University of West Bohemia
Faculty of Applied Sciences
Department of Computer Science and Engineering

Bachelor's thesis

Home Water Leaks Detection

Plzeň 2021 Jakub Šilhavý

ZÁPADOČESKÁ UNIVERZITA V PLZNI

Fakulta aplikovaných věd

Akademický rok: 2020/2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Jakub ŠILHAVÝ**
Osobní číslo:	**A17B0362P**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Informatika**
Téma práce:	**Sledování úniků vody v domácí instalaci**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Prostudujte možnosti detekce úniku vody v podmínkách domácího vodovodního rozvodu.
2. Navrhněte algoritmy pro detekci nejčastějších typů úniku vody z vodovodního rozvodu.
3. Zvolte vhodnou platformu a navrhněte další potřebné technické vybavení pro konstrukci detekčního zařízení.
4. Vytvořte potřebné programové vybavení detektoru. Doplňte zařízení o možnost dálkové signalizace úniku.
5. Uvažte další možné funkce zařízení (průběžný záznam odběru vody, možnost dálkového snímání dat o odběru apod.).
6. Celé zařízení podle možnosti realizujte.

Rozsah bakalářské práce:	**doporuč. 30 s. původního textu**
Rozsah grafických prací:	**dle potřeby**
Forma zpracování bakalářské práce:	**tištěná**
Jazyk zpracování:	**Angličtina**

Seznam doporučené literatury:

Dodá vedoucí bakalářské práce.

Vedoucí bakalářské práce: **Dr. Ing. Karel Dudáček**
Katedra informatiky a výpočetní techniky

Datum zadání bakalářské práce: **5. října 2020**
Termín odevzdání bakalářské práce: **6. května 2021**

Západočeská univerzita v Plzni
Fakulta aplikovaných věd
L.S.

Doc. Dr. Ing. Vlasta Radová
děkanka

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 26. října 2020

Declaration

I hereby declare that this bachelor's thesis is completely my own work and that I used only the cited sources.

Plzeň, 29th April 2021

Jakub Šilhavý

Abstract

This document addresses the issue of potential water leaks in a family house. Water leaks could represent a significant threat to any residence as they may cause considerable damage to it. As the first part of the thesis, different kinds of water leak occurrences are taken into consideration, along with possible ways of detecting them. As a suggested solution, a process of designing a device solving this issue is described in the second part of the document. Its essential functionality accounts for aspects such as detecting a water leak, preventing potential damages, remote control, and informing the user about what has happened. The last part of the thesis regards testing of the functionality along with the process of installing the device in a family house.

Abstrakt

Tento dokument se zabývá problémem možných úniků vody v rodinném domě. Úniky vody mohou představovat významné riziko, protože mohou způsobit značné škody. V první části jsou brány v úvahu různé druhy úniků vody a možné způsoby jejich detekce. Jako navrhované řešení je v druhé části dokumentu popsán proces návrhu zařízení řešícího tento problém. Jeho základní funkce zohledňuje aspekty, jako je detekce úniku vody, prevence potenciálních škod, dálkové ovládání a informování uživatele o tom, co se stalo. Poslední část se věnuje testování funkčnosti spolu s procesem instalace zařízení v rodinném domě.

Contents

1 Introduction

There are many different water-leak-related scenarios of what could unexpectedly happen in a family house. The user might have just bought a new water-consuming device, such as a dishwasher or a washing machine, and while operating, it may break down due to a variety of reasons. For instance, the machine may have already broken while shipping, or it may have not been installed properly. Regardless of what the reason is, water leaking out of the machine may not only damage walls or furniture, but it may also destroy other devices close to the crash site, such as a television or other electrical devices. Another possible example of what may happen could be a pipe hidden somewhere in a wall that suddenly breaks off because of advanced corrosion or excessive water pressure. However, these may not be the only possible causes. Such events require an immediate response to prevent the equipment of the house from getting damaged, and therefore saving money. The situation gets even more critical when the residents are away on vacation, and there is nobody at home to deal with it immediately after it happens. The *Residential End Uses of Water* study [12] revealed that the average household losses roughly 17 gallons (64.4 liters) of water a day due to indoor leaks.

In this document, I am going to go over the process of designing a device that should be always a few steps ahead of these unfortunate events. When the device discovers a leak taking place somewhere in the house, it should automatically respond to it, for example, by closing the main valve of the house, in order to avoid any potential damage. The user should immediately know what has happened as the device keeps them updated at all times. Also, the device should provide a user-friendly interface through which the user will interact with the system remotely in order to change settings, see various kinds of statistics, set up an e-mail address for regular notifications, etc.

2 Environmental analysis

First of all, it is essential to specify the environment in which the device is supposed to operate. The whole concept mainly focuses on family houses, cabins, and cottages, where the device continuously detects whether or not there is a water leak taking place. However, if appropriate installation-related adjustments are made, the device may not be limited only by these types of houses but could be used in any residence like dormitories, apartment complexes, and other such buildings.

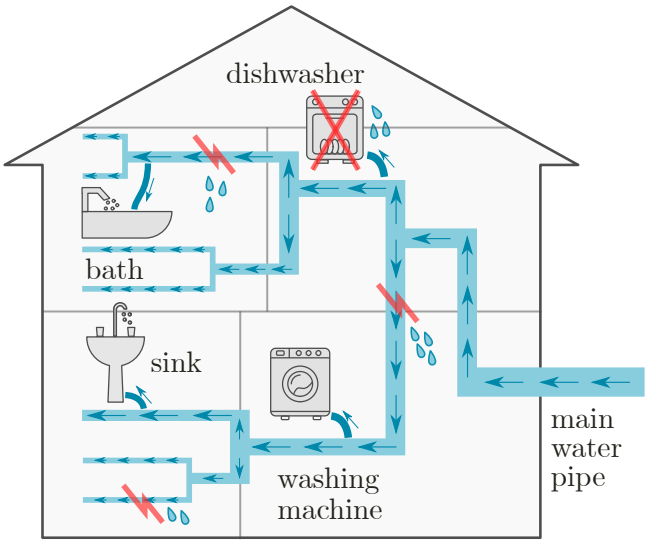


Figure 2.1: An example of crash spots in a house

The figure 2.1 shown above illustrates the basic concept of the environment. There is a main water pipe entering the house and splitting up further down into smaller pipes running through the walls of the house, onto which the user has connected various types of water-consuming machines. The red marks in the figure represent different spots where a leak may occur. These, however, may not be the only possible crash spots.

