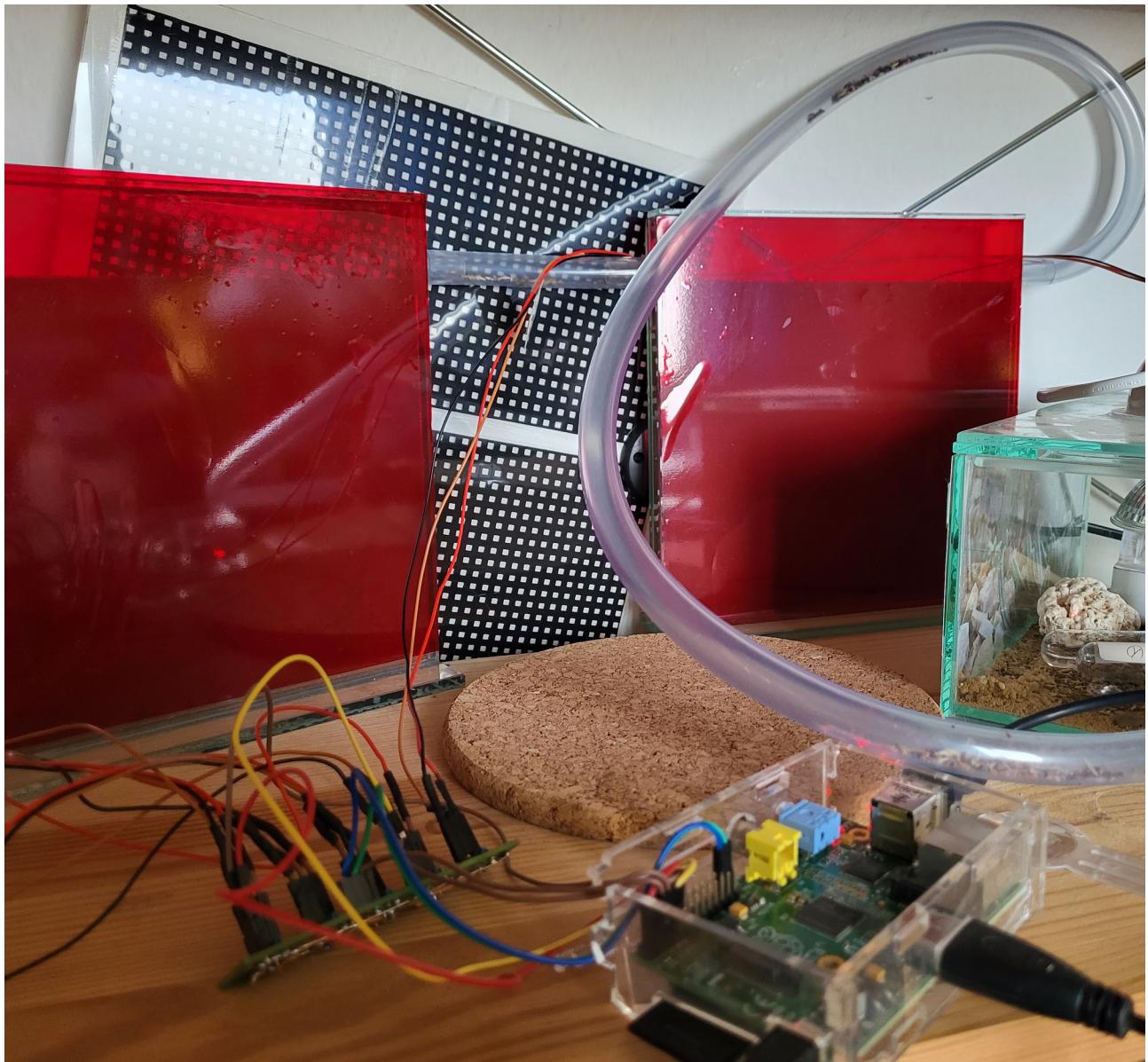


# Raspberry Pi with sensors for temperature and humidity

**Version 1.0 (this help file) for Weather Station 2023 (desktop app)**



© Dipl.-Biol. Björn Zedroßer 2023

# Content

1 Introduction.....	3
2 Project.....	3
3 Soldering the circuit board.....	3
3.1 Item list.....	3
3.2 Connect the circuit board to the Raspi.....	5
4 Sensors.....	6
5 Setup of a formicarium/terrarium.....	6
6 Setup of the Raspberry Pi.....	7
6.1 Headless installation via WiFi.....	7
6.2 User configuration.....	7
6.3 Database configuration.....	7
6.4 DHT2SQL.py.....	8
6.5 Cron job.....	9
6.6 Moving the database to an external Solid State Drive (SSD).....	9
6.7 Security.....	9
7 Weather Station 2023.....	11
7.1 Database setup.....	11
7.2 Species setup.....	13
7.3 Valid species file.....	14
7.4 Sensor configuration.....	15
7.5 Valid sensor.cfg.....	16
7.6 Error codes.....	16
7.7 Github.....	18
7.7.1 WeatherStation2023.pdf.....	18
8 YouTube videos (German).....	19
8.1 Playlist.....	19
8.2 Videos.....	19
9 Stackoverflow Credits.....	20
10 List of software used for this project.....	21
10.1 Raspberry Pi.....	21
10.1.1 Raspberry Pi OS Lite (32-bit) / Raspberry Pi Imager v1.7.3.....	21
10.1.2 MariaDB 10.5.15.....	21
10.1.3 MariaDB Connector for Python 1.0.7.....	21
10.1.4 Python 3.9.2.....	21
10.1.5 pigpio / pigpio Daemon.....	21
10.2 Windows.....	22
10.2.1 DaVinci Resolve.....	22
10.2.2 Win32DiskImager.....	22
10.2.3 Putty.....	22
10.2.4 Microsoft Visual Studio 2022.....	22
10.2.5 Paint.NET.....	22
11 Sources.....	23
11.1 DHT2SQL.py.....	23
11.2 Testing.....	30

# 1 Introduction

In this project a Raspberry Pi will be used to collect temperature and humidity sensor data for several formicariums, which are basically terrariums for ants, but monitoring of any other species is supported. The basic idea is to monitor several of these sensors, even if they are in the same room. But of course, instead of a “species” “room names” could be used to monitor a house or a building.

The project is accompanied by a video playlist on YouTube (in German).

Playlist

It is also available as Open Source Project on [Github](#).

# 2 Project

This project is divided in several steps which could be called milestones. The section of this document contains these milestones which need to completed in the following order:

1. Soldering the circuit board
2. Configure the sensors
3. Setup the objects to be monitored (Setup of a formicarium/terrarium)
4. Setup of the Raspberry Pi to collect sensor data<sup>1</sup>
5. Setup the Desktop app to display data visually (Weather Station 2023)

# 3 Soldering the circuit board

This part of the project is highly depending on what someone wants to achieve. It depends on the RasPi that will be used as well as the number of sensors that should be used. During this project a circuit board for **five** sensors will be used, but the maximum number is limited by the connection pins of the RasPi model used.

