Notification platform using air quality sensors and IoT

P2 - A larger program developed by a group

Daniel Møller, Kristian Johansen, Martin Sinkbæk Thomsen, Rasmus Secher

Computer Science

Aalborg University

Denmark

From February To May 2020

Title: Notification platform using air quality sensors and IoT

Theme: IoT Event Manager
Project period: February - May

Project group: C1-3
Participants:

- Daniel Møller
- Kristian Johansen
- Martin Sinkbæk Thomsen
- Rasmus Secher

Supervisor: Ramoni Adeogun

Page Count: 67 Pages
Appendix: Page A to B
Finished: May 12, 2020

AALBORG UNIVERSITET
STUDENTERRAPPORT

Første Studieår
Datalogi
Strandvejen 12-14
9000 Aalborg
http://tnb.aau.dk

Abstract

Abstract here

Preface

This report was written by a group of students from Aalborg University studying Computer Science, written between February 2020 until May the same year. The reason for this choice of topic is that the group study in environments that have a tendency to have bad IAQ, mainly rooms such as auditoriums and group rooms. This report consists of 6 main chapters, going from the introduction to the ending section. Some problems came up during the progress of this report, mainly the outbreak of COVID-19, drastically chanced the work method that was used. The development progress of the report was fairly linear, with first the problem analysis to determine the scope of the problem, followed by modelling, programming and implementation, and lastly experimentation and conclusion.

Glossary

- **IAQ** : Indoor Air Quality
- **RH** : Relative Humidity
- **TVOCs** : Total Volatile Organic Compounds

Table of contents

1 Introduction

The subject IoT Event Manager has many aspects that can be considered. The general idea is to get some data, run that data through some algorithms and give information to a user. The question is what information to analyse. An idea could be to measure the air quality of a room, and give a prediction on when said room becomes uncomfortable to be in.

The issue of predicting when a room becomes unbearable to be in, based on factors such as temperature and CO2, is a subject that has been brought up more often in recent times[2, p. 228]. The reason for this is that more and more studies have shown that bad IAQ reduces productivity, and increases other discomforts such as headaches and sweating[11, p. 1675]. It would not help to simply notify when a room has poor air quality, since that information on its own cannot change the actual air quality. Therefore it is not only interesting to predict when a rooms air quality becomes poor, but also to give some sort of advice or warning beforehand, so that occupants can take precautionary action. These predictions can take use of statistical models and algorithms to help give a better idea on what actions should be taken.

Such actions could be to open a window or to turn on air conditioning in the room. The issue is therefore that it is difficult to know when the air quality has become poor, since the only symptoms humans get are discomfort, and at that point it is already too late to take preemptive measures, and precious work time may have gone to waste. An increase in CO2 has a negative effect on the learning ability of humans[11, p. 1675], as just a CO2 concentration of over 1000 ppm (parts per million) has a significant impact on ones learning ability. This issue can be reduced or removed entirely by properly informing occupants in a room in time, and thereby helping their learning ability[7]. This is a project that will make use of IoT subjects, since it is very useful to be able to place sensors anywhere with an internet connection. This will therefore be a project about making an IoT event manager that is capable of giving preemptive warning on poor air quality in a room.

2 Problem Analysis

2.1 What is bad air quality?

There are several factors in play when it comes to determining indoor air quality (IAQ), such as ventilation rates, total volatile organic compounds (TVOCs), relative humidity (RH), CO2 and temperature. These parameters all have an impact on the quality of the air[2, p. 229].

According to a literature review[2], ventilation rates over 8 l/s-p (litres per second per person) and CO2 concentrations below 1000 ppm has shown an improvement in health and comfort for occupants in the room in question and can cultivate the learning process. Furthermore, lower ventilation rates showed increased nasal patency, asthmatic symptoms and risk of viral infections. When combined with increased CO2 concentrations, it also showed impaired attention span, concentration loss and tiredness. Moreover, increased CO2 concentrations also resulted in increased concentration of TVOCs.[2]

Indoor moulds also showed a strong relation to IAQ and were directly related to temperature and RH, in a way so higher temperature and higher rates of RH showed an increased concentration in indoor moulds.[2]

By looking further into the previously mentioned literature review[2], thresholds of the parameters indicating bad air quality in an indoor environment can be established. The following thresholds will be used in this report[2, p. 252]:

CO2: 1000 PPM

Temperature: 25 Degrees

RH: 45%

These thresholds were chosen of various reasons, all derived from the literature review[2]; CO2 levels are determined to be 1000 PPM, relative humidity is determined to be 45% and temperature is determined to be 25 degrees. If one of the parameters exceeds its threshold, the IAQ can be seen as being bad.

To sum this up, five parameters have been looked in to; ventilation rates, TVOCs, RH, CO2 concentrations and temperature. Measuring ventilation rates is difficult without access to the ventilation system in question and as such ventilation rates will not be covered in this project. CO2 concentration, temperature and RH can, however, be measured with a sensor that can be put anywhere, and will therefore be considered in this project. TVOCs can also be measured, however it requires specialised equipment that is fairly expensive[3][5][4] to work with and can be difficult to get correct measurements on, therefore TVOCs will not be considered in this project.

2.2 Stakeholders

The effect on humans in bad IAQ is universal, and it does not matter what institution or workplace it is on (SOURCE). This being said this project will focus on schools and universities, since it is a center of learning, and where bad IAQ have the biggest impact (SOURCE). The reason for the greater impact is that children and young people are affected much more than adults[2], and have lead to more strict building requirements when it comes to schools. This however does not mean that it is always followed correctly, and even a cost saving action that on an administrations level seems fine, can have a serious impact on the students.[source]

When it comes to indoor air quality on schools and universities, there are several stakeholders included. In the front line, there are the students and teachers, who are getting the direct effect of bad air quality. A humans focus and work performance is significantly decreased by high levels of CO2 and other air pollutants by as much as 95% effective work in basic activity, down to 50% efficiency[6, p.4]. From the same report, this means that an average CO2 level of over 1000 ppm have a significant impact on how well the students perform. Since the same effects would happen to the teachers, it means that their performance also decrease, which in turn could result in a worsened learning experience by the students. There have also been conducted studies about the amount of complaints that are connected to ventilation types[9], and they have shown that buildings with bad ventilation have an increase in complaints. A room that can be used as an example, is an auditorium at a university. Here there are usually a lot of students in a single room, and it results in a large amount of CO2 and heat being generated. If improper ventilation is installed, this could lead to a bad learning experience for the students.

Administration is also a part of this problem, not only if their own offices have bad indoor air quality, but also on the fact that their job is to make sure that students learn. It have been found that CO2 concentrations higher than 1000 ppm results in an increase in absenteeism by 10-20%[2], this brings the issue to the administration, since this is what they are supposed to prevent.

Another stakeholder to consider is the buildings management staff, whose job is to make sure that all the buildings system works as they should. That include the ventilation system installed, if any. It may be of use for this staff to have information on what to do with rooms that have bad IAQ, and with that information being able make their job easier.

It can thereby be concluded that the issue is present at all levels of a school, however it is those who are in the front line of the problem, eg. students and teachers, that feels the problem the most. Therefor the focus of this report will be to improve the situation for students and teachers.

2.3 Existing systems

The issue of measuring bad indoor environment is not a new one, as several other studies have been conducted on the same topic[1][10][8]. These systems have in common to evaluate the indoor air quality, and send some sort of information to a user. The way that these systems achieve this, is by having one or multiple small devices capable of monitoring air quality in a facility, that transmit data to a server. The general idea can be seen, with inspiration from from the source[10, fig. 1]:

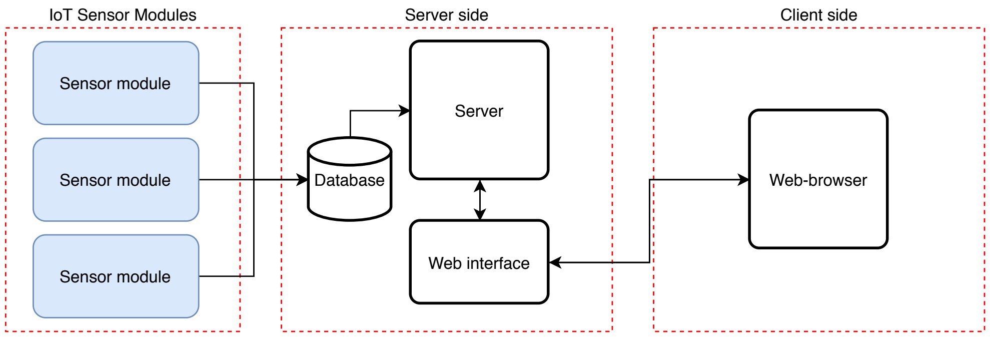


Figure 1: *Figure with inspiration from from[10, fig. 1]*

All the existing systems are based on this, since it is easy to implement and maintain. It is easy to implement because the sensor modules are a fairly standalone unit, which may require connection to the mains power supply, while others rely on battery. Then the only other thing that is missing is a server, to take care of the data that is being send from the sensors. The unit themselves consist of a sensor and some kind of micro controller with access to the internet. An example of such micro controller could be an ESP2688[12].

The web server in question can be a very simple one, that simply just stores sensor values and makes then accessible to a user by a web interface. This interface is in those three examples written in HTML with some simple JavaScript behind it. Most of these systems consist of a lot of sensors, that all send data to a server. The reason to have multiple sensors spread out, is that you can get a much more precise view of the entire facility, than you can get from just a single sensor. An example of how spread out the sensors can be, can be seen in the following image, with inspirations from sources:[10, fig. 5]:

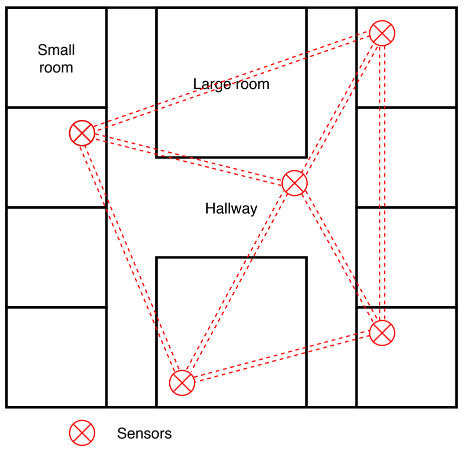


Figure 2: *Figure with inspiration from from[10, fig. 5]*

Where it can be seen, that these units have been spread out to give a better overview on how the IAQ is. What can be done different from these other systems, is actually put it to use and make a notification system, to inform the occupants of a room about bad IAQ. This system must be able to predict when bad IAQ is going to happen, and give some preemptive actions to the user.

2.4 Thesis statement

The problem of bad IAQ have been analysed, and it have been concluded that the problem is indeed real, and that it have large implications. Parameters such as CO2, RH and Temperature have been determined to be the most significant factors to bad IAQ. The stakeholders of the students and teachers have also been determined, since they are in the front line of the problem. Several other systems have been identified as well, as well as how this groups target differs from such said systems. Following this is the solution section, where a set of requirements for the solution is set up, as well as documentation on it. A thesis statement can now be made, to connect the problem analysis and the solution section:

How can a web-application be developed to monitor Indoor Air Quality (IAQ) parameters and generate preemptive alerts to appropriate stakeholders on recommended actions when abnormal IAQ level is detected?

2.5 Requirements

A look may now be taken on what requirements a solution for this problem could have. In order for the solution to function as intended, several requirements have been set for the solution. A table of these requirements and a short description can be seen in Table 1:

Requirements	Description
Website	The program consists of a website, which is the platform users will utilise
Node.js Server	The program will run on a server, which keeps track of all backend data
SQL Database	The server will also have a database, containing all sensor data
Specific languages	The website should be written in HTML/CSS/JS and the server in Node.js
Multiple sensors	The program should take input from multiple sensors to evaluate different input
Prediction system	The program should be able to make predictions on a rooms air quality
Warning system	The program should warn the user if the IAQ becomes too low
Admin Page	The program should have a managed interface to edit room/sensor/threshold data

Table 1: Requirements for solution

A website would be a fitting platform, since it would not require any other programs installed other than a browser, and access to the internet. It is therefore easy to access the platform, and can be opened up on the fly. To complement this website, a server is needed to host the website and act as the interface where the client can get sensor data, prediction data and warnings from the database. The server will be made in Node.js, since it is easy to work with. The server will get its data from an SQL database, this will make it easy to store a vast amount of data and the data storage easily expandable and flexible. Since all data is gathered through the server by an interface, no direct client-to-database interaction is possible, hence enhancing security. The languages that will be used are HTML, CSS and JS for the client, since they are easy to work with and makes the platform easy to develop in a short timeframe. The system must be able to employ a multitude of sensors, simply to increase the dataset and thereby make more accurate predictions. The prediction system must be able to take historical data from the database and make accurate predictions on how the IAQ will progress. These predictions will be used to give the users a useful warning on when the IAQ becomes bad and some possible solutions on what to do about it. Lastly the admin page is to give an admin an easy and simple way to edit sensor info, room info and sensor thresholds, so that one does not have to do it manually to the database.

2.6 Feasibility

The feasibility of the previous requirements can be accessed to determine if they are plausible to uphold. This section will base the feasibility on how other projects were made[1][10][8]. These projects are described in the "Existing Systems" section.

The creation of a website and a server is not something new and those three projects use websites. The use of a database is as well not something new, since it is one of the most widely used methods of data storage. The languages HTML, CSS and JS is also widely used, and was also used in those three projects[10, p.63]. All those three systems employed multiple sensors, meaning it is also possible to include in this project. The creation of a prediction system is however something the three other systems did not make. It is however not impossible to make, since one can make the use of prediction algorithms to accurately predict such. If such predictions can be made, then it is an easy task to implement a warning system, that simply displays it to the user, as well as some possible solution to the issue.