3 Water leaks classification

When it comes to theoretical scenarios of what may happen, it is crucial to realize that all of them cannot be treated in the same way, as they have distinct characteristics, and potentially, a different impact. For example, a broken dishwasher may not cause the same damage as a leaking pipe hidden in a wall; these are two different kinds of leaks. Distinguishing and defining various types of leaks is vital when it comes to their detection, as different approaches and techniques could be applied.

3.1 Low-water leak

3.1.1 Low-water leak description

As a first example, there may be loose screws holding a pipe under a sink, whether it is in the bathroom or the kitchen. Such a tube may cause a tiny constant leak that may not be easy to discover. Also, it may be quite expensive, considering it could take a couple of weeks or even months until someone would take notice of this anomaly. However, the water pipes may not be the only spot where this type of leak could occur. A broken water-consuming machine could be a source of a tiny constant leak as well. For instance, the user might have just bought an external device for watering their backyard, which is permanently connected to the water system of the house, and without even noticing, it may turn out to be leaking. There may be some other scenarios that could be thought of, resulting in the same type of leak, out of which some are more likely to happen than others.

A sign that a tiny constant leak is taking place could be that there would be no day, even when nobody is at home, with absolutely no water consumption whatsoever. How quickly such a leak is discovered would depend on the user's daily habits.

3.1.2 Low-water leak definition

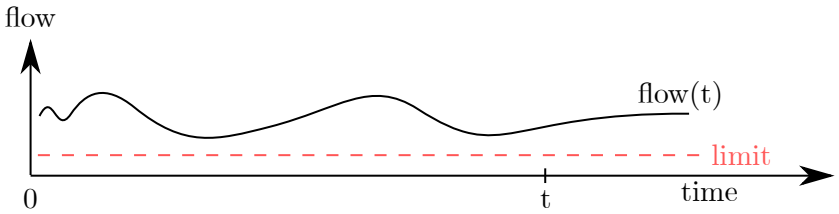


Figure 3.1: An example of a low leak occurrence

In order to find out whether there is a low-water leak taking place or not, the following formula could be used.

$$\exists\, x \in \left\langle 0,\, t-h \right\rangle : \forall\, y \in \left\langle x,\, x+h \right\rangle \; : \text{flow}(y) < \text{limit};\; x, t, h \in \mathbb{R}$$

In other words, there must be such an interval of size h between time 0 and $t - h$ in which the flow value is less than a defined limit. For example, when the residents are out of the house for a couple of days, there must be at least a few hours a day of no water consumption at all. Another example could be that there must be at least two hours every night without any water consumption.

3.2 High-water leak

3.2.1 High-water leak description

Another considered kind of leak has the character of an enormous amount of water running through the pipes of the house within a short time. Perhaps there is a water pipe hidden in a wall that suddenly breaks off due to abnormal pressure in the water system of the house, or because it was not fastened properly when being installed. Another cause could be, as mentioned above, a broken machine, such as a washing machine, out of which the water would be leaking uncontrollably. Additionally, it could all happen close to an electrical device such as a switchboard cabinet, short the circuits, and possibly set the whole house on fire. It is worth reminding that there might be nobody at home to deal with this situation immediately after it happens.

3.2.2 High-water leak definition

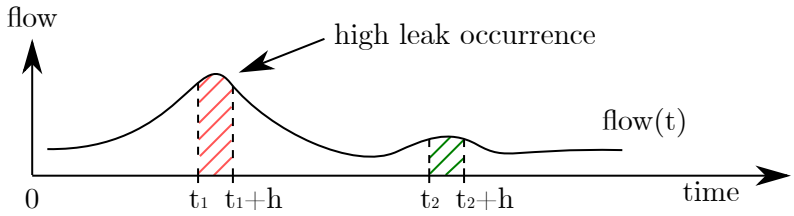


Figure 3.2: An example of a high leak occurrence

$$\int_{t}^{t+h} \text{flow}(x)\,dx < \text{limit};\ t, h \in \mathbb{R}$$

Generally, a high-water leak could be defined as a huge amount of water being consumed within a relatively short time, such as 100 liters within 5 minutes. Since every household is different, the parameters h and *limit* will be different for every type of house as well. There are several factors determining the values of these variables. For example, the number of residents living in the house, the size, whether it is permanently occupied or just a cabin, etc.

3.3 Total-water leak

3.3.1 Total-water leak description

The third and last type of leak taken into consideration, is primarily caused by the user's daily habits such as flushing the toilet, taking showers, washing dishes, and so on. It is rather a combination of the two previously mentioned leaks; even though they may not be detected separately, together they could cause a situation that could be theoretically considered as a leak.

This detection helps the user reduce the overall water consumption as it keeps track of how much water they use in terms of their daily habits. For example, the owner may say that they do not want to consume more than 500 liters of water a day. If they do, they will be notified right away as they exceed the limit that results in a total-water leak. According to the *Residential End Uses of Water* study [13], toilets represent the highest indoor use of water followed by faucets and showers. Therefore, keeping track of how often the residents use these facilities could significantly reduce the amount of money the user pays for their water bill.

3.3.2 Total-water leak definition

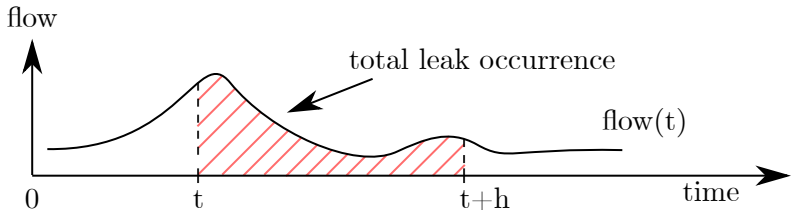


Figure 3.3: An example of a total leak occurrence

$$\int_{t}^{t+h} \text{flow}(x)\,dx < \text{limit};\ t, h \in \mathbb{R}$$

It may seem to be like this kind of leak is the same as the high-water leak (section 3.2). However, this one helps the user monitor their overall water consumption of a day or a month (this depends on the particular parameters used in the formula shown above) while the high leak detection detects abnormal water consumption within a short time.

4 Water leaks detection

Although all the leaks have been defined in chapter 3 using the flow of water running through the pipes of the house, it may not be necessarily the only possible way to detect them. In the next chapter, I will focus on different options of how to find a leak, where each of them has its pros and cons.

4.1 Using a water meter

One of the simplest ways to find out whether or not the user has a water leak in their house is to use a water meter. This process does not require any special device, and its guideline can be found at *home-water-works.org* [19].

4.1.1 How to carry it out?

As *home-water-works.org* [19] suggests, there are essentially three steps that need to be done in order to check if there is a leak taking place.

1) First, all water needs to be turned off. You might also want to make sure that no automatic water equipment is being used as it could affect the result of the leak detection.