## 3.1 Item list

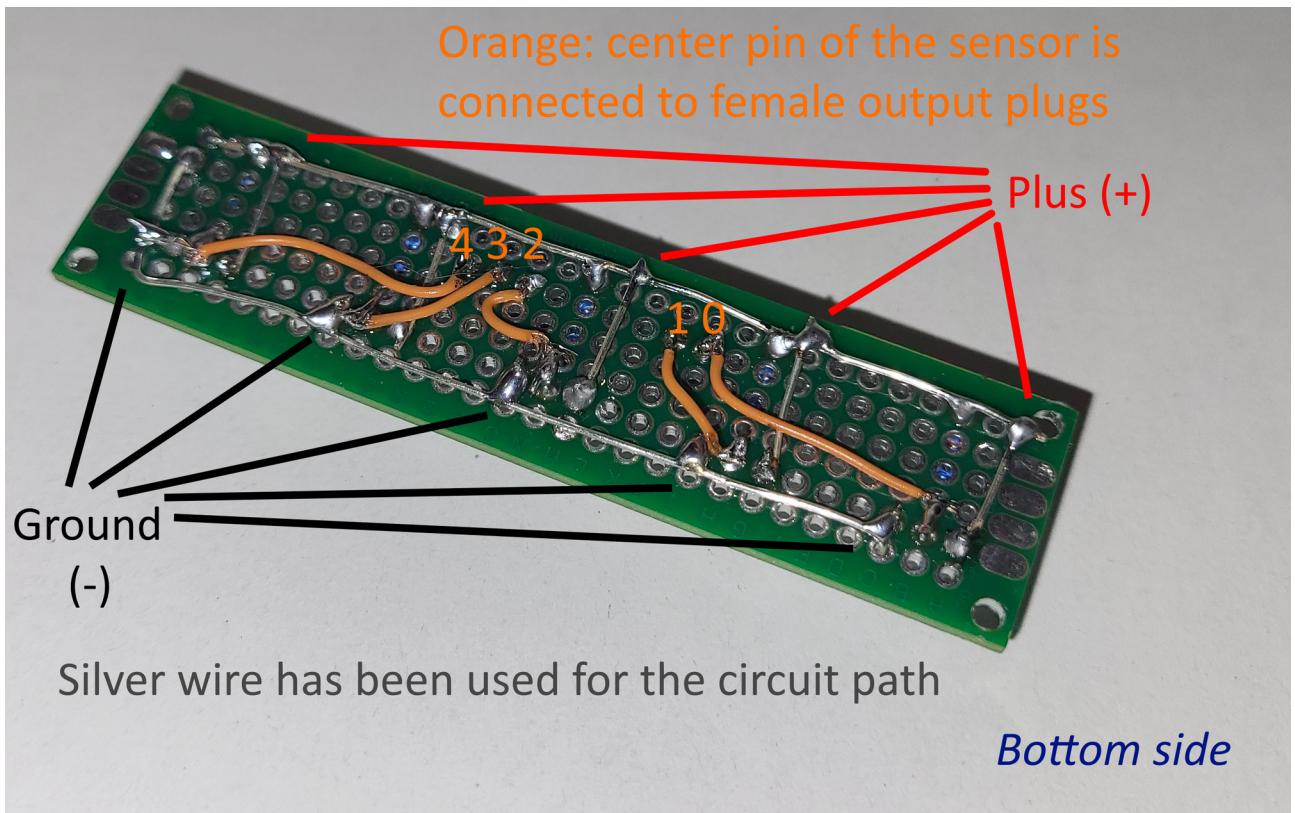
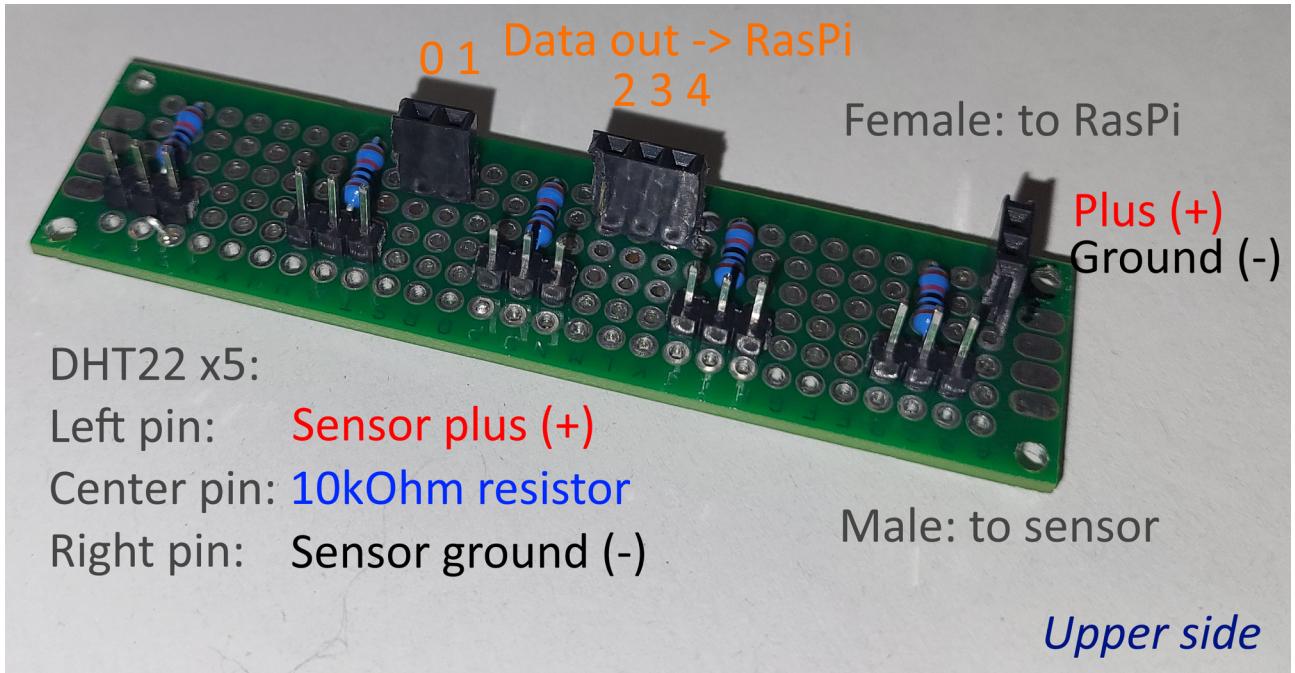
The following was needed for five sensors:

- 5× DHT22 sensors
- 5× 10kΩ resistors
- 1× circuit board (28×6 holes)
- Silver wire
- 7× female plugs (out to RasPi)