3 Model

The model consists of several sub-parts, however a general overview can be seen in figure 4:

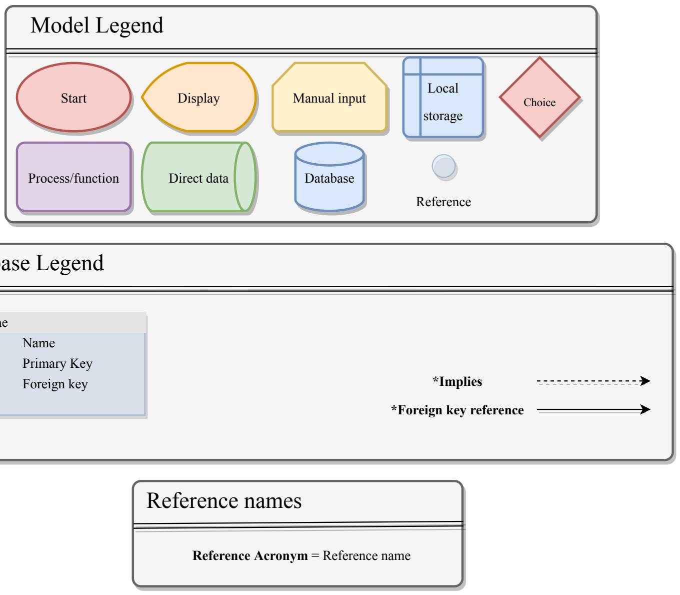


Figure 3: *Legend of the items used in the following models*

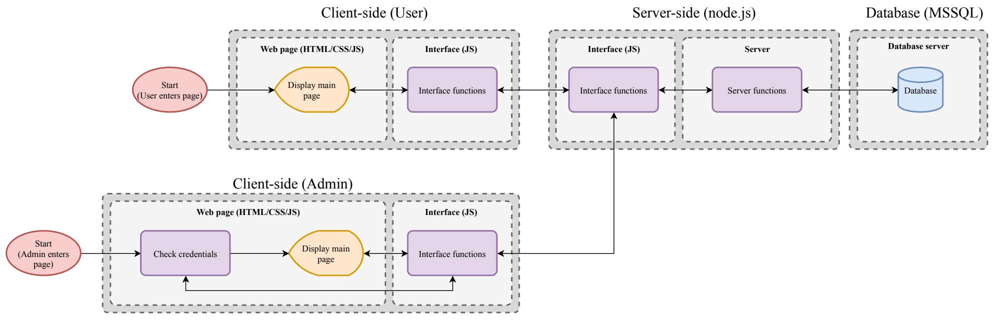


Figure 4: *Overview of the following models*

3.1 User side

The solution is split into two parts, the first part is for the users and the second part is for admins. The user section will be looked at first

3.1.1 User Client-side

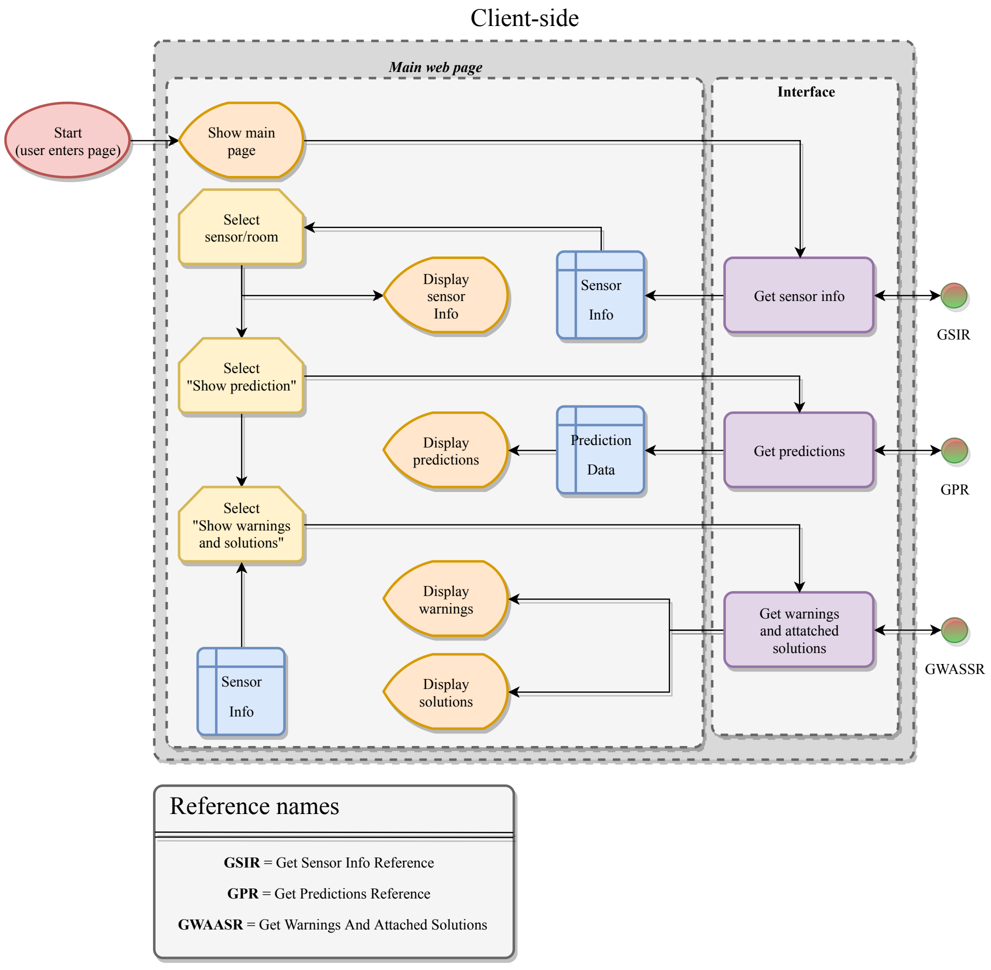


Figure 5: *Model of client-side*

The model for the user client-side, as can be seen in figure 5, consist of a single web page and an interface. The reason for a single web page is that there are not that much static data to display. When a user enters the web page, it calls the server to get general sensor info. This info consists of rooms, what sensors are in that room, sensor types, etc..

This data is then used to populate a dropdown menu, that have all the rooms that the system have. The user can then select a room, where the general data on that rooms is displayed, and an option is given to get prediction data on when the IAQ gets bad. This again calls the server to get data, and returns it to local storage and shows it to the user. The method for showing this data can be in a graph, or simply just a number until bad IAQ.

Lastly the user gets an option to get warnings and solution based on the prediction data. If the user selects it, the web page will again send a request to the server for data, and in return simply directly display it.

The interface that is there to fetch the server for info. The method that this interface will work, is by using query strings[13].

3.1.2 User Server-side

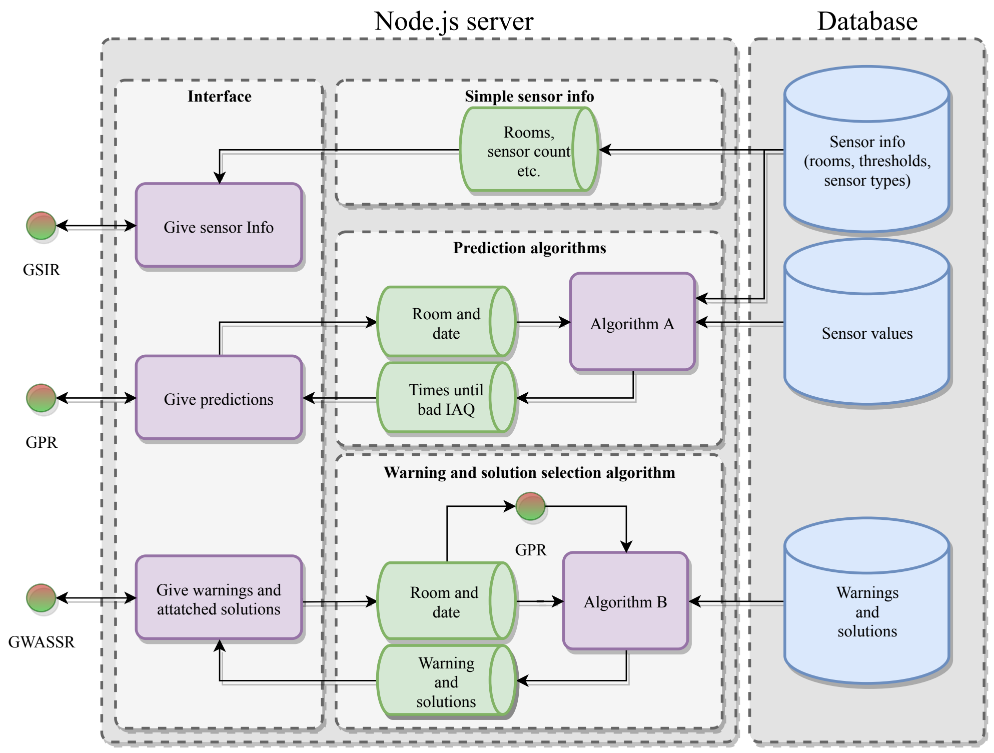


Figure 6: *Model of server-side*

The model for the user server-side code of the website can be seen in figure 6, which consists of two large boxes; Node.js server and Database. The database will be explained later in this section. The Node.js server consists of several smaller boxes; Interface, Simple sensor info, Prediction algorithms and Warning and solution selection algorithm. Furthermore the Node.js draws info from the Database through various queries. Simple sensor info gets data attached to sensors from the database through a query, such as which room a sensor belongs to, how many sensors there are, what type a sensor is etc.

This data is then sent to the interface.

Prediction algorithms receives a specific room and data from the interface and sends it into Algorithm A, which contains one or more algorithms designed to predict when the air quality of the given room will turn bad. This algorithm will output an array of the times of day that the IAQ have been bad historically. A more detailed description comes after this section.

Warning and solution selection algorithm receives some sensor info and a time until the IAQ turns bad, calculated in the Prediction algorithms box, which it sends into Algorithm B containing an algorithm to determine which warning and solutions it should display and recommend. This data is then sent to the interface.

The interface is where the server receives requests from the client-side and responds with info or data related to the request. The server gives three different types of responses; Give sensor info, Give prediction and Give warnings and attached solutions. In Give sensor info the server receives a request for sensor info and the server pulls that data from the Simple sensor info box and then sends the data to the client through a response. The procedure is the same for the two other responses, only the request and the data the server responds with is different; in Give predictions the server responds with the estimated time until the IAQ turns bad and in Give warnings and attached solutions the server responds with a warning and possible solutions.

Server-side Algorithm A

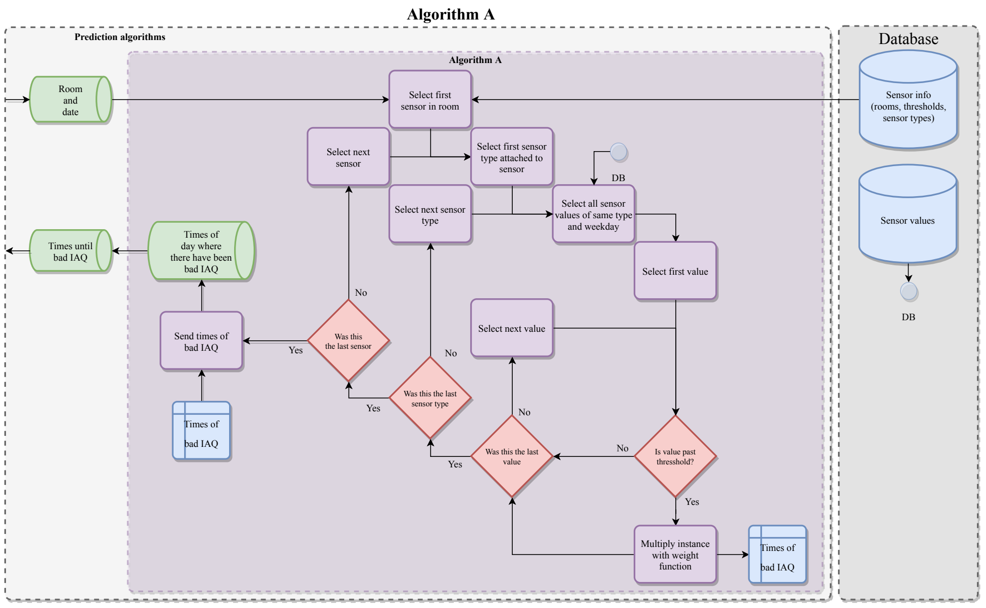

Algorithm A
Prediction algorithms
Algorithm A
Database
Room and date
Select first sensor in room
Sensor info (rooms, thresholds, sensor types)
Select next sensor
Select first sensor type attached to sensor
DB
Select next sensor type
Select all sensor values of same type and weekday
Sensor values
Times until bad IAQ
Times of day where there have been bad IAQ
Select first value
DB
No
Send times of bad IAQ
Yes
Was this the last sensor
Select next value
No
Yes
Was this the last sensor type
No
Times of bad IAQ
Was this the last value
No
Is value past thresshold?
Yes
Yes
Multiply instance with weight function
Times of bad IAQ

Figure 7: *Model of Algorithm A*

The Algorithm A is used to get prediction data based on what room is given, and the time of day. The algorithm will run through all the sensors in a room, and select the data entries that match the weekday given. E.g. if the date "Friday" is given, then the algorithm will choose all the previous entries that happened on a Friday. The different sensor types are then evaluated, to see if they have passed their preset thresholds, and if they have, the time and value is put into a local storage, to be send later. This continues for all the sensors in a room. In the end, all the data on bad IAQ is send out of the algorithm.