2) As a second step, you can start recording the reading of the meter for about 15 minutes. Be sure no one is using any water during this time.

3) Lastly, start recording again. If the meter has recorded use of water during the test, it may be due to a leak.

If the user wants to check if they have a leak only once in a couple of months, for example, this solution may work great for them. However, this approach is not fully automatic, and a leak may be discovered too late, as they probably will not be doing it on a daily basis. Therefore, there is a need to find a better, more automatic way of finding a leak.

4.2 Using humidity sensors

Since a broken pipe would most likely result in a wall becoming damp, humidity sensors could be installed over some critical spots sending information to the central unit if a leak has been found.

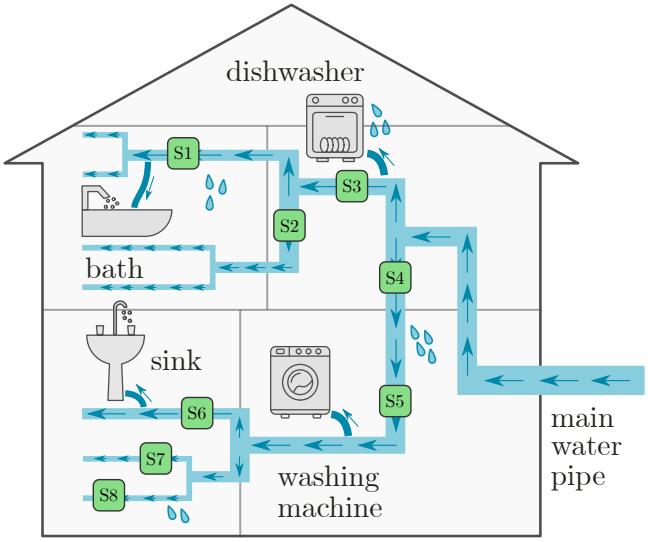


Figure 4.1: Humidity sensors installed in a house

Humidity sensors are generally quite cheap, and their use is straightforward as there are many tutorials available on the internet, which could be considered as an advantage. As well, having the humidity sensors installed all over the house would bring the advantage of locating where exactly a leak has occurred. On the other hand, the total number of sensors to cover up the entire house would need to be taken into consideration. Not to mention, it may be difficult to discover every leak if it happens in a spot where no sensor is installed (this is shown in figure 4.2 down below). As mentioned previously, every household is unique, and therefore, I believe that this solution would not be so efficient due to its inconsistency when it comes to installation.

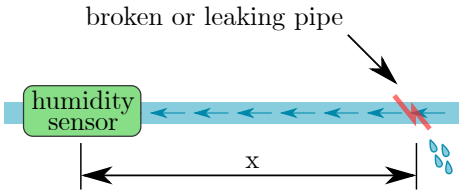


Figure 4.2: A crash site located too far away from the closest sensor

Another possible reason to rule out this option is communication between the sensors and the central unit (a microcontroller). A communication protocol, such as Bluetooth, would have to be used in case the sensors are wireless. This might be an issue when it comes to large building since Bluetooth only works within a couple of meters. It could be argued that there are some solutions for this. For instance, a network could be used. However, I still believe this solution would be too complicated and imprecise, considering the goal is to come up with a simple and effective solution.

4.3 Using flow sensors

This solution is similar to the previous one, 4.2, but instead of using humidity sensors, flow sensors would be used, which would have a couple of advantages. A humidity sensor detects a leak up to a certain distance from its location, which is a problem when it comes to installation. As shown in figure 4.2, installing a sensor in the wrong spot may lead to inaccuracy as a leak could take place out of the reach of the sensor. However, flow sensors solve this issue quite well since a single sensor can monitor an entire pipe as water flow changes equally. In other words, it should not matter where exactly the sensor would be installed as long as it is on the pipe that is meant to be checked for leaks. This solution would be more accurate, and at the same time, it would still have the benefit of spotting the exact location of a leak.

However, the drawback of this design is its price. The more accurate this solution is required to be, the more sensors would have to be installed throughout the house. Therefore, this solution would be quite expensive as far as large buildings are concerned. Also, the installation process itself would probably have to be done along with the construction process of the building.

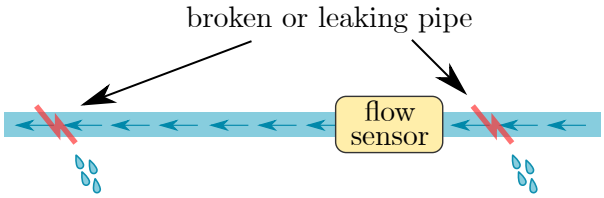


Figure 4.3: Flow sensor installation on a pipe

As figure 4.3 above shows us, unlike a humidity sensor, a flow sensor can cover up an entire pipe when it comes to finding a leak.

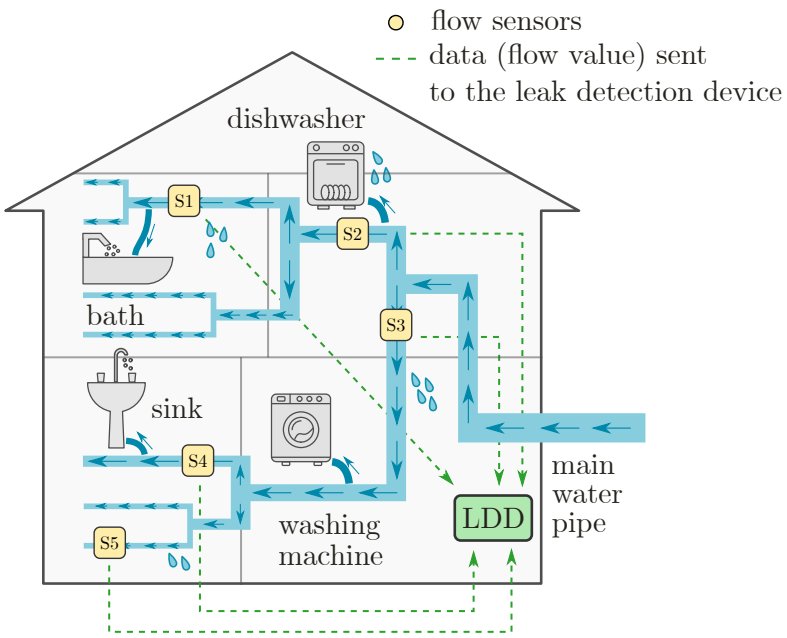


Figure 4.4: Flow sensors installation in the house

All the flow sensors send data to the central unit, which determines if there is a leak taking place. The detection algorithms used in the central unit will be discussed later in chapter 6.

