https://opcfoundation.org/about/opc-technologies/opc-classic/.

[41] OPC Foundation. *OPC Foundation technologies: OPC UA.* [online]. ©2020 [cit. 2020-06-10]. Available from: https://opcfoundation.org/about/opc-technologies/opc-ua/.

[42] OPC Foundation. *OPC UA Online Reference, Part 1, Chapter 2 Reference documents.* [online]. Ver. 1.04, Nov. 2017, ©2020 [cit. 2020-06-10]. Available from: https://reference.opcfoundation.org/v104/Core/docs/Part1/2/.

[43] OPC Foundation. *OPC UA Online Reference, Part 1, Chapter 5.3 Design goals.* [online]. Ver. 1.04, Nov. 2017, ©2020 [cit. 2020-06-10]. Available from: https://reference.opcfoundation.org/v104/Core/docs/Part1/5.3/.

[44] OPC Foundation. *OPC UA Online Reference, Part 14 PubSub.* [online]. Ver. 1.04, Feb. 2018, ©2020 [cit. 2020-06-10]. Available from: https://reference.opcfoundation.org/v104/Core/docs/Part14/.

[45] OPC Foundation. *OPC UA Online Reference, Part 2, Chapter 4.2.1 Security objectives.* [online]. Ver. 1.04, Aug. 2018, ©2020 [cit. 2020-06-10]. Available from: https://reference.opcfoundation.org/v104/Core/docs/Part2/4.2.1/.

[46] Portland State Aerospace Society. *A Quick Derivation relating altitude to air pressure.* [online]. Ver. 1.03, Dec. 2004. ©2004 [cit. 2020-04-20]. Available from: http://archive.psas.pdx.edu/RocketScience/PressureAltitude_Derived.pdf.

[47] Prampec. *IotWebConf, ESP8266/ESP32 non-blocking WiFi/AP web configuration Arduino library.* GitHub. [online]. [cit. 2020-06-07]. Available from: https://github.com/prampec/IotWebConf.

[48] RobotDyn. *RobotDyn store: Mega 2560 PRO (Embed).* [online]. [cit. 2020-05-19]. Available from: https://robotdyn.com/mega-2560-pro-embed-ch340g-atmega2560-16au.html.

[49] Roveti, D. K. *Choosing a humidity sensor: A review of three technologies.* In: fierceelectronics.com [online]. 2001-07-01 [cit. 2020-04-20]. Available from: https://www.fierceelectronics.com/components/choosing-a-humidity-sensor-a-review-three-technologies.

[50] Senseair. *Product Specification: Senseair ®LP8.* [online]. Rev. 7, ©2019 [cit. 2020-02-11]. Available from: https://rmtplusstoragesenseair.blob.core.windows.net/docs/Dev/publicerat/PSP1334.pdf.

[51] Senseair. *Senseair LP8 CO_2 Sensor specification and integration guideline.* [online]. [cit. 2020-02-11]. Available from: https://rmtplusstoragesenseair.blob.core.windows.net/docs/Dev/publicerat/TDE2712.pdf.

[52] Senseair. *Why NDIR?* [online]. [cit. 2020-02-11]. Available from: https://senseair.com/knowledge/sensor-technology/technology/why-ndir/.

[53] Sensirion. *Datasheet Sensirion SCD30 Sensor Module.* [online]. Rev 0.94, ©Jun. 2019 [cit. 2020-02-11]. Available from: https://eu.mouser.com/pdfDocs/Sensirion_CO2_Sensors_SCD30_Datasheet1.pdf.

[54] Sparkfun.com. *CO_2 Humidity and Temperature Sensor - SCD30*. [online]. [cit. 2020-05-29]. Available from: https://www.sparkfun.com/products/15112.

[55] STANFORD-CLARK, A. AND TRUONG, H. L. *MQTT For Sensor Networks (MQTT-SN)*. [online]. ©2013. Version 1.2, Nov. 2013 [cit. 2020-05-19]. Available from: https://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf.

[56] The HiveMQ team. *Client, Broker / Server and Connection Establishment - MQTT Essentials: Part 3*. In: hivemq.com [online]. 2019-07-17 [cit. 2020-04-20]. Available from: https://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment/.

[57] The HiveMQ team. *Foundational Changes in the MQTT 5 Protocol - MQTT 5 Essentials Part 2*. In: hivemq.com [online]. 2018-01-08 [cit. 2020-04-20]. Available from: https://www.hivemq.com/blog/mqtt5-essentials-part2-foundational-changes-in-the-protocol/.