The way the data is returned is the time of the day split up in intervals, this could be 5 min intervals. These intervals are then populated with the times that there have been bad IAQ in the past, so that if there are bad IAQ two times in the same interval, the count of bad IAQ is added up. This will make it easy to show the predictions in a graph like manner, since all the data can now be presented in an x-y coordinate system. A example of such a graph can be seen in figure 8:

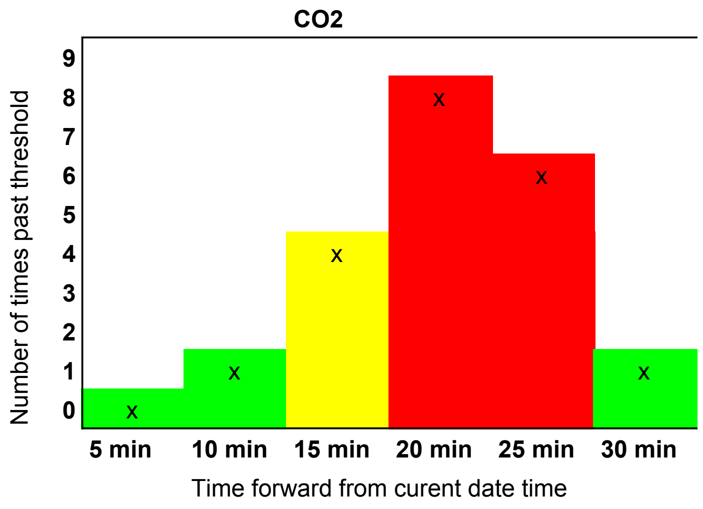


Figure 8: *Return style of Algorithm A*

Server-side Algorithm B

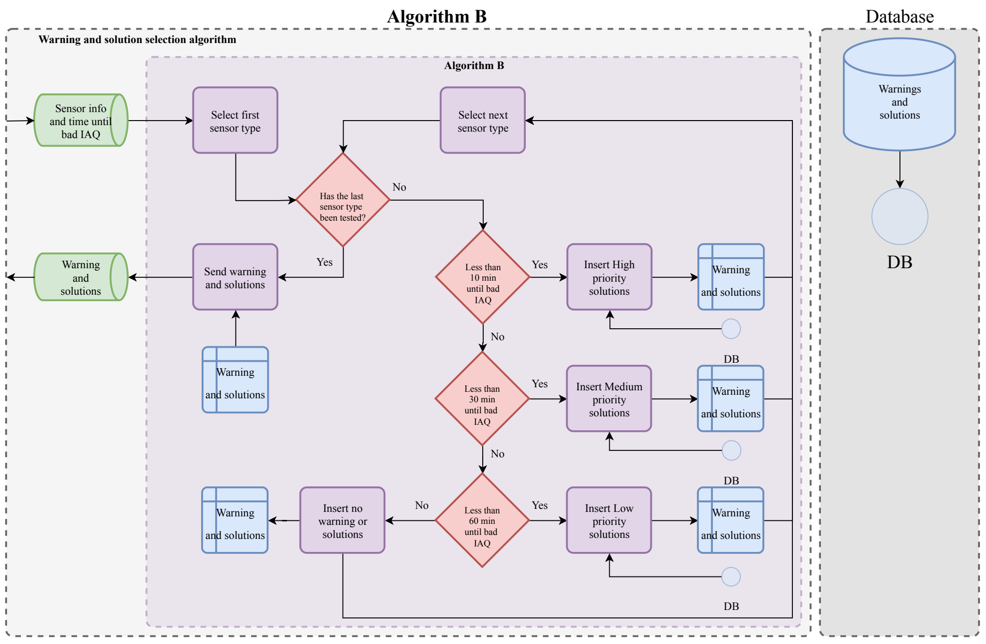

Algorithm B
Database
Warning and solution selection algorithm
Algorithm B
Sensor info and time until bad IAQ
Select first sensor type
Select next sensor type
Has the last sensor type been tested?
No
Yes
Warning and solutions
Send warning and solutions
Less than 10 min until bad IAQ
Yes
Insert High priority solutions
Warning and solutions
No
DB
Less than 30 min until bad IAQ
Insert Medium priority solutions
Less than 60 min until bad IAQ
Insert Low priority solutions
Insert no warning or solutions
Warnings and solutions
DB

Figure 9: *Model of Algorithm B*

The model for the server-side Algorithm B shows the last of the algorithm where warnings are put into action. The algorithm is fed with information such as sensor info and time until bad IAQ. From here it flows through the model starting with the module of selecting the first sensor type and then checking whether or not it is the last sensor available.

Each sensor goes through a set of if statements to check whether the IAQ will exceed their thresholds within the next 10 min, 30 min or 60 min. These intervals correspond to one of three tiers of solutions being either ”high priority”, ”medium priority” or ”low priority”. If one of these if-statements comes out as true a sufficient warning is then found from the ”Warning and solutions” database which corresponds to the priority given and the given sensor info.

For example there could a situation where a prediction has been conducted and informs that there is less than 10 minutes until the room enters a state of bad IAQ. This will then trigger the ”high priority” tier and will then find sufficient warnings and solutions to combat this. An example of such a solution could be to take a break from work and leave windows and doors in an open position.

In the case that a sensor type does not trigger a true statement from neither of the three if-statements the next sensor type will be chosen and the cycle continues. If a warning is activated it will be transferred to local storage of the server through the ”warning and solutions”. The information is then sent back to the ”warning and solution” module on the server. Multiple warnings can also be sent, e.g. if both CO2 and temperature is too high.

3.2 Admin side

The admin side of the solution also consists of a client and server part. These models are quite large, since they consists of a lot of get and set functions, however the logic off the admin side is very simple.

3.2.1 Admin Client-side

The admin client side consists of several smaller sections, starting from figure 10:

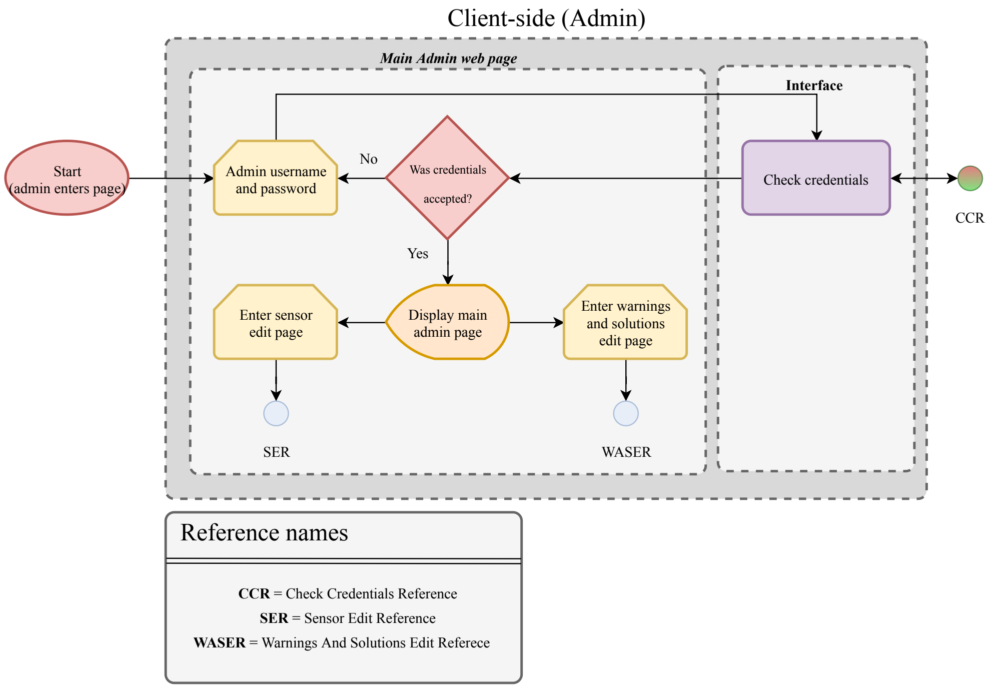

Client-side (Admin)
Main Admin web page
Interface
Start (admin enters page)
Admin username and password
No
Was credentials accepted?
Check credentials
CCR
Yes
Enter sensor edit page
Display main admin page
Enter warnings and solutions edit page
SER
WASER
Reference names
CCR = Check Credentials Reference
SER = Sensor Edit Reference
WASER = Warnings And Solutions Edit Referece

Figure 10: *Model of the admin client-side*

The start of the admin page is very simple, the user is prompted to input a username and a password, where it is then checked in the server to see if it is correct. If it is, the admin is given two options, to go to the sensor edit page or the warning and solutions edit page.

The Sensor edit side (the SER reference) will be shown first.

SER Part 1

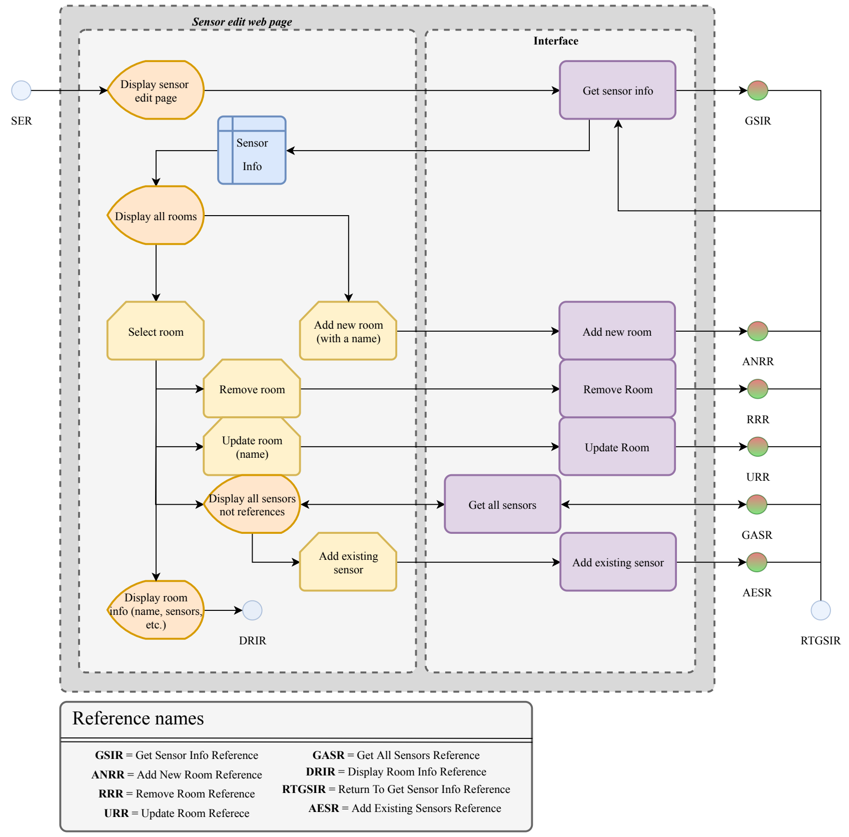

Sensor edit web page
Interface
SER
Display sensor edit page
Get sensor info
GSIR
Sensor Info
Display all rooms
Select room
Add new room (with a name)
Add new room
ANRR
Remove room
Remove Room
RRR
Update room (name)
Update Room
URR
Display all sensors not references
Get all sensors
GASR
Add existing sensor
Add existing sensor
AESR
Display room info (name, sensors, etc.)
DRIR
RTGSIR
Reference names
GSIR = Get Sensor Info Reference
ANRR = Add New Room Reference
RRR = Remove Room Reference
URR = Update Room Referece
GASR = Get All Sensors Reference
DRIR = Display Room Info Reference
RTGSIR = Return To Get Sensor Info Reference
AESR = Add Existing Sensors Reference

Figure 11: *Model of the SER side part 1*

This is the top of the SER model. The page starts with prompting the server for sensor info, saves the data and lastly displays it. The admin can now choose to select a room, or add a new one. If the admin chooses to add a new one, the admin will give it a name, and send it to the interface, that calls a function on the node.js server. When that function have been called, the program returns to the top to get fresh sensor info from the server.

If the admin chooses a room, the admin is given the option to edit the room name or to remove the room, as well as displaying what sensors are attached to the room, and another box that shows all the sensors that are not attached. The admin can then choose to add an existing sensor to the room.

SER Part 2

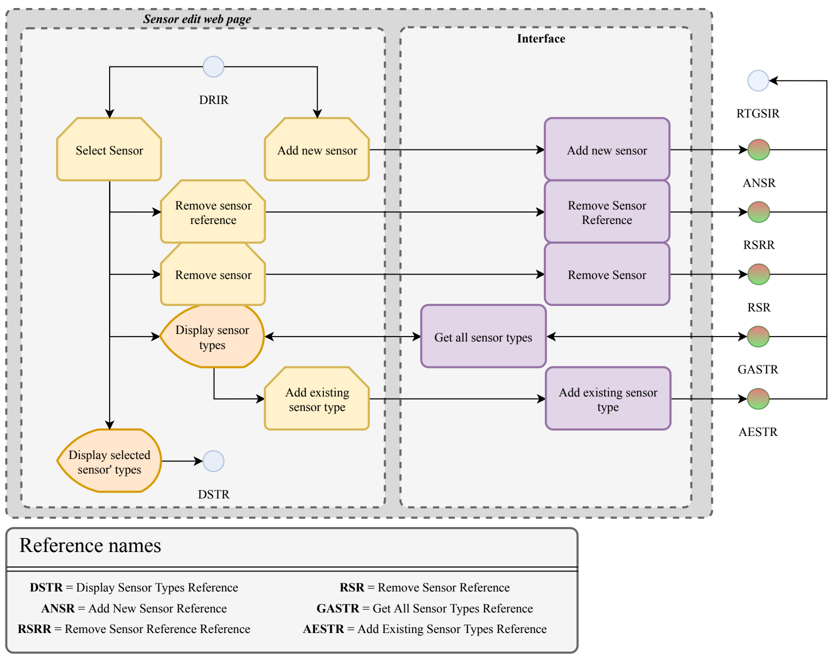


Figure 12: *Model of the SER side part 2*

The next part of the SER page, is where the admin can choose to make a new sensor, or select a sensor thats already there. The admin can then choose to remove the sensor from the room, or remove it from the entire system. Lastly the sensor types is shown just as the room showed what sensors it have. These types could be RH, temperature or CO2, or some other type.