4.3.1 Using one flow sensor

Another possible solution, based on the previous one, is to focus on the main pipe and use only one flow sensor. We would indeed lose the benefit of being able to say where exactly a leak has occurred, but despite that, I still believe this solution would be more efficient overall.

Not only would there be only one flow sensor to maintain, which is generally cheaper, but also, the installation process would be almost the same for every house. The process of transferring data from the sensor to the central unit, where they would be processed further, would be much easier than in the case of multiple sensors placed all over the building. How this is done will be discussed later in the document.

5 Block diagram of the system

In chapter 4, I mentioned several possible ways of leak detection. All of them have their strong advantages under the right circumstances, but for this project, I decided to go with the option discussed in section 4.3.1 (using a single flow sensor).

In my opinion, this solution could be considered the best for this project because it encapsulates all logical parts of the system into a single package that should be easy to install in any family house. It should also be easy to maintain, repair, or replace if needed. I think the simplicity and price of this solution are the compensation for not being able to point out the exact location of a leak in the house. Not to mention that this feature could be implemented later in a software way.

The whole system is composed of three main logical parts, as you can see in figure 5.1.

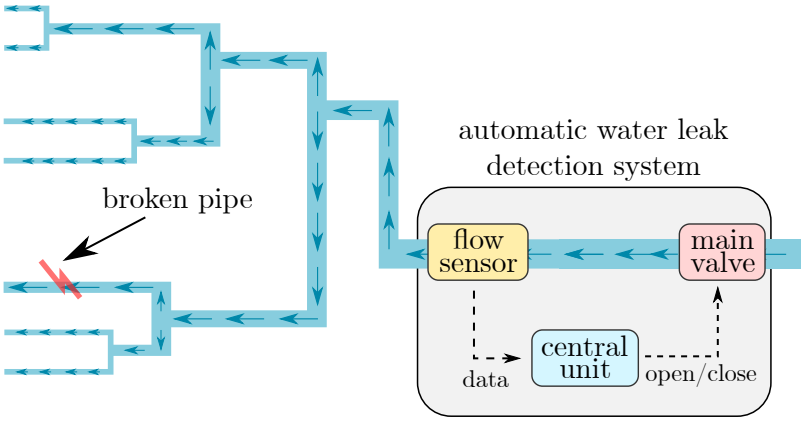


Figure 5.1: Block diagram of the system for leak detection

The input block represents the flow sensor itself which provides data about the current flow to the central unit. The central unit processes the input data in such a way that the output is in a binary format; either open or close the main valve. To achieve that, it could use the algorithms described in chapter 6. If the central unit finds out that there is a leak somewhere in the house, it will send a command to close the main valve in order to prevent the house from being flooded.

6 Algorithms for leak detection

All detection algorithms discussed in this chapter are based off of the characteristics described in chapter 3. As for the input of the algorithms, the data from the flow sensor will be used, as mentioned in section 4.3.1. Using data analysis, it will be determined if there is a leak of any kind taking place in the house which could cause serious damage.

6.1 IO of the algorithms

When it comes to an algorithm, one of the first things that need to be specified is the input and the output. In this case, as you can see in diagram 5.1, the input is represented by the flow value sent from the sensor, and the output is either yes, there is a leak taking place, or no, everything is functioning properly. Also, each algorithm takes into account several parameters associated with the particular house where the device would be installed, such as the limit triggering a high leak, maximum daily water consumption, etc.

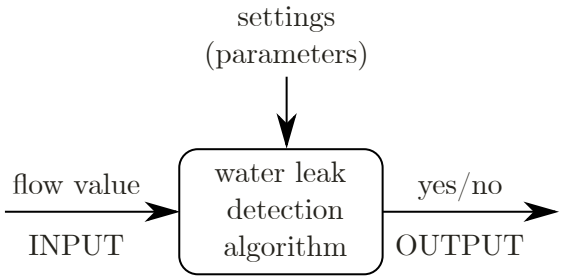


Figure 6.1: Diagram of a leak detection algorithm

6.1.1 Input data format

As mentioned previously, the input of the algorithm is given by the output of the flow sensor. This could be essentially divided into two main categories depending on the type of the sensor.

Analog input data

The first category is when the output of the sensor is given as a continuous signal. This type of signal will be transformed into a digital signal using a technique called sampling. This would be done automatically as the signal is read off a digital input pin of the microcontroller repeatedly with a certain period. *Renesas-FS2012* could be an example of this type of water meter. However its output is in volts, and as company *Renesas* suggests, it needs to be converted into SCCM [1] using the appropriate formula. [11].

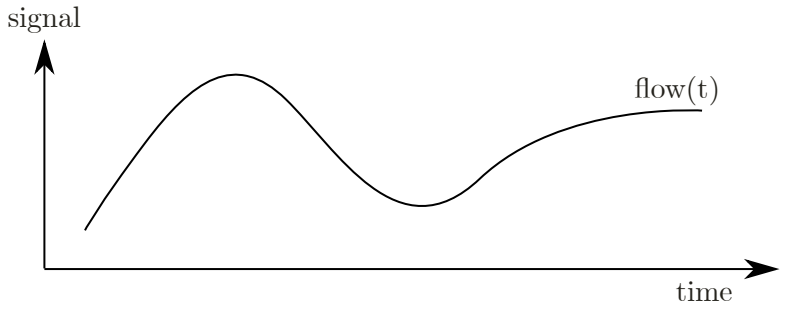


Figure 6.2: Analog signal from the flow sensor

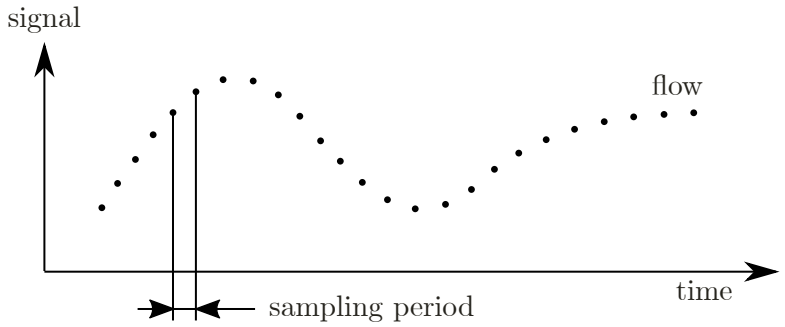


Figure 6.3: Analog signal from the flow sensor after sampling

When it comes to sampling, a sampling period needs to be specified as it plays a particularly important role in the final form of the signal. Generally, if the sampling frequency is too low, the sampled signal will not follow the original shape, and some information might get lost. For example, a leak may have occurred in between two samples, but it was impossible to detect it due to the unsuitable size of the sampling period.

Contrastingly, if the sample period is too small, the system may get overloaded because of the number of samples being stored per a constant time.