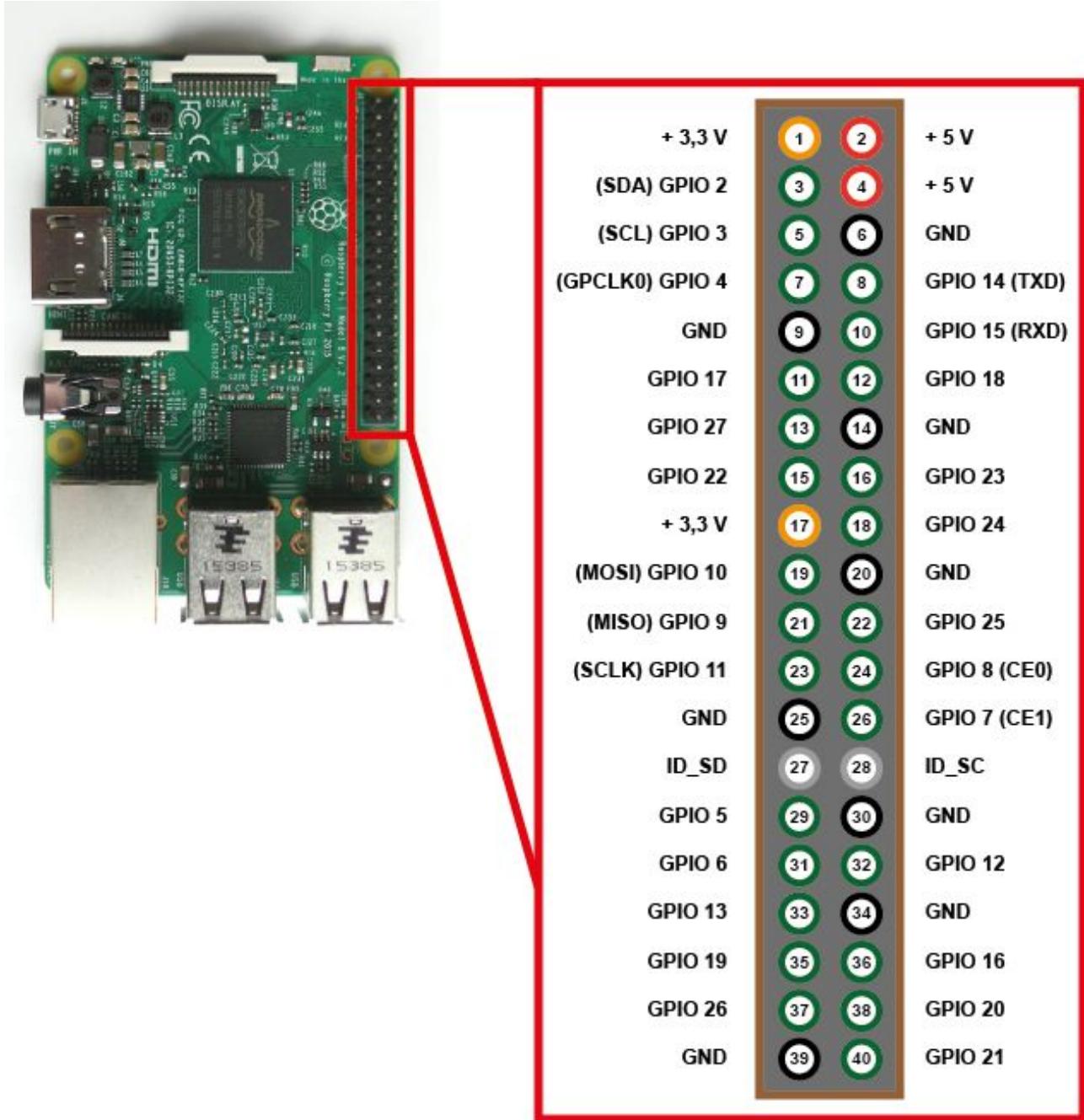
---

<sup>1</sup> The “Raspberry Pi” will in the following be abbreviated as “RasPi”

- 5× 3pin-male (input for the sensors)
- Heat shrink tube
- Wire: 3 different colors (e.g. red, orange, black), 10m each



### 3.2 Connect the circuit board to the Raspi



In this project a very old RasPi is used and the GPIO layout might change for different Raspis or their clones. So it is very important to check the pin layout carefully.

As the layout of the circuit board needs only one +3.3V (1) and one GND (6) using a circuit board saves a couple of connections as normally each sensor would need its own +3.3V and GND lines.

E.g. in this help file the data pins used are the following and thus will be shown in screenshots for the desktop application as examples:

- GPIO 2 → Sensor 2

- GPIO 3 → Sensor 3
- GPIO 4 → Sensor 4
- GPIO 7 → Sensor 7
- GPIO 8 → Sensor 8

## 4 Sensors

The requirement for each sensor is that it could be used between 3 and 5V and that it has got three outputs. In this project DHT22-sensors are used, but any sensor that fulfills the requirements should work.

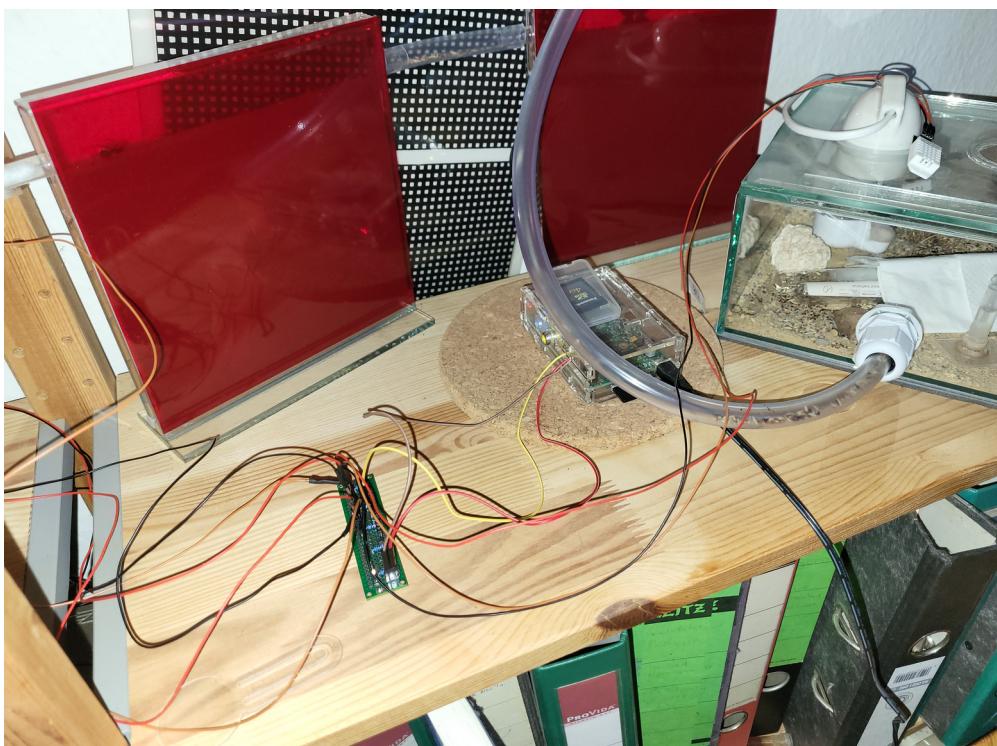
Each sensor owns three output lines:

- GROUND (black)
- DATA (yellow/orange)
- PLUS (red)



## 5 Setup of a formicarium/terrarium

A “formicarium” is a special type of terrarium. They are used in this example for the sensors, but any type of terrarium is supported. Therefore a species name can be defined for each monitoring object. The wires of the sensor should be long enough to connect both, the formicarium and the RasPi.



# 6 Setup of the Raspberry Pi<sup>2</sup>

## 6.1 Headless installation via WiFi

The RasPi can be installed without monitor via SSH-client. Putty can be used to access the RasPi which is obviously a Linux system<sup>3</sup>. There are a lot of installation guides on the internet which might be different according to the version used.

In this project a working RasPi that can be accessed via (W)LAN is a prerequisite and will not be discussed further.

For useful links, please check chapter 10.1 of this document.

## 6.2 User configuration

The user “root” is only to be used to setup the RasPi, but to access the database via a desktop application the user “ant\_admin” is created. Ideally, this user should only have read rights while only the DHT2SQL.py can write to the database.

Follow this guide to setup MariaDB for LAN access and user creation:

<https://mariadb.com/kb/en/configuring-mariadb-for-remote-client-access/>

## 6.3 Database configuration

After MariaDB has been installed, the table which stores sensor data needs to be created.

```
MariaDB [(none)]> create table antstation.raw_data (
-> id int auto_increment primary key,
-> temp float not null,
-> hygro float not null,
-> sensor_id int not null,
-> created_at timestamp);
Query OK, 0 rows affected (0.065 sec)
```

This step is *critical* and the database columns are used for Weather Station 2023 to display sensor data (Database setup). Whenever a column name is different to these defaults, it should be considered throughout this help file.

Also, DHT2SQL.py refers to these database columns.