SER Part 3

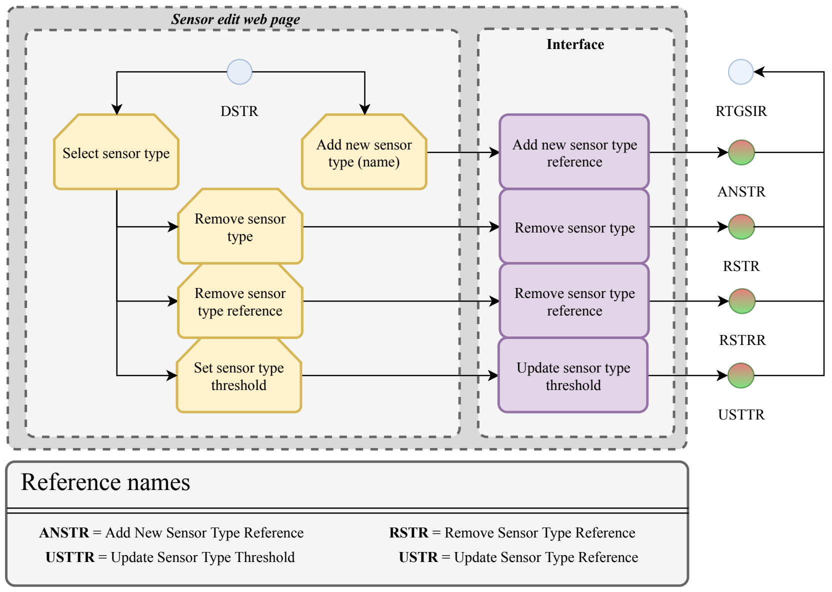
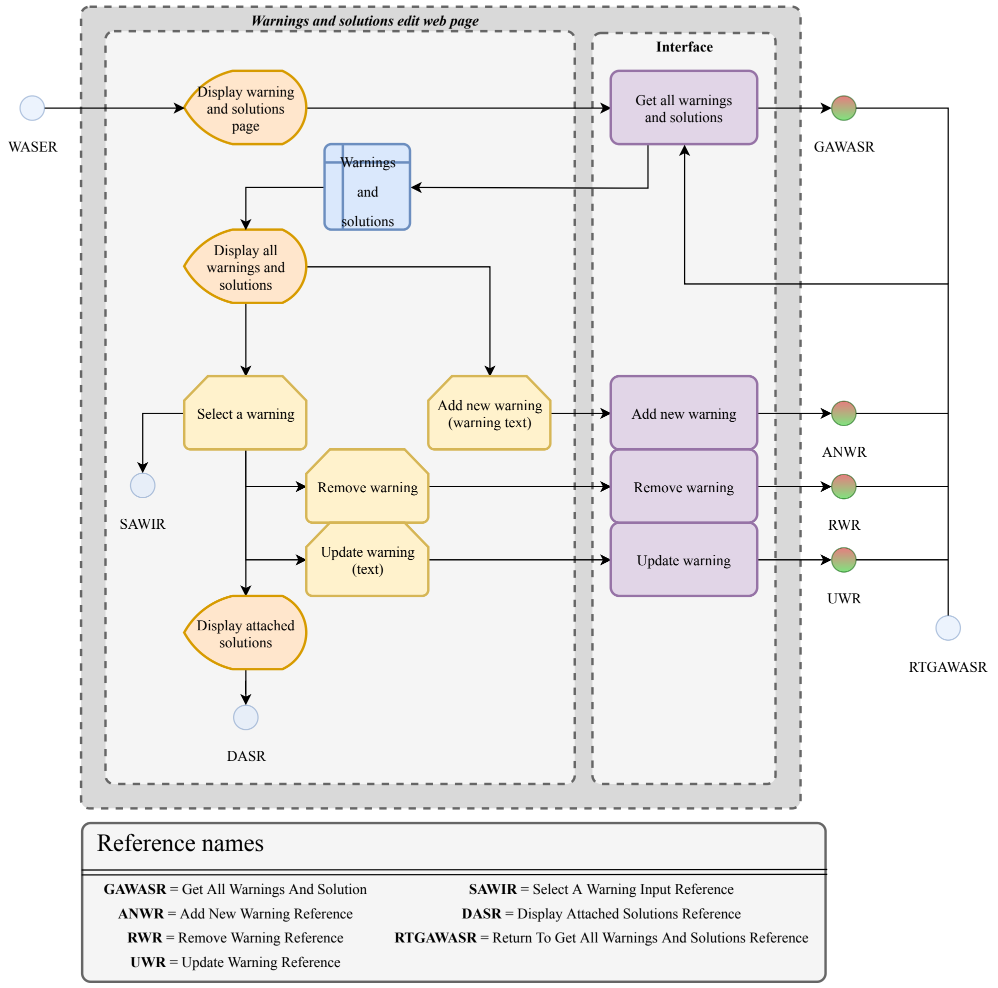

Sensor edit web page
Interface
DSTR
Select sensor type
Add new sensor type (name)
Remove sensor type
Remove sensor type reference
Set sensor type threshold
Add new sensor type reference
Remove sensor type
Remove sensor type reference
Update sensor type threshold
RTGSIR
ANSTR
RSTR
RSTRR
USTTR
Reference names
ANSTR = Add New Sensor Type Reference
RSTR = Remove Sensor Type Reference
USTTR = Update Sensor Type Threshold
USTR = Update Sensor Type Reference

Figure 13: *Model of the SER side part 3*

The last part of SER is much like the other parts, however here there are also an option to update the threshold of a sensor type.

WASER Part 1

The Warning And Solution Edit Reference (WASER) side of the admin page is next.

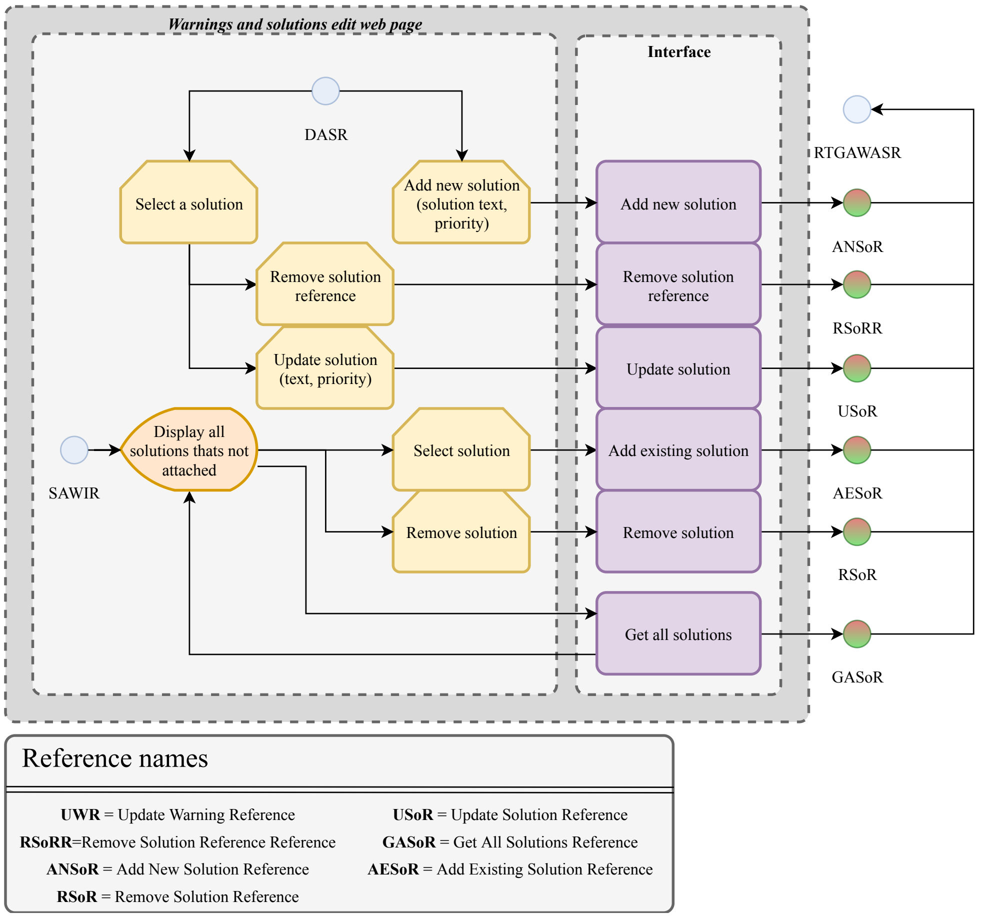

Warnings and solutions edit web page
Interface
WASER
Display warning and solutions page
Get all warnings and solutions
GAWASR
Warnings and solutions
Display all warnings and solutions
Select a warning
Add new warning (warning text)
Add new warning
ANWR
SAWIR
Remove warning
Remove warning
RWR
Update warning (text)
Update warning
UWR
RTGAWASR
Display attached solutions
DASR
Reference names

GAWASR = Get All Warnings And Solution	**SAWIR** = Select A Warning Input Reference
ANWR = Add New Warning Reference	**DASR** = Display Attached Solutions Reference
RWR = Remove Warning Reference	**RTGAWASR** = Return To Get All Warnings And Solutions Reference
UWR = Update Warning Reference	

Figure 14: *Model of the WASER side part 1*

The WASER side of the admin client is again much like the SER side, where the admin mostly adds, edits or removes data. The WASER sections will not be explained into detail, since it is just like the SER side.

WASER Part 2


Warnings and solutions edit web page
Interface
DASR
Select a solution
Add new solution (solution text, priority)
Remove solution reference
Update solution (text, priority)
SAWIR
Display all solutions thats not attached
Select solution
Remove solution
Add new solution
Remove solution reference
Update solution
Add existing solution
Remove solution
Get all solutions
RTGAWASR
ANSoR
RSoRR
USoR
AESoR
RSoR
GASoR

Reference names

UWR = Update Warning Reference	**USoR** = Update Solution Reference
RSoRR=Remove Solution Reference Reference	**GASoR** = Get All Solutions Reference
ANSoR = Add New Solution Reference	**AESoR** = Add Existing Solution Reference
RSoR = Remove Solution Reference	

Figure 15: *Model of the WASER side part 2*

3.2.2 Admin Server-side

The admin side of the server is again split into a WASER references side and a SER referenced side. However they all share a common function, to check credentials, as can be seen in figure 16:

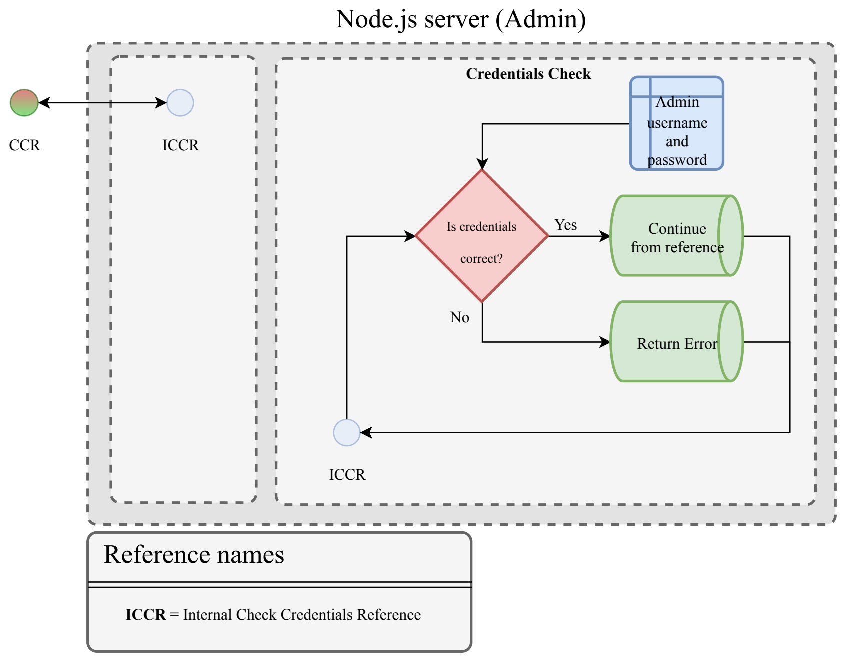


Figure 16: *Model of the admin server side*

Here the credentials for the admin is simply saved on the node.js server, and checked whenever a admin resource is references. The following models are very simple, and will not be explained in detail, however the general idea is that the credentials of the user is checked, to make sure that the user is allowed to edit the database. The green direct data boxes show what data is refereed to in to the database.