[1] standard cubic centimeters per minute

Digital input data

If the output of the sensor is in a discrete format, it means the sensor does the sampling process all by itself and provides digital data on its output. This could be done by the sensor's construction and the physical method it uses to measure flow, or it may have a sampler directly in it.

Cyble Sensor made by the *Itron* company [6] could fall into this category of water meters. It generates an electrical pulse on its output with every K liters of water flown through it. This form of data will be taken into consideration in the algorithms as well.

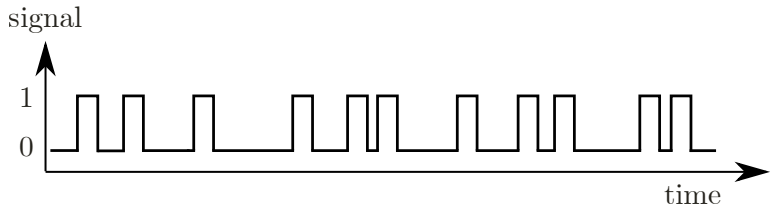


Figure 6.4: An example of output of the Cyble Sensor

6.1.2 Output data format

As for the output of all the algorithms, the main valve is required to be closed if there has been a leak detected. Therefore, the output has a simple form of a 1/0.

6.1.3 Time and space complexity

While the efficiency of the algorithms described below may not be as critical to boards such as Raspberry Pi (section 7.1), it is still essential to analyze them because of some other microcontrollers that may be less powerful.

6.2 High-water leak

6.2.1 High-water leak detection algorithm 1

Description

Supposing a high-water leak is defined as in section 3.2.2, and the data has been sampled as shown in figure 6.5, then in order to detect a high leak, we need to find an area such that with a constant width, it exceeds the given limit.

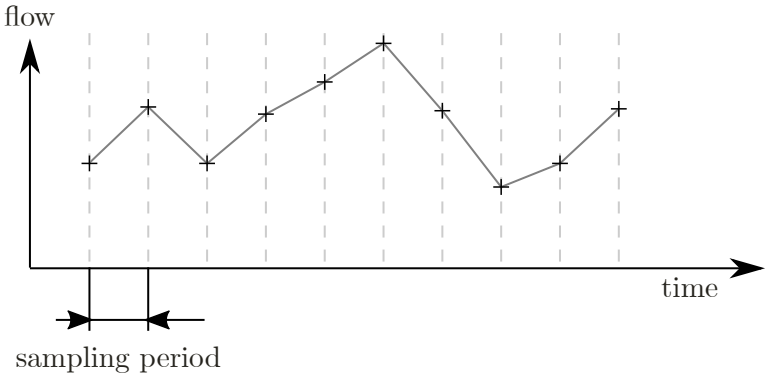


Figure 6.5: An example of a sampled signal

As the input signal is being sampled, we always look back at k samples and calculate the total area S. If the area exceeds the limit, it means that there is most likely a high-water leak taking place. In order to calculate the area, we can use approximation and create a line between every two points. Although, the real curve might be slightly different.

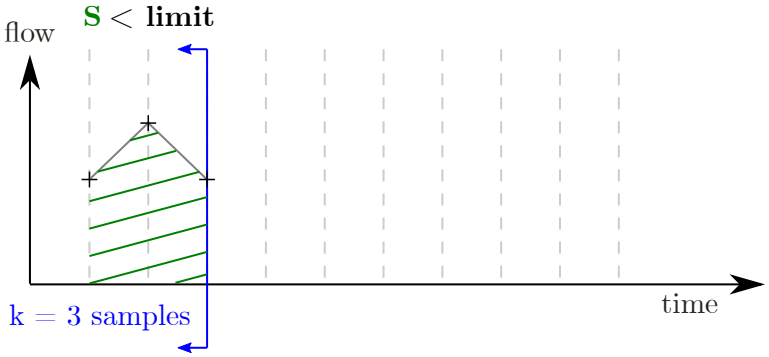


Figure 6.6: An example of inactivity of a high-water leak

For instance, the parameters of the algorithm can be adjusted, so a high-water leak is defined as 300 liters, or more, per 10 minutes. Knowing the sampling period, we can determinate the number of samples needed to calculate the area which can be then compared against the limit value. In this case, the limit is 300 liters. In case the sampling period is 1 minute, for instance, then exactly 10 samples need to be included in the process of calculating the area. However, this is just an example of what a set up could look like. All the parameters will be adjustable, so the user can change them as they need to.

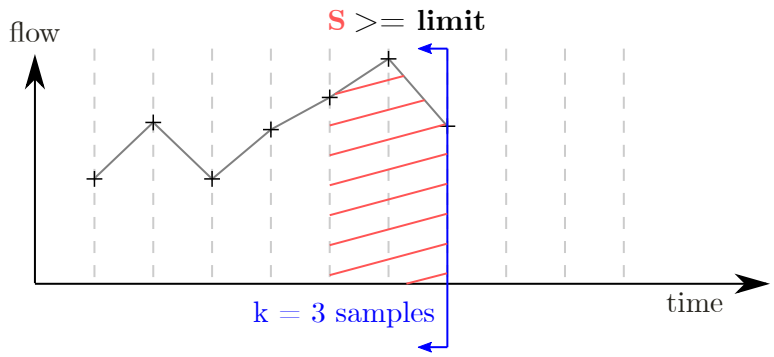


Figure 6.7: An example of a high-leak occurrence

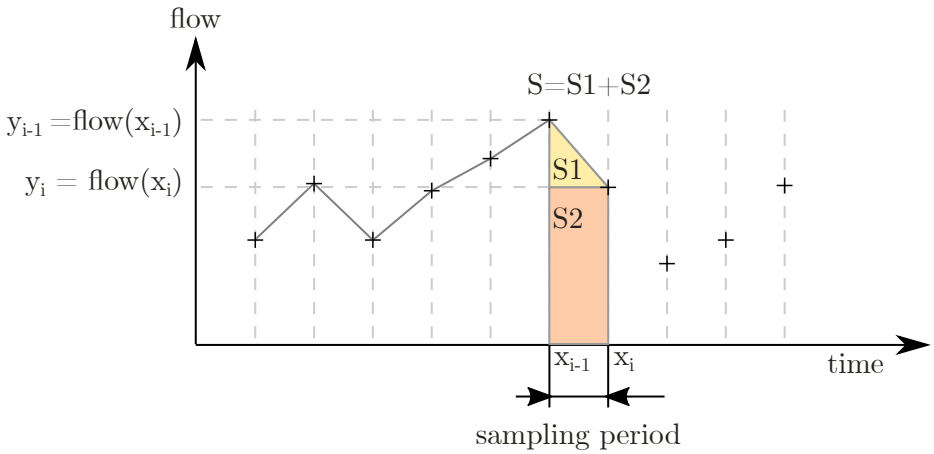


Figure 6.8: Process of calculating the area for a high-leak occurrence

Conclusion

The algorithm shown below takes four parameters in total. The first one, Y, is an array of sampled flow values (see figure 6.8) which is one of the two major values for calculating the area. The value k represents the amount of samples involved in the process of calculating the area. And finally, the last two values represent the limit and the sampling period.