[58] The HiveMQ team. *Getting started with MQTT*. In: hivemq.com [online]. 2019-07-04 [cit. 2020-04-20]. Available from: https://www.hivemq.com/blog/how-to-get-started-with-mqtt/.

[59] The HiveMQ team. *MQTT Publish, Subscribe & Unsubscribe - MQTT Essentials: Part 4*. In: hivemq.com [online]. 2015-02-02 [cit. 2020-04-20]. Available from: https://www.hivemq.com/blog/mqtt-essentials-part-4-mqtt-publish-subscribe-unsubscribe/.

[60] The HiveMQ team. *Retained messages - MQTT Essentials: Part 8*. In: hivemq.com [online]. 2015-03-02 [cit. 2020-04-20]. Available from: https://www.hivemq.com/blog/mqtt-essentials-part-8-retained-messages/.

[61] Toorshia. *JustGage, a handy JavaScript plugin for generating dashboard gauges. Based on Raphaël library for vector drawing*. GitHub. [online]. [cit. 2020-06-07]. Available from: https://github.com/toorshia/justgage.

[62] ZHENGZHOU WINSEN ELECTRONICS TECHNOLOGY CO.,LTD. *Intelligent Infrared CO2 Module (Model: MH-Z19)*. [online]. Rev. 1.0, Mar. 2015 [cit. 2020-02-11]. Available from: https://www.winsen-sensor.com/d/files/PDF/Infrared%20Gas%20Sensor/NDIR%20CO2%20SENSOR/MH-Z19%20CO2%20Ver1.0.pdf.

[63] ZinggJM. *GxEPD2, Arduino Display Library for SPI E-Paper Displays*. GitHub. [online]. [cit. 2020-04-20]. Available from: https://github.com/ZinggJM/GxEPD2.

List of Abbreviations

°C	degrees Celsius
µA	microampere
A	Ampere
AP	Access Point
API	Application Programming Interface
CO_2	Carbon dioxide
DC	Direct Current
DIL	Dual In-Line
EEPROM	Electrically Erasable Programmable Read Only Memory
FPU	Floating Point Unit
GPIO	General Purpose Input/Output
http	Hypertext Transfer Protocol
I/O	Input Output
I^2C	Inter-Integrated-Circuit bus
IDE	Integrated Development Environment
IoT	Internet of Things
IP	Internet Protocol
JSON	JavaScript Object Notation
LDO	Low dropout voltage regulator
LED	Light Emitting Diode
mA	milliampere
MCU	Microcontroller
MEMS	Micro Electro-Mechanical Systems
MQTT	MQ Telemetry Transport
N/A	Not available
NDIR	Non-dispersive Infra Red
ntp	Nertwork Time Protocol

PCB	Printed Circuit Board
PLA	Polylactic acid
ppm	parts per million
QoS	Quality of Service
QSPI	Quad Serial Peripheral Interface
RH	Relative Humidity
RISC	Reduced Instruction Set Computer
RTC	Real-time clock
SPI	Serial Peripheral Interface
SQL	Structured Query Language
SSID	Service Set Identifier
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UI	User interface
USB	Universal Serial Bus
V	Volts
VOC	Volatile Organic Compound
VPS	Virtual Private Server

List of Figures

List of Tables

APPENDIX A

Folder structure of the attached archive. The archive contains authored software and its dependencies, schematics and 3D models as well as a copy of this text.

```
b_thesis_pillar_attachment.zip
  Firmware
    fw_mega
      fw_mega.ino
      FreeSans22pt7b.h
      FreeSansBold11pt7b.h
    fw_esp8266
      fw_esp8266.ino
    Libraries
      bcl_co2_arduino
        bc_co2_module_arduino.cpp
        bc_co2_module_arduino.h
  Hardware
    Models
      panel_back.stl
      panel_front.stl
      shell_bottom.stl
      shell_top.stl
    Schematics
      inenvmon_breakout.pdf
      inenvmon_breakout.kicad_pcb
      inenvmon_breakout_power_conns.kicad_pcb
  Server
    inenvmon_web
      static
        js
          Chart.min.js
          jquery.min.js
          justgage.min.js
          main.js
          moment.min.js
          raphael.min.js
        styles
          Chart.min.css
          style.css
      templates
        index.html
      inenvmon_web.py
    inenvmon_collector.py
    inenvmon_data.db
  b_thesis_pillar.pdf
```