SER Server Part 1

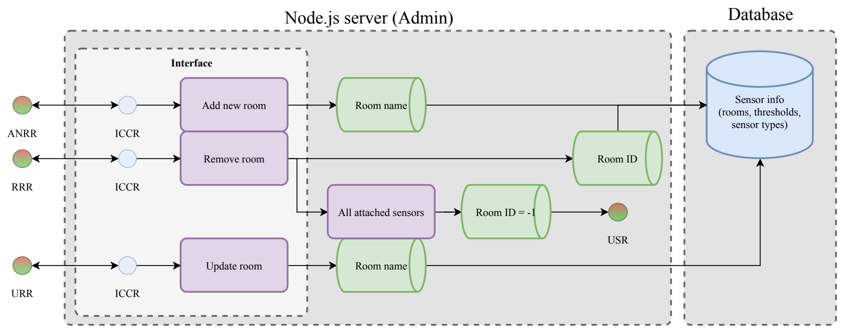


Figure 17: *Model of the SER Server part 1*

SER Server Part 2

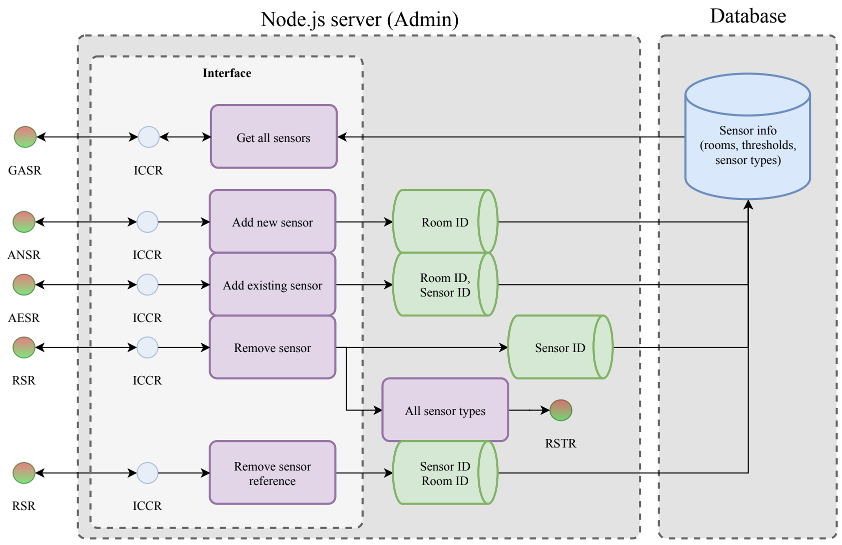


Figure 18: *Model of the SER Server part 2*

SER Server Part 3

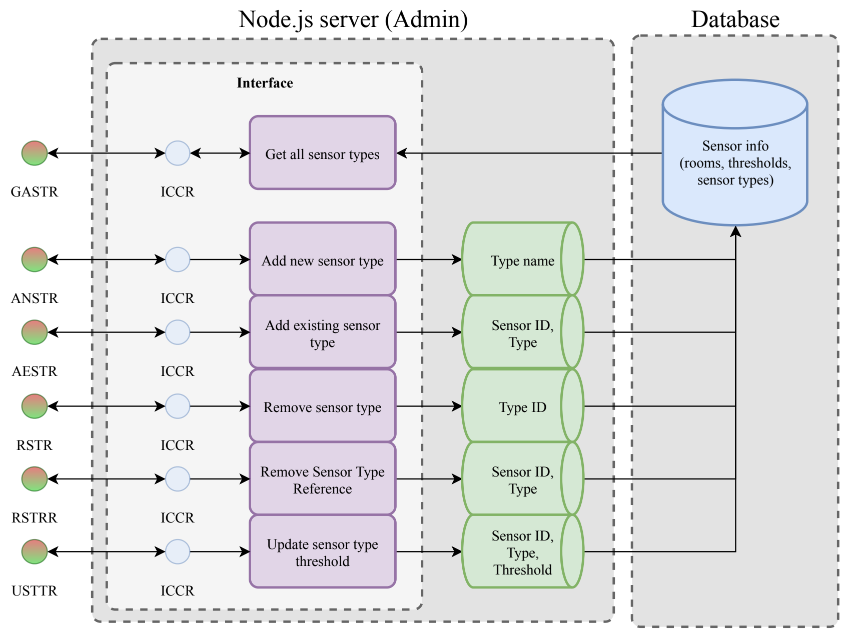


Figure 19: *Model of the SER Server part 3*

WASER Server Part 1

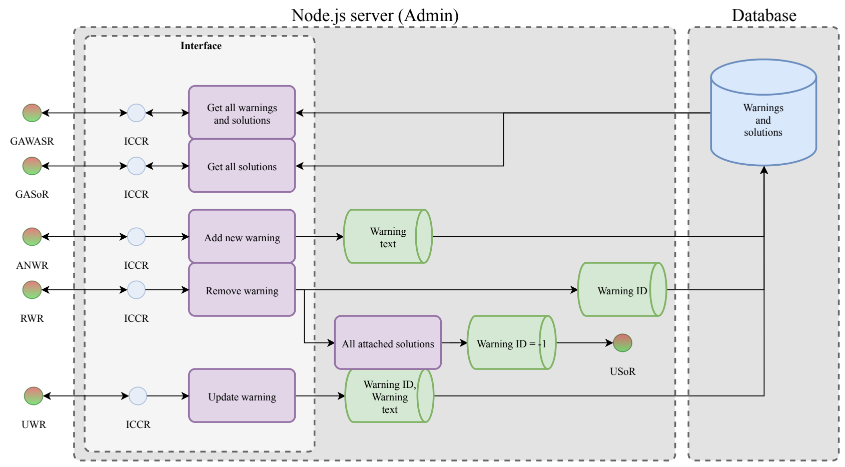


Figure 20: *Model of the WASER Server part 1*

WASER Server Part 2

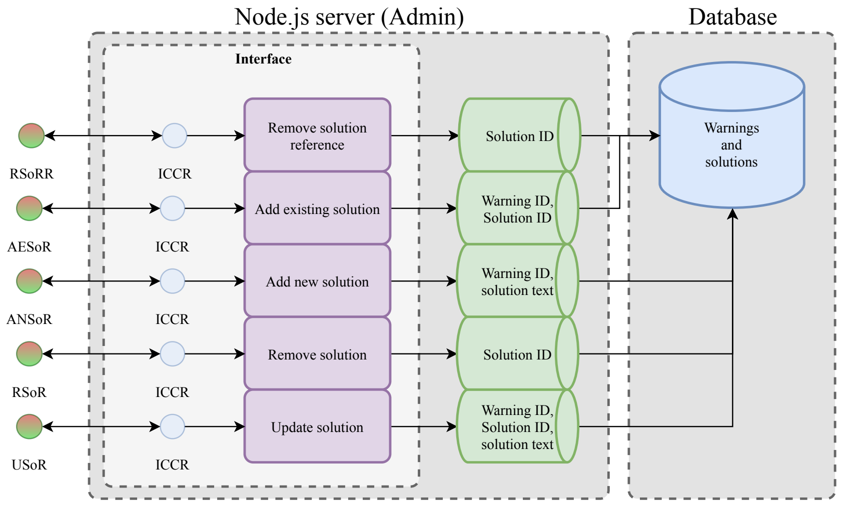


Figure 21: *Model of the WASER Server part 2*

3.3 Database

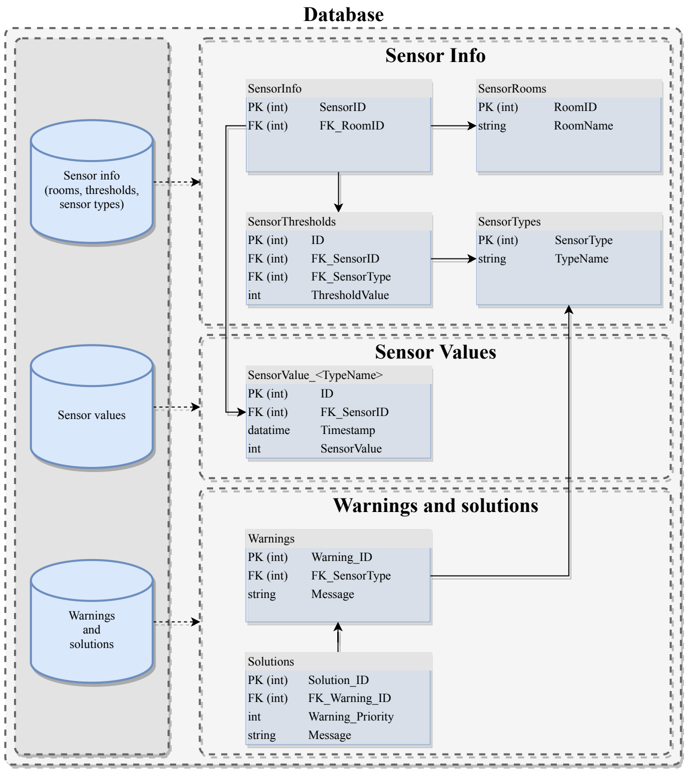


Figure 22: *Model of database*

The model for the websites database can be seen in figure 22, that consist of four different databases; "Sensor Info", "Sensor Values" and "Warnings and Solutions". These databases are here to provide the information that the node.js server needs. First off is the "Sensor Info" database.

The "Server Info" contains basic information on the sensor network, such as Rooms, sensors, sensor types and Thresholds for these sensors. At its base there is a table called "SensorInfo", this is the table of sensors that is on the network, these sensors have a primary key, SensorID and a foreign key called FK_RoomID, which informs what room it is. This is then followed by the table "SensorRooms", that consist of the RoomID referenced before, as well as the name of the room. Then there is a table called "SensorThresholds", this table consist of an ID, two foreign keys and a threshold value. The first foreign key points to a given SensorID from before, and now also a SensorType, that points to the table "SensorTypes". Such said table consist of a primary key SensorType as well as the name of the sensor, this name could as an example be "CO2" or "RH".

The next database is called "Sensor Values", it consist of the raw values that the sensors are sending. Here there are as many tables as there are sensor types, meaning that the sensor type "CO2" values gets put into the table called "SensorValues_CO2". These values all have a foreign key to a SensorID, a timestamp of when the value was put into the database, and lastly the value itself.

The last database is the "Warnings and solutions", it consist of warnings to different sensor types, as well as some attached solutions. The table "Warnings" consist of an id called Warning_ID, a foreign key to a sensor type and a message. This message could be something like "CO2 levels are too high!". The other table "Solutions" consist of a id called Solution_ID, a foreign key to a warning id, an integer defining the priority of the solution (0 = high, 1 = medium and 2 = low) and a message.

4 Implementation

4.1 Node.js Server

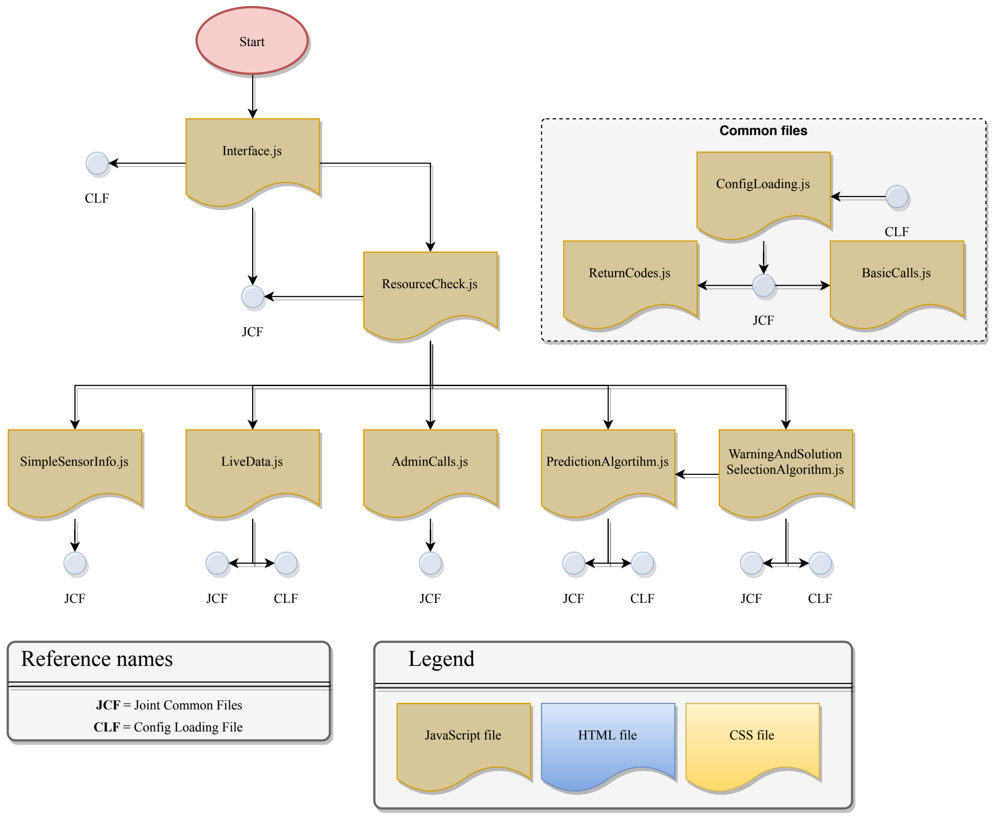


Figure 23: *File tree of the server side*

The figure 23 shows how the files in the server side are in relation to each other. It can be see that there are some reference calls, to some common files, such as ReturnCode.js, ConfigLoading.js and BasicCalls.js.

4.1.1 Return codes

All exported functions on the server side, returns objects between each other. The objects are structured so that there is a message and a return code with it. As an example, the admin section have a function called addNewWarning, this function adds a new warning to the database. If the function worked correctly, it will return the following object:

```
returnValue = new retMSG(201, "Warning added successfully!");
```

Listing 1: addNewWarning return object

The codes themselves are loosely based on HTTP codes, where the 200-299 range are success codes and the 400-499 range are error codes. The specific codes can be found in appendix[Section 8.1, Page A in appendix.].

4.1.2 Interface

4.1.3 Prediction Algorithm

4.1.4 Warnings and Solutions Algorithm

4.1.5 OpenAPI

An openAPI have also been made for the project, to make interfacing between server and client side simpler. It also makes it simpler for external programs to use the data that the program provides. It utilises openAPI 3.0.0. The API describes how one should request data from the server, as well as the structure of the return data. The openAPI can be accessed on the website of this project, by the resource "/api-doc", this will return a JSON object describing the openAPI. The API can also be viewed in SwaggerHub, where one can also interact with API (`https://app.swaggerhub.com/apis/kris701/P2Project/1.0.0-oas3#/`).

The API is split up into several sections, mainly the UserLevel and the AdminLevel. The UserLevel functions are only get functions, and requires no admin credentials to fetch. The AdminLevel is further split into four subsections, Get, Put, Delete and Path. All the admin functions requires that there is a username and password send with them.

4.2 Client-side

The client side is split into two sections, one being the main page, accessible all the time and the admin pages, only accessible by admins.