The implementation of this algorithm is straightforward. However, there are a few disadvantages that need to be pointed out. The overall time complexity of the algorithm is $\mathcal{O}(k * n)$, where n is the number of samples and k, as mentioned above, is the amount of samples involved in the process of finding a high-water leak. Since k is a constant number, it can be simply ignored, so in the end, the algorithm will remain only with the time complexity being $\mathcal{O}(n)$. Nonetheless, if we consider one sample at a time, the time complexity is $\mathcal{O}(1)$ as we always look back k steps, and k is a con-

stant value. The more samples are needed, the more values also need to be stored in memory. This could be an issue when it comes to the implementation. However, from an analytic point of view, the space complexity of the algorithm is $\mathcal{O}(1)$ as the array of values has a fixed size of k.

Implementation

Algorithm 1 High-water leak detection (1)

```
1:  procedure DETECT_HIGH_LEAK(Y, k, limit, sampling_period)
2:      s = 0                                                    ▷ total area
3:      n = len(Y)                                               ▷ number of samples
4:      for i = k − 1 to n do
5:          s = 0
6:          for j = i − k + 2 to i do
7:              s_1 = 1/2 (abs(Y[j] − Y[j − 1]) ∗ sampling_period)   ▷ area s_1
8:              s_2 = min(Y[j], Y[j − 1]) ∗ sampling_period          ▷ area s_2
9:              s = s + s_1 + s_2                                ▷ update total area s
10:             if s >= limit then          ▷ test if the limit has been exceeded
11:                 return 1                                     ▷ high leak detected
12:     return 0                                                 ▷ high leak not detected
```

6.2.2 High-water leak detection algorithm 2

Description

For this algorithm, I was inspired by an algorithm for finding the maximum sum of k consecutive elements inside the array [17].

The algorithm described in section 6.2.1 can be slightly improved using a sliding-window technique. This would reduce the time complexity a little bit further as it would no longer require iterating over the array k-times with each sample. We need to keep track of the current sum, and with each sample, we need to remove the last calculated area and add the new one.

Implementation

Algorithm 2 High-water leak detection (2)

```
1: procedure DETECT_HIGH_LEAK(Y, k, limit, sampling_period)
2:     s = 0                                                  ▷ total area
3:     n = len(Y)                                             ▷ number of samples
4:     for i = 1 to k do
5:         s_1, s_2 = calculate_areas(i, i − 1)               ▷ calculate areas
6:         s = s + s_1 + s_2                                  ▷ update total area s
7:         if s >= limit then                                 ▷ test if the limit has been exceeded
8:             return 1                                       ▷ high leak detected
9:     for i = k to n do
10:        s_1, s_2 = calculate_areas(i − k + 1, i − k)       ▷ calculate areas
11:        s = s − (s_1 + s_2)                                ▷ substract the last area
12:        s_1, s_2 = calculate_areas(i, i − 1)               ▷ calculate areas
13:        s = s + s_1 + s_2                                  ▷ add the latest area
14:        if s >= limit then                                 ▷ test if the limit has been exceeded
15:            return 1                                       ▷ high leak detected
16:    return 0                                               ▷ high leak not detected
```

Conclusion

As you can see in the code above, a nested loop is no longer needed to calculate the area. This is the main reduction in terms of time complexity using a sliding-window technique. However, as all the samples still need to be kept in memory, the space complexity does not change at any rate.

6.2.3 High-water leak detection algorithm 3

Description

Another approach is to consider the input of the algorithm as shown in figure 6.4. The goal is to detect a high-water leak as visualized in figure below 6.9.

Essentially, each pulse represents a certain amount of water flown through the sensor. So in order to detect a high leak, all we need to do is to keep counting the pulses, and if a certain limit is reached, there has been a high leak detected. This value is shown as $N_{critical}$ in the figure below. Also, the counter needs to be set back down to zero if a pulse has not been detected for a certain amount of time. The period is represented as $t_{critical}$. For instance,

if there are more than 10 pulses within 5 minutes, there is most likely a high-water leak taking place in the house.

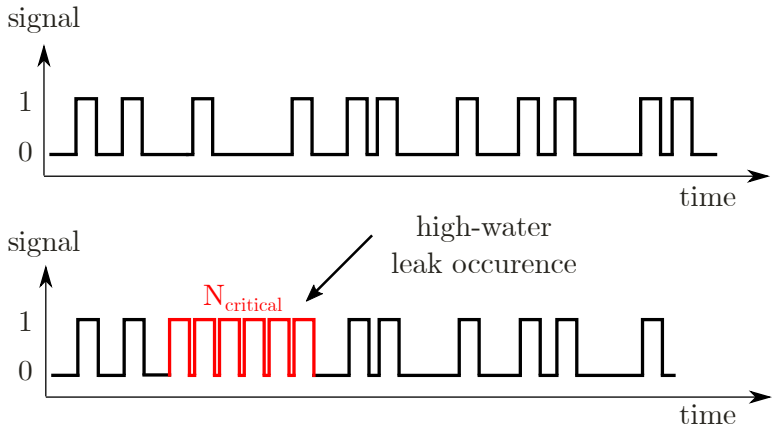


Figure 6.9: Occurrence of a high-water leak

Implementation

Algorithm 3 High-water leak detection (3)

```
1:  procedure DETECT_HIGH_LEAK(N_critical, t_critical)
2:      counter = 0                                          ▷ init counter
3:      t_1 = time()                          ▷ get current time (start of one period)
4:      t_2 = t_1                                            ▷ end of one period
5:      while 1 do
6:          if input() == 1 then                             ▷ read input value (1/0)
7:              t_1 = time()                                 ▷ a pulse has been detected
8:              counter = counter + 1                        ▷ increment counter
9:              if counter ≥ N_critical then
10:                 return 1                                 ▷ high leak detected
11:         t_2 = time()                                     ▷ get current time
12:         if t_2 − t_1 ≥ t_critical then                   ▷ no pulse detected for t_critical
13:             counter = 0                                  ▷ reset counter
14:     return 0                                             ▷ high leak not detected
```

Conclusion

What makes this algorithm more efficient is the constant number of operations that need to be done with every pulse detected on the input pin. Another considered advantage is that there are only three variables needed

to detect a high-water leak, unlike in algorithms 6.2.1 and 6.2.2, where we need a whole array of flow values. Hence, the space complexity as well as the time complexity is $\mathcal{O}(1)$.