---

2 This can be totally different for other versions of the RasPi or its clones. Use this is an exemplary guide which steps are needed.  
3 I’m writing this as a Windows user. Linux users might laugh about me and could do it better, but everything in this project was done on a Windows 11 laptop, while the RasPi is controlled via Putty.

```

pi@antstation: ~
+-----+-----+-----+
| mariadb.sys | localhost | mysql_native_password |
| root        | localhost | mysql_native_password |
| mysql       | localhost | mysql_native_password |
| ant_admin   | %        | mysql_native_password |
+-----+-----+-----+
4 rows in set (0.020 sec)

MariaDB [(none)]> grant all privileges on antstation.* to 'ant_admin'@'%' identified by 'Password.....';
Query OK, 0 rows affected (0.006 sec)

MariaDB [(none)]> flush privileges;
Query OK, 0 rows affected (0.007 sec)

MariaDB [(none)]> create table antstation.raw_data (
    -> id int auto_increment primary key,
    -> temp float not null,
    -> hygro float not null,
    -> sensor_id int not null,
    -> created_at timestamp);
Query OK, 0 rows affected (0.064 sec)

MariaDB [(none)]>

```

## 6.4 DHT2SQL.py

This script is used to write data from the sensors via CronJob (6.5) to a MariaDB, thus it needs to have database access as well. The script can be found in *Sources*, but the database connection should be changed. Otherwise the script will not be working.

### DHT2SQL.py

In this script the SQL connection needs to be changed to the root user of the RasPi.

```

233 #ANTSTATION SQL
234
235 try:
236     conn = mariadb.connect(
237         user="root",           # We're using root as this script runs only on localhost
238         password="password",  # I'm using a different pwd, of course ;)
239         host="127.0.0.1",      # localhost, no access from outside
240         port=3306,             # same port as from outside
241         database="antstation" # Ants.Cologne: in this db we're storing our sensor data!
242     )
243 except mariadb.Error as e:
244     print(f"Error connecting to MariaDB Platform: {e}")
245     sys.exit(1)
246
247 # Get Cursor
248 cur = conn.cursor()
# print("SQL works")

```

## 6.5 Cron job

Cron is a job scheduler for Unix-like operating systems, so it's already available and can be used when the script used has been tested successfully. pigpio / pigpio Daemon must also be started via Cron.

```
> sudo crontab -e
```

The following lines are added:

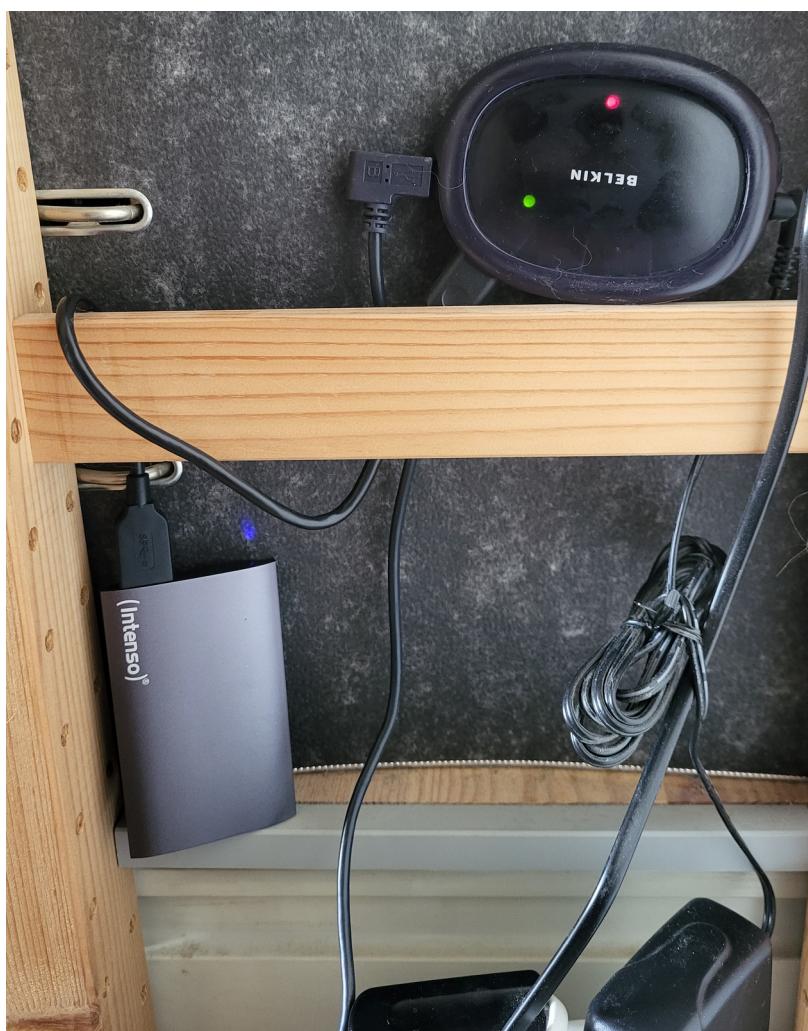
```
* */10 * * * * python3 /home/pi/DHT2SQL.py 2 3 4 7 85
@reboot sudo pigpiod
```

<https://en.wikipedia.org/wiki/Cron>

## 6.6 Moving the database to an external Solid State Drive (SSD)

As the RasPi is used 24/7 as a database server, changing from the default SD-card to an SSD is highly recommended. The SD-card will then only be used for booting while all the load of the database is handled by the SSD which should also result in less latency of common operations (read, write, access via LAN).

But the main reason is, that the first version of this project (Ant Station 1.0, 2021) only worked for 15 months: then the SD-card has crashed, all data was lost and research on the internet showed that SD-cards are not for operating system tasks, but designed and optimized for multimedia use.



For this project the external SSD was connected to an external USB hub with own power supply. This is critical, as the used RasPi can not supply a drive with its own power which could result in making the RasPi inaccessible in the network.

## 6.7 Security

In the context of this project the RasPi is only available in LAN, via SSH and a different user than root for the database connection. There-

own projects.

fore it can only be recommended to follow more in-depth security advises if this solution should be used in other environments.

It is not recommended to use this “home grown solution” in any production environment without further security audits. Feel free to download from Github and modify for your own needs.

## 7 Weather Station 2023

The desktop app can be used after all sensors have been connected to the RasPi, which also writes data via cron job every x minutes to the database.

### 7.1 Database setup

The screenshot shows a 'Connection Setup' dialog box with the following fields filled in:

Setting	Value
App Name	Ant Station 2023
Username	ant_admin
Password	*****
Host	192.168.178.25
Port	3306
Database	antstation
Table	raw_data
Temperature Column	temp
Humidity Column	hygro
Sensor Column	sensor_id
ID Column	id
Created at Column	created_at
Sensors	2,3,4,7,8

At the bottom are 'Save' and 'Cancel' buttons.