4.2.1 Main page

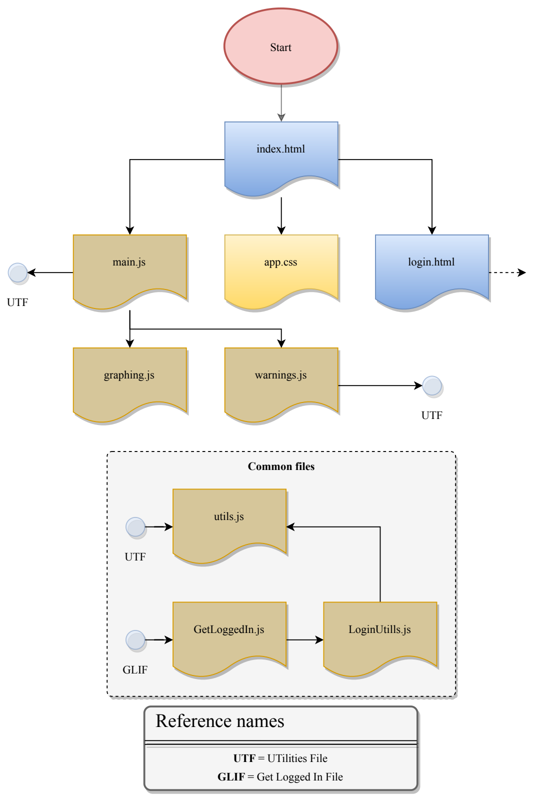


Figure 24: *Flowchart of file interaction on the main page*

On figure 24 the file interactions can be seen. In general it is the index.html holding it all together, as it is the entry point onto the website. From there a reference to a html page called login.html is made, this is refereeing to the admin pages that will be explained later down. Other then that, there are some common files, just as there where on the server side, where there this time are the files utils.js, GetLoggedIn.js and LoginUtils.js.

4.2.2 Admin pages

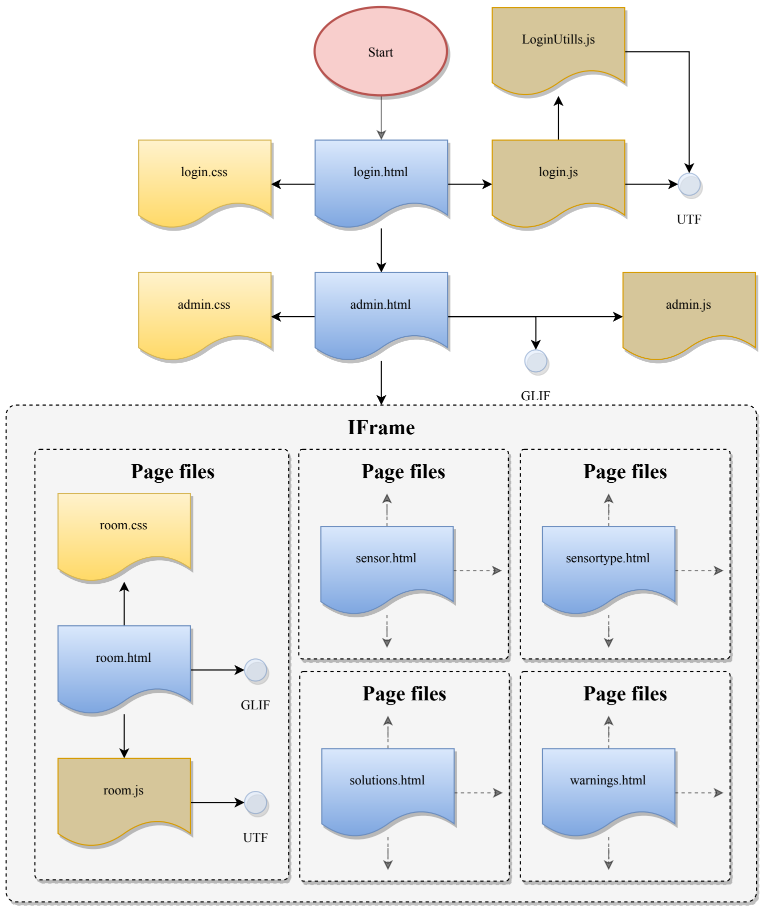


Figure 25: *Flowchart over file interaction on the admin page*

The admin pages use some of the same utilities files from the main page, however here there is an IFrame holding all the subpages of the admin page together. These pages are structured the same, with a html, css and js file for each, where the only difference is the name of the file. All these five pages can be reached from the admin.html page.

4.3 Database

The implementation of the database, is much like the model shows. The general structure is the same, and the same tables are there. The only difference there is is the addition of some more utilities tables and a table that contains solution priority:

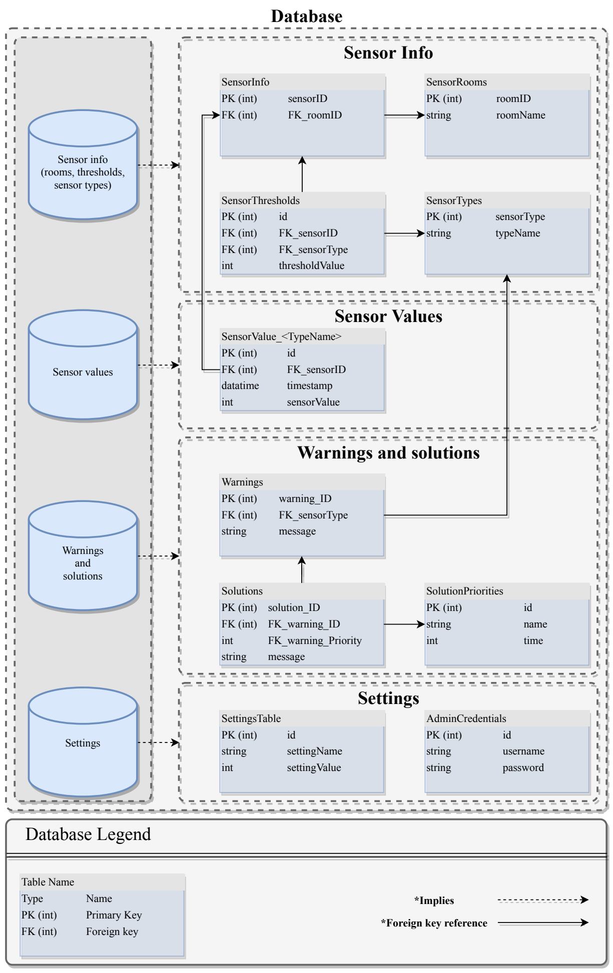


Figure 26: *Flowchart of the implementation of the database*

The solution priority table is there so that more solutions can be made, as well as edit the timefram for when they should trigger. The settingsTable consist of some static variables, that was directly in the node.js server before. However changing them required the server to restart, which is an unfavorable thing to do in web-dev. So those variables are placed in the database now, these then gets loaded, if a set amount of time is passed since the last time it loaded its settings from the database. The setting to tell how long there should go between config updated, are also placed in the database with the other settings.

The last addition table is the AdminCredentials table, that simply contain credentials for admin logins, again, to remove static variables from the server.

4.4 Program correctness

This section will discuss some of the more advanced part of the program correctness and prove that the algorithms are correct.

4.4.1 Prediction Algorithm

The mathematical version of the prediction algorithm can be seen below:

$$p(\text{Will pass again}) = \frac{\sum_{n=1}^{n} W(E(i,t,int),t)}{n} \tag{1}$$

This can be generally seen as all the times in the past x weeks that the threshold have been passed at this date. Take that number, weighted by its age, and divide it by x weeks that it looks back, the output is then a value between 0 and 1 representing percentages.

First there is a function to get the time difference between two timestamps in days:

$$T(t_1,t_2) = \text{Time difference between } t_1 \text{ and } t_2 \text{ in days} \tag{2}$$

Then there is a function to get a weighted value from the age of the timestamp. The function is a simple linear function:

$$W(t_1,t_2) = T(t_1,t_2) * a + b \tag{3}$$

Where a is -0.028 and b is 2. The reason for this a, is so that if the time difference in days are the maximum, e.g. 70 days, then the weight would end up roughly being 0:

$$70 * -0.028 + 2 = -1.96 + 2 = 0.04 \tag{4}$$

The weight 0.04 is such a small value, that it does not really have much influence. This a, b and n can be easily changed, as they are simply just values inside the database.

$$E(x, t, int) = \text{First Passed threshold at t timestamp within int minutes interval x weeks ago} \tag{5}$$

The simplest way to test if the algorithm works, is to try to run it manually, and compare the result to what comes out of the program. The core of the algorithm is the SQL query:

```sql
SELECT * FROM SensorValue_CO2 WHERE sensorID=2 AND
   TIME_TO_SEC(TIME(timestamp)) >= 39600 AND
   TIME_TO_SEC(TIME(timestamp)) <= 40140 AND
   sensorValue>=1000 AND timestamp >= "2020-02-06
   11:00:00" AND timestamp <= "2020-04-16 11:00:00"
   AND WEEKDAY(timestamp) = 3
```

This query is set up to take all the sensor values from the table SensorValue_CO2, with the sensor id 2, timestamp is to be larger than 39600 seconds and less than 40140 seconds. That interval corresponds to the time 11:00:00 until 11:15:00. Then it also takes only those entries that have a sensor value above the threshold, in this case 1000 PPM. Then it also only takes values that is newer than the 6Th of February, this corresponds to the date that its looking from, 16Th of April, subtracted by 10 weeks. It then also only gets values whose timestamp are less than the date 16Th of April. Lastly it only gets entries that is on the correct weekday, in this case 3 stands for Thursday. When entering this large query into the database, a table of five entries is outputted:

id	sensorID	timestamp	sensorValue
31138	2	2020-03-12 11:00:56	1077
31139	2	2020-03-12 11:06:06	1077
33249	2	2020-03-19 11:01:49	4715
33250	2	2020-03-19 11:06:58	4715
33251	2	2020-03-19 11:07:07	4715

The interval that is looked within is 15 minutes, so all in all this corresponds to an acctual output of:

id	sensorID	timestamp	sensorValue
31138	2	2020-03-12 11:00:56	1077
33249	2	2020-03-19 11:01:49	4715

Since the first entry within this interval, is the one that is taken. A look can now be taken on the equation from earlier:

$$p(\text{Will pass again}) = \frac{\sum_{n=1}^{n} W(E(i,t,int),t)}{n} \tag{6}$$

The values are now n = 10, t = 16Th of April at 11:00:00 and int = 15. We can then go through the sum equation, and assign the weights:

$$timestamp_1 = E(n, \text{6Th of April at 11:00:00}, 15) = \text{12Th of Marts at 11:00:56}$$

$$outWeight_1 = W(timestamp_1, \text{6Th of April at 11:00:00}) = 1.02$$

$$timestamp_2 = E(n, \text{6Th of April at 11:00:00}, 15) = \text{19Th of Marts at 11:01:49}$$

$$outWeight_2 = W(timestamp_2, \text{6Th of April at 11:00:00}) = 1.216$$

These can then be summed up, and put instead of the summation equation:

$$p(\text{Will pass again}) = \frac{1.02 + 1.216}{10} = 0.22 * 100 = 22\% \tag{7}$$

From doing this manually a predicted value of 22% is set. With the same settings on the website, a value of exactly 22% is also given, at the 15 min mark. This thereby proves that the prediction algorithm is correct.

4.5 Unit testing

It is of high importance to unit test code. Testing the code on a unit level basis, makes sure that the code written is of high quality, and simplifies trouble shooting. The method of unit testing that have been made is on an export level basis, where it is all exported functions, e.g. functions that can be called from other files, that is being unit tested.

Unit testing have been carried out on the server side, with the help of the test framework Mocha (`https://mochajs.org/`). All the unit tests use a general test class, consisting of the following test options:

```
module.exports.GTC = class {
    static shouldFailWithNoParameters(functionCall)
    static shouldFailWithnoParametersSimple(functionCall)
    static shouldReturnArray(functionCall)
    static shouldReturnArrayDotData(functionCall)
    static shouldReturnObject(functionCall)
    static shouldReturnADateObject(functionCall)
    static shouldReturnAString(functionCall)
    static outputArrayMustBeLargerThanDotData(functionCall,
       largerThan)
    static shouldFailIfTargetIDIsDefaultID(functionCall,
       defaultID)
    static expectErrorCodeFromInput(expectText, functionCall,
        returnCode)
    static expectErrorCodeFromInputSimple(expectText,
       functionCall, returnCode)
    static shouldReturnDatabaseErrorWithInput(functionCall)
    static shouldNotReturnCodeWithInput(functionCall,
       returnCode)
}
```

Listing 2: *GeneralTests.js* Unit test options

The tests are self explanatory, however those that end with ”DotData” means that the return object from a given function, it is the object property ”data” that is being evaluated. As an example the test option shouldReturnArrayDotData() in comparison to shouldReturnArray() can be looked at:

```
module.exports.GTC = class {
    ...
    static shouldReturnArray(functionCall) {
        it('Should return an array', async function () {
            let returnValue = await functionCall;
            expect(returnValue.message).to.be.an('array');
        });
    }

    static shouldReturnArrayDotData(functionCall) {
        it('Should return an array', async function () {
            let returnValue = await functionCall;
            expect(returnValue.message.data).to.be.an('array'
                );
        });
    }
    ...
}
```

Listing 3: *GeneralTests.js* DotData example

Where the ”DotData” one simply looks at the return message object

".message.data" instead of the direct object ".message". All in all there are 84 unit test for the server side, that all pass. Every exported function is unit tested, as so the calls between files and functions are correct. All the functions have at least one unit test, and usually multiple. As example on how many unit tests there can be, and how to use them can be seen in the following snippet:

```
module.exports.GTC = class {
    ...
    describe('addSolution function', function () {
        GTC.shouldFailWithNoParameters(ACC.WASC.addSolution()
            );
        GTC.shouldReturnDatabaseErrorWithInput(ACC.WASC.
            addSolution(-99, 0, ""));
        GTC.shouldNotReturnCodeWithInput(ACC.WASC.addSolution
            (0, [], ""), successCodes.AddSolution);
        GTC.shouldNotReturnCodeWithInput(ACC.WASC.addSolution
            (0, 0, []), successCodes.AddSolution);
        GTC.shouldNotReturnCodeWithInput(ACC.WASC.addSolution
            ([], 0, ""), successCodes.AddSolution);
    });
    ...
}
```

Listing 4: *GeneralTests.js* addSolution unit test

4.6 Security

The level of security of the application can be discussed. There are several factors that have been made to the program, especially the server part, to make sure that the user can not either access data that they should not, or be able to call resources that is disallowed. An overview on how requests from the client are handled can be seen here:

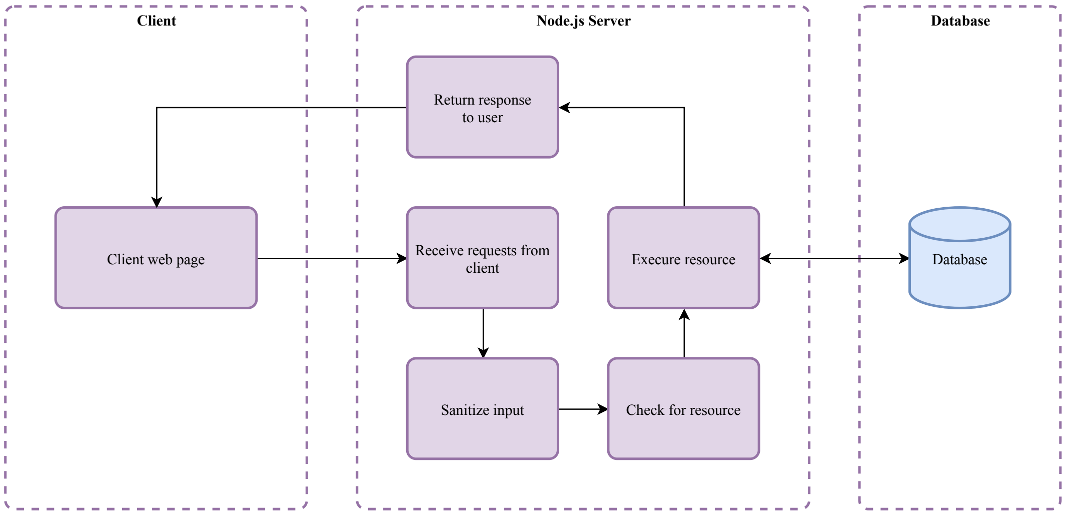


Figure 27: *Security overview*

It can be see that the user have no direct access to the database, this is to ensure that the user can not manually send queries to the database, that can potentially be harmful. In addition to that, all request inputs are sanitised, to make sure that the input is correctly formatted, and if not, returns an error. In conclusion, the security level of this project is good, owning to the properties of no direct access to the database and input sanitation.