It could be argued that the time complexity of the previous algorithms is also, analytically, $\mathcal{O}(1)$. This is true, but it is the constant size of the array that makes the time complexity $\mathcal{O}(1)$ while here, a single variable is being updated without any need of using a loop.

6.3 Low-water leak

It is assumed that a low-water leak is defined as in section 3.1. As a first step, a period during which the user wants to monitor their overall water consumption needs to be specified. For example, a day, but it could be anything satisfying the user's needs. And secondly, there must be another period specified, during which the level of water consumption should be less than a defined limit. In other words, there must be at least three hours every day without any water consumption.

Similarly to section 6.2, two different forms of input of the algorithms will be considered.

6.3.1 Low-water leak detection algorithm 1

Description

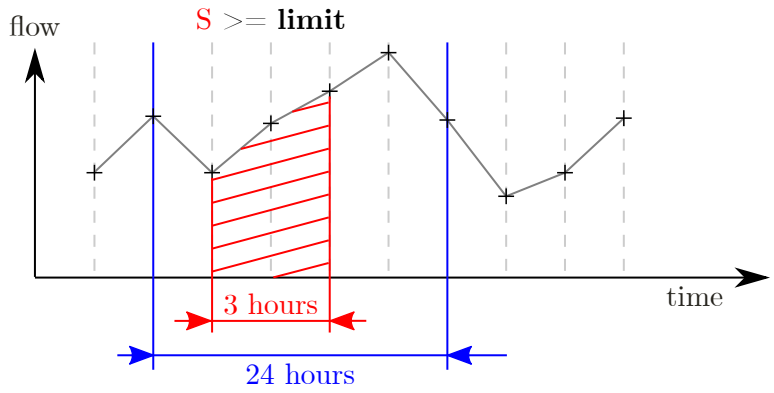


Figure 6.10: 3 hours within 24 hours exceeding the limit of a low-water leak

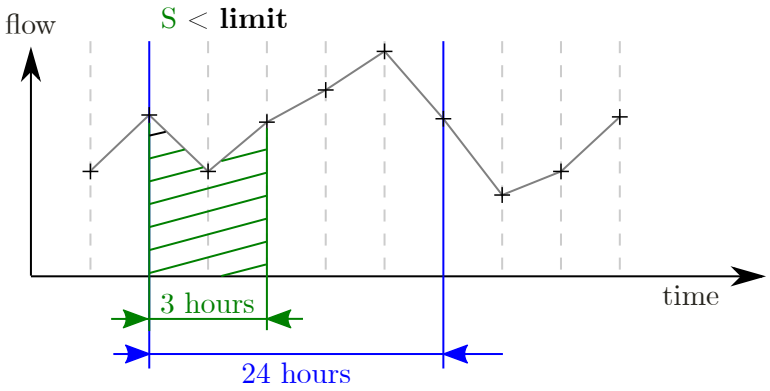


Figure 6.11: 3 hours within 24 hours classifying the whole day as low-leak free

As you can see in the figure above, this algorithm is a slight modification of the algorithm discussed in section 6.2.1. Instead of iterating back k steps with every new sample of the current flow value, the entire monitoring period is analyzed. In this case, twenty-four hours. Within the twenty-four hours, the same algorithm as in section 6.2.1 is used, and if we manage to find a window of time that does not exceed the defined limit, the whole monitoring period will be classified as low-leak free. This algorithm would be run every twenty-four hours. The consequential steps in case of a leak being detected are the same as when it comes to a high-water leak.

Conclusion

The time complexity is $\mathcal{O}(k * n)$, where n is the number of samples within the monitoring period, and k is the amount of sample making up the time window in the monitoring period. However, the same technique as in section 6.2.2 could be used to reduce it, resulting in the time complexity in a total of $\mathcal{O}(n)$. As far as space complexity is concerned, n samples need to be stored for every run of the algorithm, hence the overall space complexity is $\mathcal{O}(n)$.

6.3.2 Low-water leak detection algorithm 2

Description

For this algorithm, the input is given in a format shown in figure 6.4. After the monitoring period has been defined, e.g. twenty-four hours, the time differences between pulses need to be analyzed.

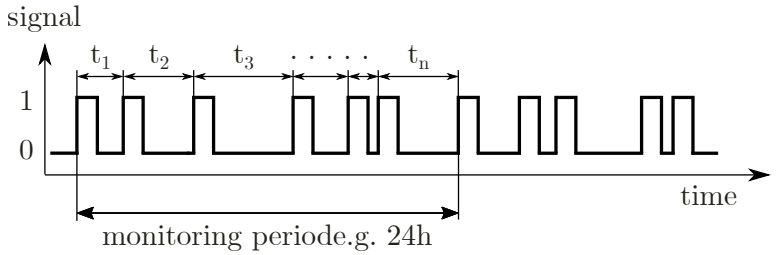


Figure 6.12: time differences between pulses in low-water leak detection

Also, the size of the time window characterizing a low-water leak needs to be specified. For instance, three hours. Once these two parameters have been defined, whether or not there is a low-leak taking place could be determined using the following formulas.

$$(\forall i \in \{1, 2, ..., n-1\} : t_{i+1} - t_i \leq limit) \Rightarrow \text{low-water leak has occurred}$$

$$(\exists i \in \{1, 2, ..., n-1\} : t_{i+1} - t_i > limit) \Rightarrow \text{monitoring period is low-leak free}$$

Implementation

Algorithm 4 Low-water leak detection (2)

```
1:  procedure DETECT_LOW_LEAK(t_reset, t_critical)
2:      set = 0                                          ▷ a flip-flop variable
3:      t_s = −1                                         ▷ flip-flop set time
4:      t_l = −1                                         ▷ time of last pulse
5:      t_c = −1                                         ▷ current time
6:      while 1 do
7:          if input() == 1 then                         ▷ there is a new pulse
8:              t_l = time()                             ▷ update time of the last pulse
9:              if set == 0 then
10:                 set = 1                              ▷ set flip-flop if it is not already set
11:                 t_s = time()                         ▷ update flip-flop set time
12:         t_c = time()                                 ▷ update current time
13:         if set == 1 ∧ t_c − t_s >= t_critical then
14:             return 1                                 ▷ low leak detected
15:         if set == 1 ∧ t_c − t_l >= t_reset then
16:             set = 0
17:     return 0                                         ▷ low leak not detected
```

The algorithm above takes two parameters. The first one, t_{reset}, is the window of time required to be without any water consumption, e.g. four hours. The second parameter, $t_{critical}$, represents the monitoring period, for example, forty-eight hours. Naturally, $t_{critical}$ is supposed to be greater than t_{reset}. The variable *set* is being set to logical one with every pulse occurrence on the input pin of the microcontroller. If there is no pulse for t_{reset}, the value of the variable will be set back to logical zero. However, if the variable has been set to logical one for more than $t_{critical}$, it is a sign that there is a low-water leak taking place.