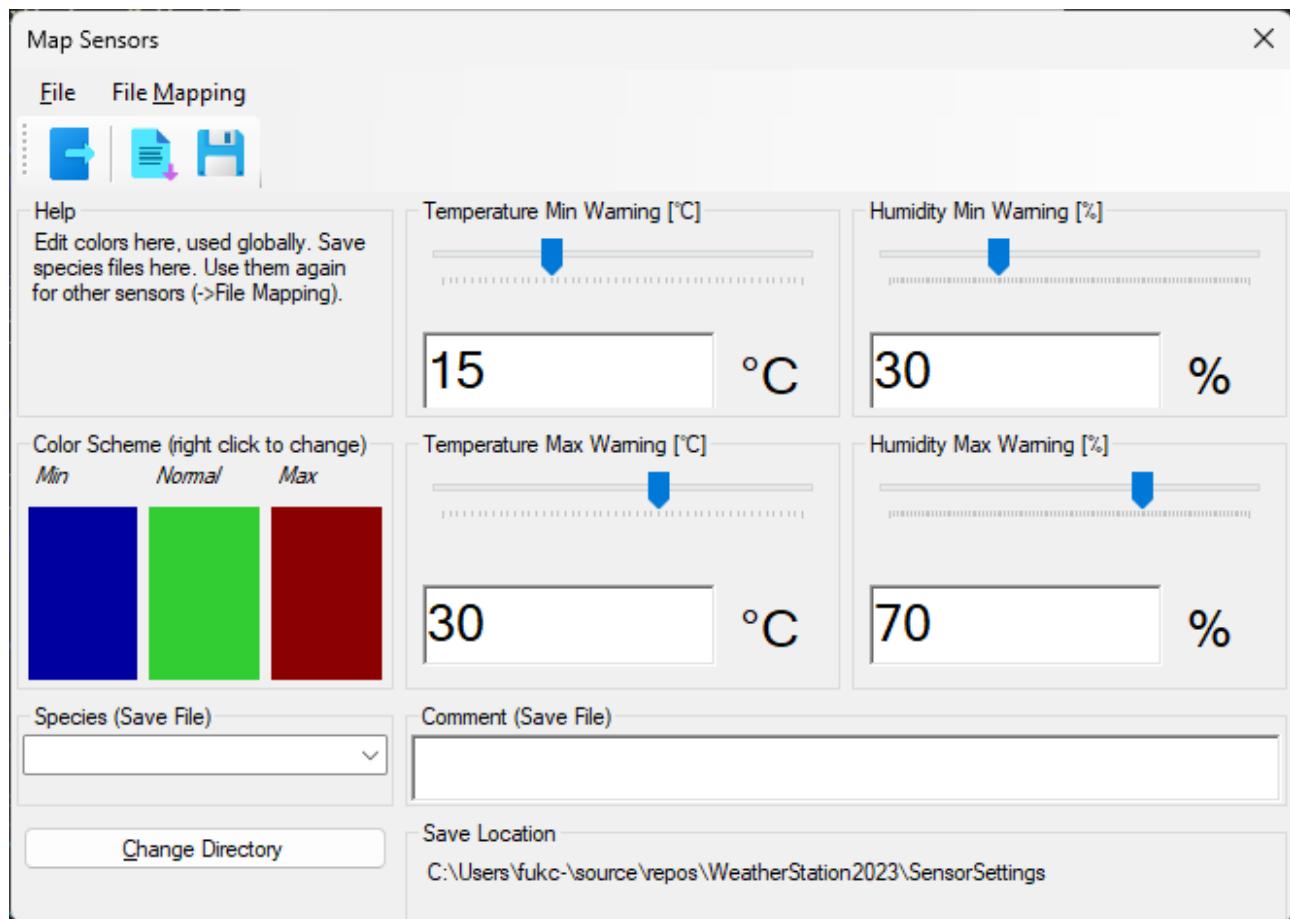
**Without filling in the correct values to this form the app will not run. Make sure that the database and user is set up correctly and that the database is accessible from the client computer.**

<b>App Name</b>	The title of the main window, as shown e.g. in the taskbar, can be changed, e.g. if someone wants to monitor frogs instead of ants. <b>Default:</b> Weather Station 2023
<b>Username</b>	Name of the MariaDB user used to write to the database.
<b>Password</b>	Password of this user.
<b>Host</b>	IP-address of the RasPi – should be LAN only, thus IPv4 is used in this project. IPv6 could work, but was not tested.
<b>Port</b>	MariaDB port <b>Default:</b> 3306
<b>Database</b>	Name of the database as defined during setup.
<b>Table</b>	Table of sensor data used by DHT2SQL.py.
<b>Temperature Column</b>	Name of the temperature column.

<b>Humidity Column</b>	Name of the humidity column.
<b>Sensor Column</b>	Name of the sensor column.
<b>ID Column</b>	Name of the database ID column.
<b>Created at Column</b>	Name of the created_at column (timestamp).
<b>Sensors</b>	List of Sensor-IDs (int), can be separated by “ “ (space), “;” or “,”. A sensor is defined by the pin (sensor id) to which the data cable of the DHT22 sensor is connected. It is the same ID that is used by the Python script DHT2SQL.py for writing to the database.

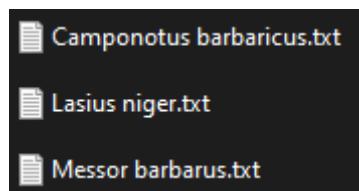
## 7.2 Species setup

Basically, and what makes the difference to other weather monitoring applications, this application is designed to monitor certain animal species in their terrariums (closed spaces).



This form is used to create *species files* ("\*.txt")<sup>6</sup>. As a *species file* could be used by more than one sensor no specific details are used here. It is only used to color the temperature and humidity values, but the color scheme changes also some other coloring. The colors can be changed so that people with e.g. a red-green blindness could set better colors for them which are used globally.

If you want to use the possibility to monitor several different species, you should start creating as many different *species files* as needed, even if they only contain the default values first. Values can be changed later, as long as a *species file* exists.



<sup>6</sup> Although *species files* are basically text files and are easily readable by humans, they should only be created with the application itself as there are certain requirements for a valid file. Bad files loaded during startup of the application might lead to a crash.

## 7.3 Valid species file

```
1 [antconfig 1.0]
2 # Lasius niger - Ants Cologne
3 Lasius niger
4 12
5 25
6 30
7 80
8
```

**Line 1** is essential and **must** be: [antconfig 1.0]

**Line 2** is a comment (only one line allowed) and can be used for authorship, versions, testing etc.

**Line 3** is the name of the species. In other projects it could be used for labels like “living room”, “kitchen”, “bed room” etc.

**Line 4:** Temperature minimum.

**Line 5:** Temperature maximum.

**Line 6:** Humidity minimum.

**Line 7:** Humidity maximum.

**Line 8:** Always empty (EOF)

A valid file has got the “.txt” extension, fulfills the requirements of lines 1, 2 and 8.

**Lines 4 – 7** must contain **integer numbers**.

### Remark:

For ants, there is actually little scientific knowledge about optimal temperature and humidity values. It might be an interesting topic for the ant community to discuss optimal values for each species monitored. The author of this project is monitoring his ant colony especially to get an idea about the correlation of temperature/humidity to colony activity, including the identification of (maybe) existing thresholds.

## 7.4 Sensor configuration

Mapping sensors to files

**File**

Sensor 2  
Alias  File   
C:\Users\██████████ WeatherStation2023\SensorSettings\Messor barbarus.txt

Sensor 3  
Alias  File   
C:\Users\██████████ WeatherStation2023\SensorSettings\Lasius niger.txt

Sensor 4  
Alias  File   
C:\Users\██████████ WeatherStation2023\SensorSettings\Lasius niger.txt

Sensor 7  
Alias  File   
C:\Users\██████████ WeatherStation2023\SensorSettings\Camponotus barbaricus.txt

Sensor 8  
Alias  File   
C:\Users\██████████ WeatherStation2023\SensorSettings\Messor barbarus.txt

Configuration loaded

In this form everything is put together. The “**Choose File**” button loads a *species file* for each sensor and an **alias** can be given.

This is completely optional and can be skipped if no coloring of the sensor data is needed.

If used, everything will be saved in “`sensors.cfg`” in the working directory, which is the only “`*.cfg`”-file in the directory. As it is the only *configuration file* allowed, there is no dialog to select it. Only the last status can be loaded or a changed status be saved.



## 7.5 Valid sensor.cfg

The sensor configuration file contains one line per sensor. It should never be created manually. As there can be multiple installations of WeatherStation2023 on the same system, each one can contain its own configuration file (if different user directories are used), but it is also possible to use the same directory including species files for multiple installations<sup>7</sup>.

```
1 [antconfig 1.0]
2 Sensor 2 Mb1 C:\Users\██████████\WeatherStation2023\SensorSettings\Messor barbarus.txt
3 Sensor 3 Ln1 C:\Users\██████████\WeatherStation2023\SensorSettings\Lasius niger.txt
4 Sensor 4 Ln2 C:\Users\██████████\WeatherStation2023\SensorSettings\Lasius niger.txt
5 Sensor 7 Cb C:\Users\██████████\WeatherStation2023\SensorSettings\Camponotus barbaricus.txt
6 Sensor 8 Mb2 C:\Users\██████████\WeatherStation2023\SensorSettings\Messor barbarus.txt
7
```

**Line 1** is essential and **must** be: [antconfig 1.0]<sup>8</sup>

**Line 2 – 6:** five sensors, tab-separated: *Sensor Name/ID, Alias, Species file*

**Last line (7):** Always empty (EOF)

## 7.6 Error codes

Even though the focus during the development of this application was to make it thread-safe by using background workers and exception handling, still errors can occur, e.g. if the RasPi is not reachable. Errors related to *Background Workers* (Bgw) might show that there are problems with the database, e.g. because the RasPi is busy and does not respond. If restarting the application does not help, it might be a good start to **reboot** the RasPi<sup>9</sup>.