5 Experimentation

In this chapter, larger experimentation with the program is performed, consisting of larger functionality tests, and not just unit tests. There are some static testing parameters that is used throughout the experimentation, these are descriped below:

Servers

For the testing of the program, servers where made available by Aalborg University. These servers where hosted by ITS, and the webpage can be accessed by the following URL:

`https://dat2c1-3.p2datsw.cs.aau.dk/`

The servers from ITS also included a MySQL database, that is also used by this project.

Physical sensors

Physical sensors was originally going to be used for this, however the COVID-19 lockdown closed the university. This meant that real-time sensor measurements could not be made.

Static dataset

As a backup, a dataset consisting of a lot of sensor values measured over the timespan of two months was used instead. This dataset is static, however the only thing that was done to these measurements was move the timestamp that they had up to current time, since their timestamps was from okt-nov.

5.1 Experiments

5.1.1 Prediction Algorithm accuracy

To test the accuracy of the prediction algorithm, the web interface can be used, to look at reoccurring threshold passes. If Room B is selected, firstly at the date 16-03-2020 at 05:00:00, only live data can be seen, this is because this is the oldest data there is for this room.

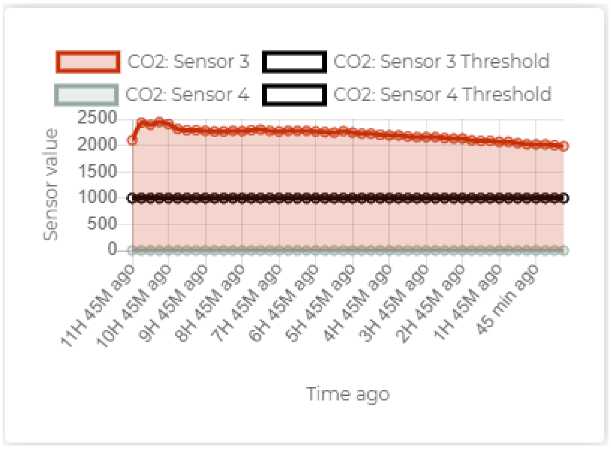


Figure 28: *16-03-2020 live data*

Here it can be seen that the sensor value for sensor 3 is way past its threshold. If the next week is then selected, at 23-03-2020 at the same time, there are now both live data and predictions. The predictions are now shown since that there are some historical data to base it on, however it is only one week, so the chance is still fairly low for it occurring again. The live data however can again be seen to have passed the threshold.

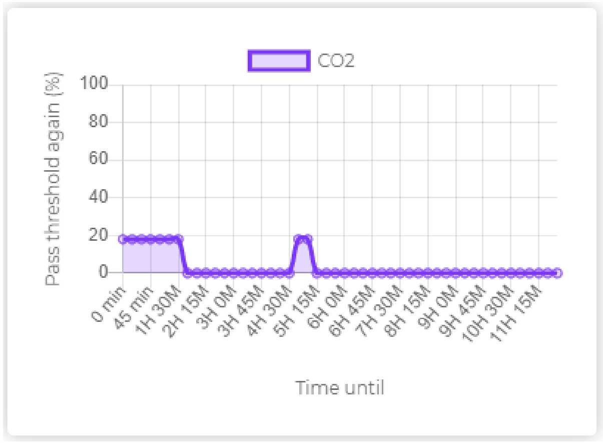


Figure 29: *23-03-2020 prediction data*

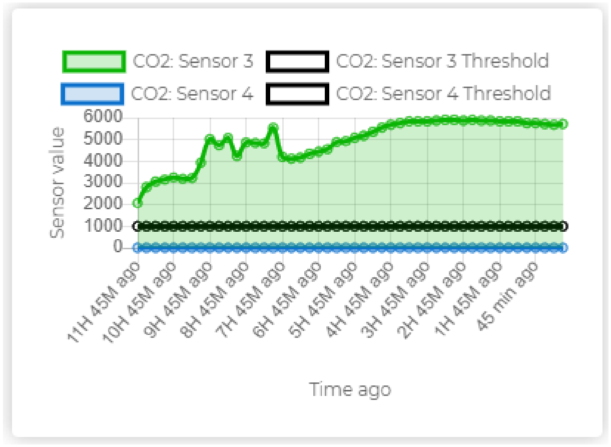


Figure 30: *23-03-2020 live data*

If the next week is then selected again, at 30-03-2020 same time, it can be seen that there is no live data. The reason for this is that there are some holes in the dataset that is being used, this does however represent a real scenario, where a sensor could break down or simply not send data for some reason. It should however be noted that the predicted chance of the threshold being passed again is higher now, since that there are now 2 times in the last two mondays that the threshold have been passed. Weighted, this corresponds to a 34% chance of it happening again:

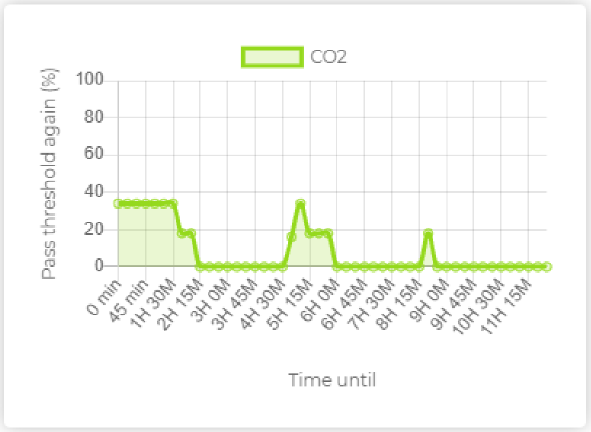


Figure 31: *30-03-2020 prediction data*

The next week at 06-04-2020 same time, the live data have again passed its threshold. It can however be seen that the predicted chance of it passing again is lowered. The reason for this is that the last week had no data, and the assumption is then made that the last mondays values where all 0.

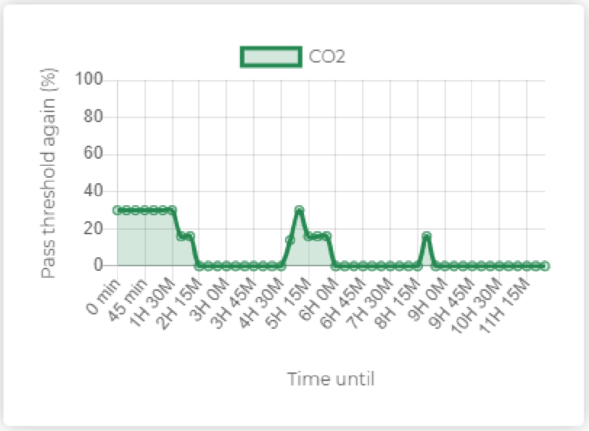


Figure 32: *06-04-2020 prediction data*

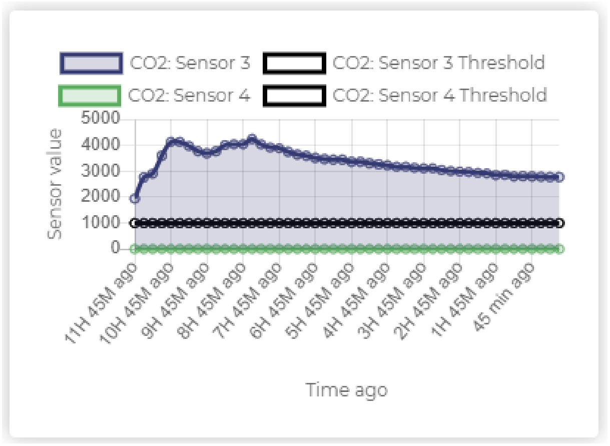


Figure 33: *06-04-2020 live data*

The next week at 13-04-2020 same time, the live data is once more passing its threshold. Now it can be seen however that the predicted chance of it happening again have increased, to a 44

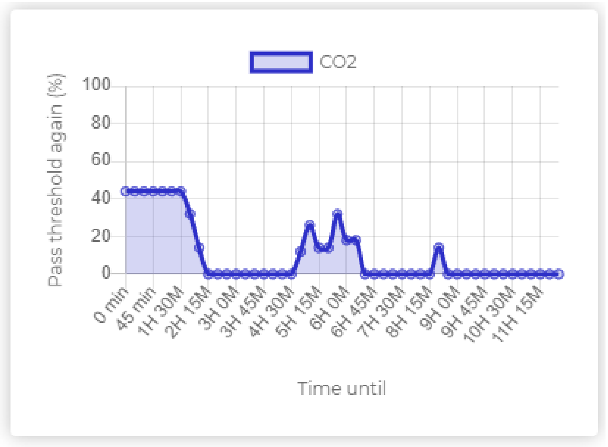


Figure 34: *13-04-2020 prediction data*

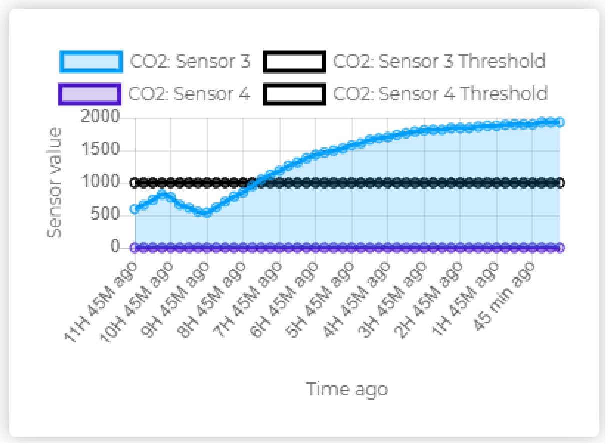


Figure 35: *13-04-2020 live data*

The next week is the last week with data, at 20-04-2020 same time, the live data is again passing its threshold, and now the predicted chance of it happening again is all the way up to 56%:

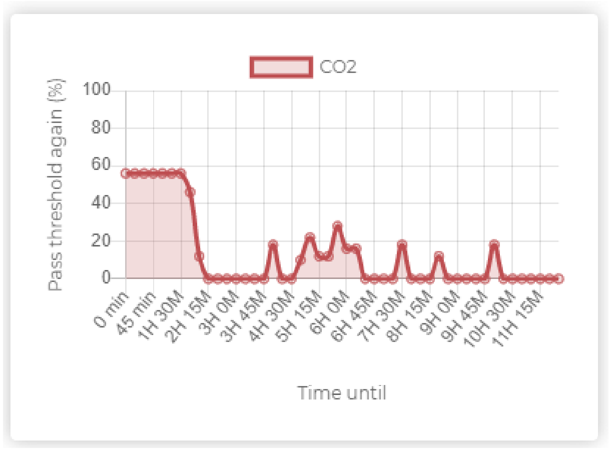


Figure 36: *20-04-2020 prediction data*

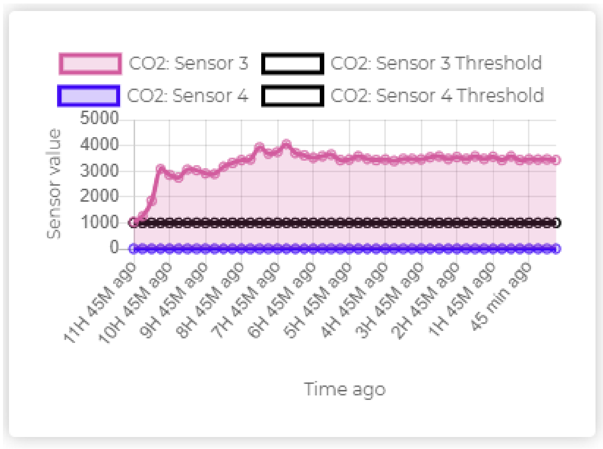


Figure 37: *20-04-2020 live data*

The end of the dataset have now been reached, however if a look is taken on the next week, at 27-04-2020 same time, there is of course no live data, however it is predicted that there is a 66% chance of it happening again. This is a fair assumption to make, since that at all the other live data that was available, the threshold was passed.

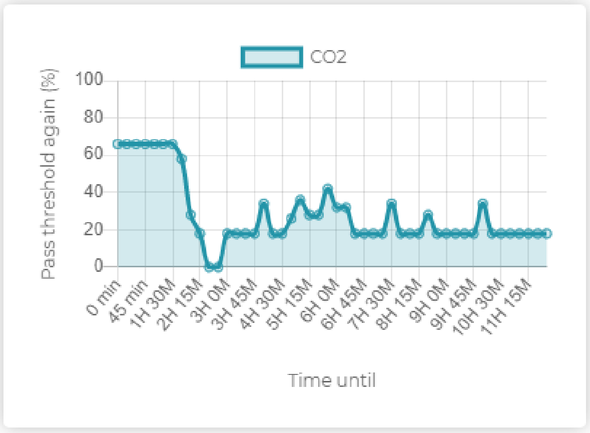


Figure 38: *27-04-2020 prediction data*

From here on out the predicted chances continue to fall the higher a week is looked into the future. This is of course because there is no more data, and that old data has a lower weight then newer data. A look can be taken at the next week again at 04-05-2020 same time, where it can now be seen that the predicted chance have fallen slightly.

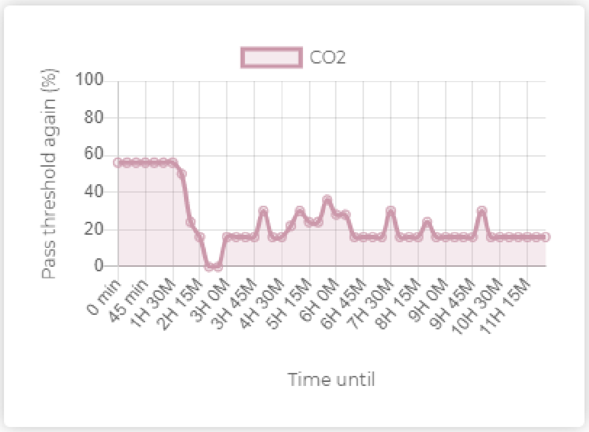
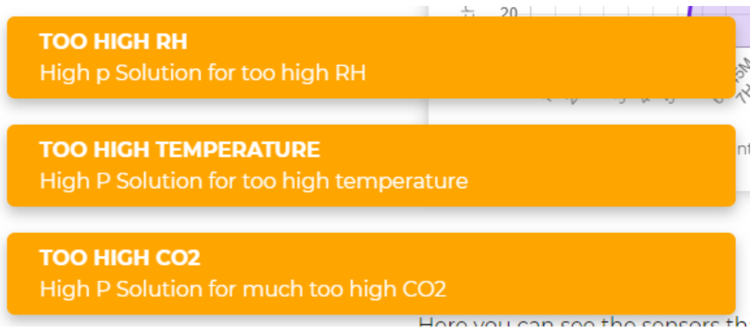


Figure 39: *04-05-2020 prediction data*

It can then be concluded that the algorithm is capable of predicting a reoccurring event, where the threshold of a sensor is passed. It can also be concluded that the larger the dataset, and preferably a ”live” dataset, would continue to increase the accuracy of the prediction algorithm.

5.1.2 Warning and Solution Display

One of the core functionalities of the program is to give a warning to the user, about up and coming bad IAQ. This functionality can be tested fairly simple, since going onto the website, there will almost no matter what be displayed some warnings and solutions. An example can be seen below:


TOO HIGH RH
High p Solution for too high RH

TOO HIGH TEMPERATURE
High P Solution for too high temperature

TOO HIGH CO2
High P Solution for much too high CO2

Figure 40: *Warnings and solutions from the 12-05-2020 at 08:57*

It can be seen that the warning priority system returns only high priority solutions. The reason for this is that the system only takes into account time, and not the predicted chance. This introduces a high level of inaccuracy to the warning system, however it should be noted that implementing a system to take the predicted chance into account as well would not be difficult to make.

For the purpose of this program however it can be tested if the priority system works. The priority times are set in the database at:

- **High Priority**: Less than 10 minutes
- **Medium Priority**: Less than 30 minutes
- **Low Priority**: Less than 60 minutes

These values can be changed anytime in the database. Next step is to find a date where a predicted sensor value reaches 0. An example can be found in Room B, at 07-05-2020 at 12:00:00, where the CO2 reaches 0:

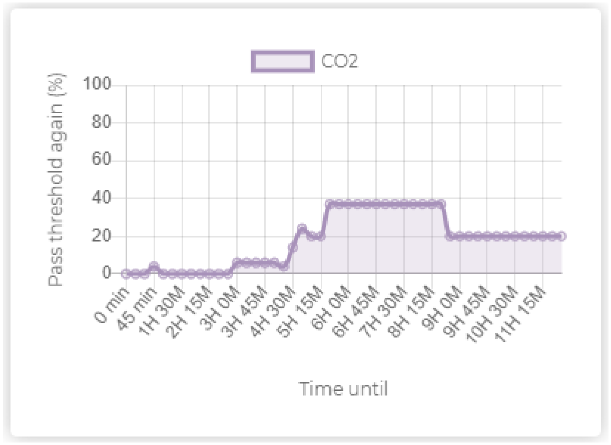
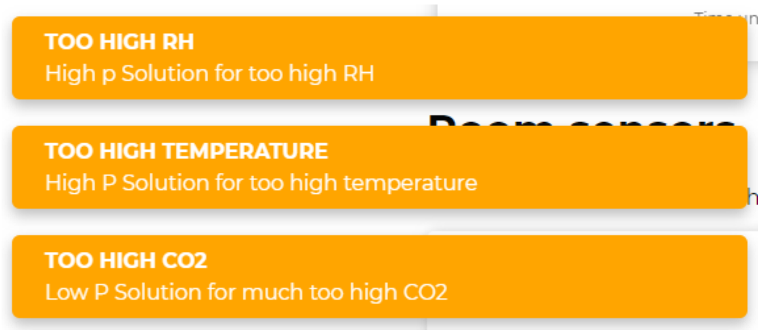


Figure 41: *The predicted chance of CO2 passing its threshold again at the date 07-05-2020 at 12:00:00*

Here it can be see that the next point, that is larger than 0, is in 45 minutes, this should trigger a low priority warning, since it is lower than 60 minutes and higher than 30 minutes. It can be seen that the low priority warning gets triggered, in the following figure:



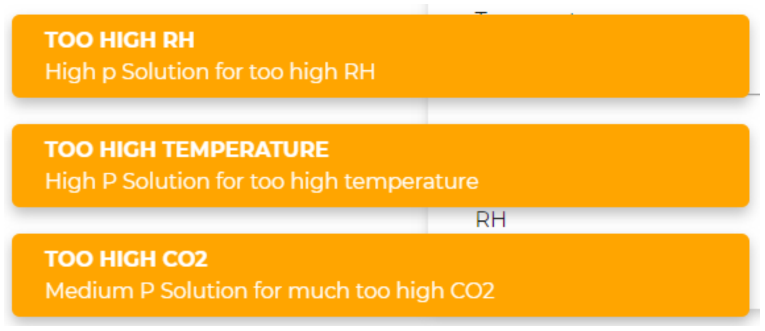
TOO HIGH RH
High p Solution for too high RH

TOO HIGH TEMPERATURE
High P Solution for too high temperature

TOO HIGH CO2
Low P Solution for much too high CO2

Figure 42: *Warnings and solution for the date 07-05-2020 at 12-00-00*

To trigger the medium priority warning, the time have to be set 30 min ahead, so that there is 15 minutes until the predicted threshold pass gets larger than 0. The result can be seen in the following figure:



TOO HIGH RH
High p Solution for too high RH

TOO HIGH TEMPERATURE
High P Solution for too high temperature

TOO HIGH CO2
Medium P Solution for much too high CO2

Figure 43: *Warnings and solution for the date 07-05-2020 at 12-30-00*

If the time was pushed forward by a further 15 min, then the high priority solution would be triggered, since there then are less than 10 minutes to the predicted threshold is passed. It can then be concluded that the warning and solution priority system works, even with its deficiencies.

6 Discussion

7 Conclusion

Bibliography

[1] MOHIEDDINE BENAMMAR et al. “Real-Time Indoor Air Quality Monitoring Through Wireless Sensor Network”. In: *International Journal of Internet of Things and Web Services* 2 (), pp. 7–13. URL: https://www.iaras.org/iaras/journals/caijitws/real-time-indoor-air-quality-monitoring-through-wireless-sensor-network. (accessed: 20.02.2020).

[2] Lia Chatzidiakou, Dejan Mumovic, and Alex James Summerfield. “What do we know about indoor air quality in school classrooms? A critical review of the literature”. In: *Intelligent Buildings International* 4.4 (2012), pp. 228–259. DOI: 10.1080/17508975.2012.725530. eprint: https://doi.org/10.1080/17508975.2012.725530. URL: https://doi.org/10.1080/17508975.2012.725530. (accessed: 14.02.2020).

[3] Digi-Key. *SEN-14348*. URL: https://www.digikey.dk/product-detail/en/sparkfun-electronics/SEN-14348/1568-1706-ND/7652735?utm_adgroup=Evaluation%20Boards%20-%20Expansion%20Boards%2C%20Daughter%20Cards&utm_source=google&utm_medium=cpc&utm_campaign=Shopping_Product_Development%20Boards%2C%20Kits%2C%20Programmers&utm_term=&productid=7652735&gclid=CjwKCAjw7-P1BRA2EiwAXoPWA6CgOpBmlqoTsra-LJWcyiKZzw0kzYmTAaqYuFFmgx2c0rzGjXoqNBoCLDIQAvD_BwE.

[4] Elfa Distrelec. *BME680*. URL: https://www.elfadistrelec.dk/da/adafruit-bme680-sensor-5v-adafruit-3660/p/30129236?channel=b2c&price_gs=195&source=googleps&ext_cid=shgooaqdkda-blcss&kw=%7Bkeyword%7D&ext_cid=shgooaqdkda-P-CSS-Shopping-MainCampaign-Other&gclid=CjwKCAjw7-P1BRA2EiwAXoPWA8fPiWI8TVwHemyt2zAgUXIVYzaUZ5PpLb8gCQ_M0xWUfijZckJngBoCic4QAvD_BwE.

[5] Let Elektronik. *CCS811*. URL: `https://let-elektronik.dk/shop/1500-biometri--gas/14193--air-quality-breakout---ccs811/?gclid=CjwKCAjw7-P1BRA2EiwAXoPWA2mV6Yqh-w-7PXdCPDubiYeTuTw1IEUZQbnXGjC7qRCV1TW8idDyPhoCVPIQAvD_BwE`.

[6] William J. Fisk et al. “Is CO2 an Indoor Pollutant? Higher Levels of CO2 May Diminish Decision Making Performance”. In: *ASHRAE JOURNAL* 55.3 (Mar. 2013). DOI: `accessed:14.02.2020`. URL: `https://www.osti.gov/servlets/purl/1171812`.

[7] Loes Geelen and A. Zijden. “Healthy learning at school!” In: *The European Journal of Public Health* 16 (Jan. 2006), pp. 44–44. URL: `https://www.researchgate.net/publication/295275585_Healthy_learning_at_school`.

[8] Jie He et al. “A high precise E-nose for daily indoor air quality monitoring in living environment”. In: *Integration* 58 (2017), pp. 286–294. URL: `https://www.sciencedirect.com/science/article/pii/S0167926016301985`. (accessed: 20.02.2020).

[9] Michele R. Kinshella et al. “Perceptions of Indoor Air Quality Associated with Ventilation System Types in Elementary Schools”. In: *Applied Occupational and Environmental Hygiene* 16.10 (2001). PMID: 11599544, pp. 952–960. DOI: `10.1080/104732201300367209`. eprint: `https://doi.org/10.1080/104732201300367209`. URL: `https://doi.org/10.1080/104732201300367209`. (accessed: 18.03.2020).

[10] S. M. Saad et al. “Indoor air quality monitoring system using wireless sensor network (WSN) with web interface”. In: *2013 International Conference on Electrical, Electronics and System Engineering (ICEESE)*. 2013. URL: `https://ieeexplore.ieee.org/document/6895043`.

[11] Usha Satish et al. “Is CO2 an Indoor Pollutant? Direct Effects of Low-to-Moderate CO2 Concentrations on Human Decision-Making Performance”. In: *Environmental health perspectives* 120.12 (2012). URL: `https://ehp.niehs.nih.gov/doi/full/10.1289/ehp.1104789`. (accessed: 17.02.2020).

[12] Wikipedia. *ESP8826*. URL: `https://en.wikipedia.org/wiki/NodeMCU`. (accessed: 25.02.2020).

[13] Wikipedia. *Query String*. URL: `https://en.wikipedia.org/wiki/Query_string`. (accessed: 26.03.2020).

8 Appendix

8.1 Return Codes

All the error codes:

Code name	**Value**	**Code name**	**Value**
NoParameters	401	TargetIsDefaultID	402
PriorityOutsideRange	403	OutputNotAnArray	404
InputNotAnArray	405	NoSensorTypes	406
IDDoesNotExist	407	DatabaseError	408
InputNotAString	409	EmptyString	410
WrongInputCredentials	411	ResourceNotFound	412

(continues on the next page)

And all the success codes:

Code name	**Value**	**Code name**	**Value**
AddWarning	201	RemoveWarning	202
UpdateWarning	203	AddSolution	204
RemoveSolution	205	UpdateSolution	206
AddExistingSolution	207	RemoveSolutionRef	208
AddRoom	209	RemoveRoom	210
UpdateRoom	211	AddSensor	212
RemoveSensor	213	AddExistingSensor	214
RemoveSensorRef	215	AddSensorType	216
AddExistingSensorType	217	RemoveSensorType	218
RemoveSensorTypeRef	219	UpdateSensorTypeThreshold	220
InsertSensorValue	221	GotWarningsAndSoluton	222
GotSimpleSensorInfo	223	GotPredictions	224
CredentialsCorrect	225	ReadOpenAPIFile	226
GotAllSensorTypes	227	GotAllSolutions	228
GotAllSensors	229	UpdateSensor	230
GotLiveData	231	GotAllSensorTypeValues	232
GotPriorityName	233	GotAllWarningsAndSolutions	234
GotAllPriorities	235		