Error code	Component	Message	Solution
0001_BgwMf-<n>	MainForm Background Worker <n>	*	**
0002_BgwSen	Sensor Background Worker	*	**
0003_BgwSsf_r	SensorStatisticsForm Background Worker Read	*	**
0003_BgwSsf_c	SensorStatisticsForm Background Worker Complete	*	**
0005_BgwSf	Statistics Form Back-ground Worker	*	**

<sup>7</sup> Using several installations of WeatherStation2023 is possible, but will not be discussed in detail here. Normally, also multiple databases on multiple RasPis would also be necessary and require a lot more prerequisites. The application title could be used for host identification.

<sup>8</sup> *Species files* and *configuration files* are distinguished mainly by their extension. If the file format changes (other than “antconfig 1.0”), a new version of WeatherStation2023 would be required.

<sup>9</sup> Try if a SSH connection to the RasPi is possible. Enter “sudo reboot” in the prompt. If the RasPi does not respond, disconnect power to restart.

0006_Mfconf	Error loading <i>sensors.cfg</i> to MainForm	<b>Not a valid sensor configuration file!</b>	Delete <i>sensors.cfg</i> and create a new one
0007_Mfconf	Corrupt <i>sensor.cfg</i>		Delete <i>sensors.cfg</i> and create a new one
0008_Mfspec	Error loading species to MainForm	<b>Not a valid species file: + path</b>	Delete <path> and create a new one
0009_Mfspec	Corrupt species file	<b>Corrupt species file: + path</b>	Delete <path> and create a new one
0010_Mfcol	Colors from species file could not be loaded	<b>Colors could not be loaded. Not a valid species file: " + path</b>	Delete <path> and create a new one

\* **Message** is generated by the system. Maybe there will be a collection of common errors in the future, but this is currently not planned. Community feedback would be highly appreciated.

\*\* **Background Workers** are handling multiple database access tasks and can sometimes throw errors, because the RasPi is not available for handling requests. They often work the next time and there is nothing a user can actually do about those errors. Therefore these errors can be switched-off in the application menu.

## 7.7 Github

The C#-code for the desktop application can be found here:

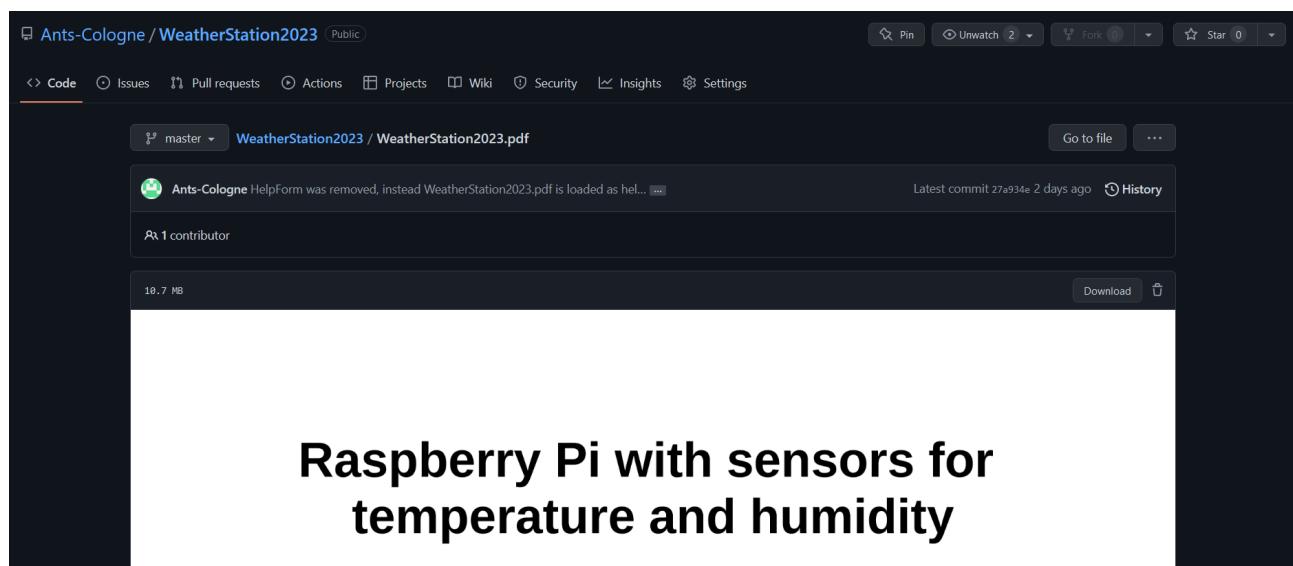
<https://github.com/Ants-Cologne/WeatherStation2023>

### 7.7.1 WeatherStation2023.pdf

This project is still in development and changes to the help file are expected.

If there was an update to the help file, downloading it from Github is sufficient and there is no need to re-install the application. Just copy **WeatherStation2023.pdf** to the installation directory.

<https://github.com/Ants-Cologne/WeatherStation2023/blob/master/WeatherStation2023.pdf>



# 8 YouTube videos (German)

## 8.1 Playlist

This project was accompanied by a German YouTube tutorial playlist which can be found here:

[Ants Cologne RasPi DHT22-Sensor-Tutorial](#)

## 8.2 Videos

If you need more details, maybe one of these videos can give advise, especially about how to solder the circuit board. English translation on YouTube should work.

<a href="#">RasPi DHT22-Sensor-Tutorial (1) - Hardware (1)</a>	Overview circuit board, DHT22 sensor, RasPi.
<a href="#">RasPi DHT22-Sensor-Tutorial (2) - Hardware (2) - Die Platine</a>	Soldering the circuit board, missing hardware needed (silver wire), preparation of the sensors: wire configuration with different length.
<a href="#">RasPi DHT22-Sensor-Tutorial (3) - Hardware (3) - Finales Platinenlayout</a>	Finalization of the circuit board layout for five DHT22 sensors.
<a href="#">RasPi DHT22-Sensor-Tutorial (4): Das Skript</a>	The circuit board has been tested, sensors are connected and now the RasPi is installed. The DHT2SQL.py script is introduced to read data from sensors and write it to the database.
<a href="#">RasPi DHT22-Sensor-Tutorial (5): Drei Sensoren an der Platine</a>	Finalization of the sensor wiring. Setup of the first formicariums.
<a href="#">Messor barbarus Update 18.12.2022 - Winterruhe?!</a>	All sensors connected, some problems still exist. At this point data is continuously written to the database.
<a href="#">RasPi DHT22-Sensor-Tutorial (6) - Die Desktop-Anwendung mit C# (1)</a>	Starting the programming of <i>Weather Station 2.0</i> to solve some problems experienced with <i>AntStation 1.0</i> (2021).
<a href="#">RasPi-DHT22-Sensor-Tutorial (7) - Die Desktop-Anwendung (2): Backgroundworker und Sensor-Klasse</a>	The basic problems with <i>AntStation 1.0</i> have been resolved by using background workers which will now be used throughout the project.
<a href="#">RasPi-DHT22-Sensor-Tutorial (8) - WeatherStation2023</a>	<b>Weather Station 2023</b> was a complete re-design of the Windows application and was the first release that was feature-complete and released on Github .

## 9 Stackoverflow Credits

Some problems were not searchable, so I asked some questions on [stackoverflow.com](https://stackoverflow.com) and want to thank these persons as they helped me when I could not continue:

**IVSoftware** for answering <https://stackoverflow.com/questions/75277474/groupbox-created-in-code-doesnt-dock-correctly-to-parent-tablelayoutpanel>

**IVSoftware** even provided me with a github example of his solution, which was a great help as he gave an example how to use custom user controls, such as a GroupBox, something that I never used before:

<https://github.com/IVSoftware/group-boxes-fill-parent-width>

**Compman2408** for answering <https://stackoverflow.com/questions/75033883/c-sharp-sqlconnection-child-form-closing-but-data-is-not-updated-in-mainform/75034158#75034158>

**Stackoverflow** is a great platform and helped me a lot with answers when searching for C# problems. Therefore: a lot of thanks to **Stackoverflow**!

# 10 List of software used for this project

## 10.1 Raspberry Pi

### 10.1.1 Raspberry Pi OS Lite (32-bit) / Raspberry Pi Imager v1.7.3

Raspberry Pi Imager was used to install the Raspberry Pi OS **Rasbian** on a 32GB SD-card<sup>10</sup>.

<https://www.raspberrypi.com/software/>

### 10.1.2 MariaDB 10.5.15

The database used for this project. Can be accessed via SQL-statements (used in DHT2SQL.py and WeatherStation2023).

<https://mariadb.com/>

### 10.1.3 MariaDB Connector for Python 1.0.7

Needed to connect to MariaDBs via Python scripts. Thus the python script DHT2SQL.py can read sensor data and write it to the database.

<https://mariadb.com/de/resources/blog/how-to-connect-python-programs-to-mariadb/>

### 10.1.4 Python 3.9.2

Was already installed with Rasbian. Necessary to start DHT2SQL.py via CronJob.

<https://www.python.org/>

### 10.1.5 pigpio / pigpio Daemon

Pigpio is the library used to access the General Purpose Input Outputs (GPIOs) of the RasPi. It is used by DHT2SQL.py to access the sensors.

The Daemon is used to start pigpio automatically after rebooting the RasPi, thus it is an essential prerequisite for the Cron job.

```
> sudo apt install pigpio python-pigpio python3-pigpio
```

Library: <http://abyz.me.uk/rpi/pigpio/index.html>

---

<sup>10</sup> An SD-card is not recommended for production use, only for setup and programming. Consider to install the root system of linux (“/”) on an external USB-SSD or hard drive. See 6.6.

Daemon: <http://abyz.me.uk/rpi/pigpio/pigpiod.html>

## 10.2 Windows

### 10.2.1 DaVinci Resolve

I just started with Video Editing for my YouTube channel. But *DaVinci Resolve* is a good start.

<https://www.blackmagicdesign.com/de/products/davinciresolve>

### 10.2.2 Win32DiskImager

Used to backup SD-cards from the RasPi. Also used for installation of images on new drives.

<https://win32diskimager.org/>

### 10.2.3 Putty

I use Putty to connect via SSH to the RasPi. There are other ways, but I use Putty since years.

<https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html>

### 10.2.4 Microsoft Visual Studio 2022

For me the best code editor (IDE) on the market.

<https://visualstudio.microsoft.com/de/downloads/>

### 10.2.5 Paint.NET

I don't have any graphical skills, so everything I need is in this free application. No need for commercial products.

<https://www.getpaint.net/download.html>

# 11 Sources

## 11.1 DHT2SQL.py

This is also available in the Github repository. It is considered as *final* and works without problems in this project. If someone uses this script for own projects, the database access needs to be configured accordingly. **Critical code** of this script which was essential for the described project, has been marked in red color.

Version used: **DHT11/21/22/33/44 Sensor – 2019-11-07**

[http://abyz.me.uk/rpi/pigpio/examples.html#Python\\_code/DHT.py](http://abyz.me.uk/rpi/pigpio/examples.html#Python_code/DHT.py)

The original script DHT22.py should be used to test the sensors and GPIOs used. This is essential for finding the appropriate command which will be used for the Cron job.

```
#!/usr/bin/env python

# DHT2SQL.py / original: DHT.py
# 2019-11-07 / Ants.Cologne: 2022-12-02
# Public Domain
# Modified for use with ANTSTATION by Ants.Cologne 2022 (I only
changed __main__)

import time
import pigpio

DHTAUTO=0
DHT11=1
DHTXX=2

DHT_GOOD=0
DHT_BAD_CHECKSUM=1
DHT_BAD_DATA=2
DHT_TIMEOUT=3

class sensor:
    """
    A class to read the DHTXX temperature/humidity sensors.
    """
    def __init__(self, pi, gpio, model=DHTAUTO, callback=None):
        """
        Instantiate with the Pi and the GPIO connected to the
```

DHT temperature and humidity sensor.

Optionally the model of DHT may be specified. It may be one of DHT11, DHTXX, or DHTAUTO. It defaults to DHTAUTO in which

case the model of DHT is automatically determined.

Optionally a callback may be specified. If specified the callback will be called whenever a new reading is available.

The callback receives a tuple of timestamp, GPIO, status, temperature, and humidity.

The timestamp will be the number of seconds since the epoch (start of 1970).

The status will be one of:

0 DHT\_GOOD (a good reading)

1 DHT\_BAD\_CHECKSUM (received data failed checksum check)

2 DHT\_BAD\_DATA (data received had one or more invalid values)

3 DHT\_TIMEOUT (no response from sensor)

"""

self.\_pi = pi

self.\_gpio = gpio

self.\_model = model

self.\_callback = callback

self.\_new\_data = False

self.\_in\_code = False

self.\_bits = 0

self.\_code = 0

self.\_status = DHT\_TIMEOUT

self.\_timestamp = time.time()

self.\_temperature = 0.0

self.\_humidity = 0.0

pi.set\_mode(gpio, pigpio.INPUT)

self.\_last\_edge\_tick = pi.get\_current\_tick() - 10000

self.\_cb\_id = pi.callback(gpio, pigpio.RISING\_EDGE,

self.\_rising\_edge)

def \_datum(self):

```

    return ((self._timestamp, self._gpio, self._status,
             self._temperature, self._humidity))

def _validate_DHT11(self, b1, b2, b3, b4):
    t = b2
    h = b4
    if (b1 == 0) and (b3 == 0) and (t <= 60) and (h >= 9) and (h
<= 90):
        valid = True
    else:
        valid = False
    return (valid, t, h)

def _validate_DHTXX(self, b1, b2, b3, b4):
    if b2 & 128:
        div = -10.0
    else:
        div = 10.0
    t = float(((b2&127)<<8) + b1) / div
    h = float((b4<<8) + b3) / 10.0
    if (h <= 110.0) and (t >= -50.0) and (t <= 135.0):
        valid = True
    else:
        valid = False
    return (valid, t, h)

def _decode_dhtxx(self):
    """
    +-----+-----+
    | DHT11 | DHTXX |
    +-----+-----+
    Temp C| 0-50  |-40-125|
    +-----+-----+
    RH%   | 20-80 | 0-100 |
    +-----+-----+
    0      1      2      3      4
    +-----+-----+-----+-----+
    DHT11 | check-| 0      | temp   | 0      | RH%   |
          | sum    |         |         |         |
    +-----+-----+-----+-----+
    DHT21 | check-| temp   | temp   | RH%   | RH%   |
    DHT22 | sum   | LSB    | MSB   | LSB   | MSB   |
    DHT33 |       |         |         |         |
    DHT44 |       |         |         |         |
    """

```

```

+-----+-----+-----+-----+
"""
b0 = self._code      & 0xff
b1 = (self._code >> 8) & 0xff
b2 = (self._code >> 16) & 0xff
b3 = (self._code >> 24) & 0xff
b4 = (self._code >> 32) & 0xff

chksum = (b1 + b2 + b3 + b4) & 0xFF

if chksum == b0:
    if self._model == DHT11:
        valid, t, h = self._validate_DHT11(b1, b2, b3, b4)
    elif self._model == DHTXX:
        valid, t, h = self._validate_DHTXX(b1, b2, b3, b4)
    else: # AUTO
        # Try DHTXX first.
        valid, t, h = self._validate_DHTXX(b1, b2, b3, b4)
        if not valid:
            # try DHT11.
            valid, t, h = self._validate_DHT11(b1, b2, b3, b4)
    if valid:
        self._temperature = t
        self._humidity = h
        self._status = DHT_GOOD
    else:
        self._status = DHT_BAD_DATA
else:
    self._status = DHT_BAD_CHECKSUM
self._new_data = True

def _rising_edge(self, gpio, level, tick):
    edge_len = pigpio.tickDiff(self._last_edge_tick, tick)
    self._last_edge_tick = tick
    if edge_len > 10000:
        self._in_code = True
        self._bits = -2
        self._code = 0
    elif self._in_code:
        self._bits += 1
        if self._bits >= 1:
            self._code <<= 1
            if (edge_len >= 60) and (edge_len <= 150):
                if edge_len > 100:
                    # 1 bit

```

```

        self._code += 1
    else:
        # invalid bit
        self._in_code = False
    if self._in_code:
        if self._bits == 40:
            self._decode_dhtxx()
            self._in_code = False

def _trigger(self):
    self._new_data = False
    self._timestamp = time.time()
    self._status = DHT_TIMEOUT
    self._pi.write(self._gpio, 0)
    if self._model != DHTXX:
        time.sleep(0.018)
    else:
        time.sleep(0.001)
    self._pi.set_mode(self._gpio, pigpio.INPUT)

def cancel(self):
    """
    """
    if self._cb_id is not None:
        self._cb_id.cancel()
        self._cb_id = None

def read(self):
    """
    This triggers a read of the sensor.

    The returned data is a tuple of timestamp, GPIO, status,
    temperature, and humidity.

    The timestamp will be the number of seconds since the epoch
    (start of 1970).

    The status will be one of:
    0 DHT_GOOD (a good reading)
    1 DHT_BAD_CHECKSUM (received data failed checksum check)
    2 DHT_BAD_DATA (data received had one or more invalid val-
    ues)
    3 DHT_TIMEOUT (no response from sensor)
    """

```

```

self._trigger()
for i in range(5): # timeout after 0.25 seconds.
    time.sleep(0.05)
    if self._new_data:
        break
datum = self._datum()
if self._callback is not None:
    self._callback(datum)
return datum

if __name__== "__main__":
    import sys
    import pigpio
    import DHT
    import mariadb      #added by Ants.Cologne
    import sys          #added by Ants.Cologne

    def callback(data):
        print("{:.3f} {:2d} {} {:.1f} {:.1f} *".
              format(data[0], data[1], data[2], data[3], data[4]))

    argc = len(sys.argv) # get number of command line arguments

    if argc < 2:
        print("Need to specify at least one GPIO")
        exit()

    pi = pigpio.pi()
    if not pi.connected:
        exit()

#ANTSTATION SQL
try:
    conn = mariadb.connect(
        user="root",           # We're using root as this
script runs only on localhost
        password="password", # I'm using a different pwd, of
course ;)
        host="127.0.0.1",     # localhost, no access from
outside
        port=3306,             # same port as from outside
        database="antstation" # Ants.Cologne: in this db
we're storing our sensor data!
    )
except mariadb.Error as e:

```

```

        print(f"Error connecting to MariaDB Platform: {e}")
        sys.exit(1)

# Get Cursor
cur = conn.cursor()
#print("SQL works")

# Instantiate a class for each GPIO
# for testing use a GPIO+100 to mean use the callback
S = []
for i in range(1, argc): # ignore first argument which is command name
    g = int(sys.argv[i])
    if (g >= 100):
        s = DHT.sensor(pi, g-100, callback=callback)
    else:
        s = DHT.sensor(pi, g)
    S.append((g,s)) # store GPIO and class

while True:
    try:
        for s in S:
            if s[0] >= 100:
                s[1].read() # values displayed by callback
            else:          # Only this is relevant for ANTSTATION
                d = s[1].read()
                print("{:.3f} {:.2d} {} {:.1f} {:.1f}.".format(d[0], d[1], d[2], d[3], d[4]))
                if d[2] == 0:          # We only write to db if sensor data was OK
                    try:
                        cur.execute(
                            "INSERT INTO raw_data
(temp,hygro,sensor_id) VALUES (?,?,?,",
                            (d[3],d[4],d[1]))
                        print("Data written to database. Sensor: " +
str(d[1]))
                    except mariadb.Error as e:
                        print(f"error: {e}")

                    conn.commit()
#time.sleep(2)      #commented out as we're not using the infinite-loop here (which will be done by CRON)
                    break                 #added to leave the while-loop
    except KeyboardInterrupt:

```

```

        break
# Close SQL connection
conn.close()

for s in S:
    s[1].cancel()
    #print("cancelling {}".format(s[0]))           #removed
pi.stop()

```

## 11.2 Testing

The original script DHT22.py can be used for testing the GPIO connections in the console.

```
> python3 DHT.py <sensor(s)>
```

```

pi@antstation:~ $ python3 DHT.py
Need to specify at least one GPIO
pi@antstation:~ $ python3 DHT.py 4
%%%%%%%%%%%%%
Can't connect to pigpio at localhost(8888)

Did you start the pigpio daemon? E.g. sudo pigpiod

Did you specify the correct Pi host/port in the environment
variables PIGPIO_ADDR/PIGPIO_PORT?
E.g. export PIGPIO_ADDR=soft, export PIGPIO_PORT=8888

Did you specify the correct Pi host/port in the
pigpio.pi() function? E.g. pigpio.pi('soft', 8888)
%%%%%%%%%%%%%
pi@antstation:~ $ sudo pigpiod
pi@antstation:~ $ python3 DHT.py 4
1670252292.085 4 0 22.1 41.5
1670252294.162 4 0 22.2 38.8
1670252296.236 4 0 22.2 40.4
1670252298.310 4 0 22.2 40.4
1670252300.385 4 0 22.2 40.3
^Ccanceling 4
pi@antstation:~ $

```

In the case sensor 4 is tested successfully. If no sensor# is given, the error in line 2 is shown.

If the sensor GPIOs are tested, DHT2SQL.py can be tested:

```
> python3 DHT.py <sensor(s)>
```

```
pi@antstation:~ $ nano DHT2SQL.py
pi@antstation:~ $ python3 DHT2SQL.py 4
1670254154.154 4 0 22.2 40.8
Data written to database. Sensor: 4
pi@antstation:~ $ sudo mysql -u root -p
Enter password:
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 45
Server version: 10.5.15-MariaDB-0+deb11u1 Raspbian 11

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> select * from antstation.raw_data;
+----+-----+-----+-----+
| id | temp | hygro | sensor_id | created_at      |
+----+-----+-----+-----+
| 1  | 22.1 | 40.9 |        4 | 2022-12-05 16:27:04 |
| 2  | 22.2 | 40.8 |        4 | 2022-12-05 16:29:14 |
+----+-----+-----+-----+
2 rows in set (0.033 sec)

MariaDB [(none)]>
```

As nothing is shown in the console, the database needs to be checked if sensor data was written correctly.

**Only when everything has been tested successfully the Cron job can be configured.**