Conclusion

The algorithm does not require any use of loops with its time complexity being $\mathcal{O}(1)$. In comparison to algorithm 6.3.1, the space complexity is $\mathcal{O}(1)$, since there are only a few variables needed to implement the whole logic.

6.4 Total-water leak

Having a similar definition, a total-water leak detection would be implemented in the way as a high-water leak detection. Therefore, instead of implementing another algorithm, we can take advantage of the algorithms described in section 6.2 and use them with different parameters. To put it another way, a total-water leak could be thought of as a high-leak stretched out over a longer period of time. For example, a high-water leak would be defined as 100 liters of water within 3 minutes, and a total-water leak, on the other hand, would be defined as 700 liters of water within 24 hours. The high-water detection algorithms are general enough to be used for this type of water leak as well, having the same advantages and disadvantages.

7 Microcontrollers

Choosing a suitable microcontroller for the project is just as important as the algorithm it is supposed to implement. In terms of functionality, several critical factors should be focused on when it comes to choosing the right one.

1) **Software support**
   I believe that supporting a variety of libraries is one of the most critical things for this project for reasons such as sending email notifications, implementing a user interface, etc.

2) **Memory**
   The more functionality is supposed to be implemented, the more memory it will take, especially RAM. Thus, it is vital to make sure the microcontroller has enough memory for all the features being implemented within this project.

3) **Security**
   This is a great issue in IoT in general, as smart devices, including this project, have access to the internet. Therefore, a responsible choice is essential in order to prevent security issues in the future.

4) **Other**
   There are many other factors that need to be taken into consideration as well. Into this category falls, for example, hardware architecture, cost, power efficiency, etc. [7].

7.1 Raspberry Pi

Raspberry Pi is a widely used single-board computer that comes with its own operating system, Raspbian, which is based on Linux. It provides a suitable amount of memory for all the features of this project, as well as their possible extensions in the future. Furthermore, it supports services such as SSH, Apache, FTP, SCP, or SAMBA. All of which can be found at *raspberrypi.org* [10]. When it comes to Raspberry Pi, we are not limited by a programming language, so we can use any programming language we want, such as `C/C++`, `Python` or `Java`. While it provides such great functionality, it could be considered more vulnerable to hackers than some ATmega-based microcontrollers, such as Arduino.

7.2 Arduino

Arduino is a very popular microcontroller finding its applications in many different fields. It comes with great library support as well as tutorials that could be found at *arduino.cc* [1]. The tutorials cover everything, from a basic LED blinking to network-related features, such as a mail-box reader or HTTP client. Arduino also comes in many flavors. For example, Arduino Nano, which is the smallest of all the boards having 2KB of SRAM along with 22 digital I/O pins, or Arduino Mega 2560, which has 8KB of SRAM and 54 digital I/O pins [2]. Also, a board can be extended by additional shields, such as the Ethernet shield, which allows the Arduino board to be connected to a network. The programming language for Arduino is based on top of `C/C++`.

7.3 STM microcontrollers

Using this brand of microcontrollers could be another possible way to implement all the features of this project. As the company claims, the 32-bit Arm microcontrollers are suitable for both small projects and end-to-end platforms [15]. They are divided into several categories in order to meet the user's requirements. For example, they can go with STM32F2 for its high performance or STM32L0 for its ultra-low power. Additionally, some external modules for Arduino can be used with this type of microcontroller as well. The company STMicroelectronics provides a variety of software tools and libraries that come in handy throughout the whole process of development. For instance, their so-called STM32Cube Ecosystem, which is a complete software solution for STM32 microcontrollers and microprocessors [16].

8 Water valves

As you can see in figure 5.1, a water valve is the last component making up the whole system. Simple functionality is required. If any kind of water leak is present, a control signal should be sent from the microcontroller to close the valve for reasons mentioned in chapter 3. The valve must be electrically controlled as a microcontroller is used to send the signal. However, other parameters, such as temperature, pressure, or cost, should be taken into account as well.

8.1 Solenoid valve

This type of valve is controlled by an electric current running through a solenoid core that changes the valve's state from open to close and vice versa. It is fast, and it could be used for either gas or liquid flowing through a pipe. The main advantage of solenoid valves is the ability to automatize changing their state by sending electromagnetic signals. It is also important to be familiar with the behavior of the valve in case the power goes out. If there has been a leak detected and the power goes out, the valve may open again, which would cause a situation as if there was no leak detection system at all.

8.2 Valve driven by an electric motor

Unlike a solenoid valve, this type of valve may not be as fast since it is driven by an electric motor. However, a couple of second latency does not make a considerable difference as far as this project is concerned. A particular example of this kind of valve could be a ball valve made by the BELIMO company. The company offers a variety of valves and actuators, so the user can purchase whatever suits them the most. For instance, they can purchase the standard actuator, which has a running time of 90 seconds, or they can decide to use the very fast running actuator, which reduces the running time down to 9 seconds [5]. In case the power goes out, the valve remains in its last position as the motor driving the valve has no power anymore.

9 IDE

Since I decided to use Arduino Mega (see chapter 7) as the microcontroller for this project, Arduino IDE seemed to be the first option that came to mind. However, I found this IDE to be too basic for maintaining a project of this size. I required synchronization with a GitHub repository, a nice-looking syntax highlighter, adding possible extensions, and other features to make the developing process easier. Other possible alternatives were CLion or Atmel Studio, both of which are widely used in the software industry.

In the end, I decided to use Visual Studio Code, shortly VSCode, for its extensions that allow the user to customize the environment to their wish. Although VSCode does not come with tools for embedded development out of the box, several extensions conveniently allow the developer to do so.

The best of which I find to be the PlatformIO [8] extension, which not only has support for Arduino development but also comes with tools for unit testing. Unit testing is an essential part of the project as it must be proven to work correctly before installation in the house. The way this is done will be discussed further in section 12.

10 SW structure of the system

Throughout the project, I use mostly object-oriented programming (OOP). I believe that with this approach, it is easier to carry out unit testing and possibly extend the already built-in functionality of the system. The Arduino language is based on top of `C/C++`, which allows creating classes. It is also a convenient way of keeping things structured and organized.

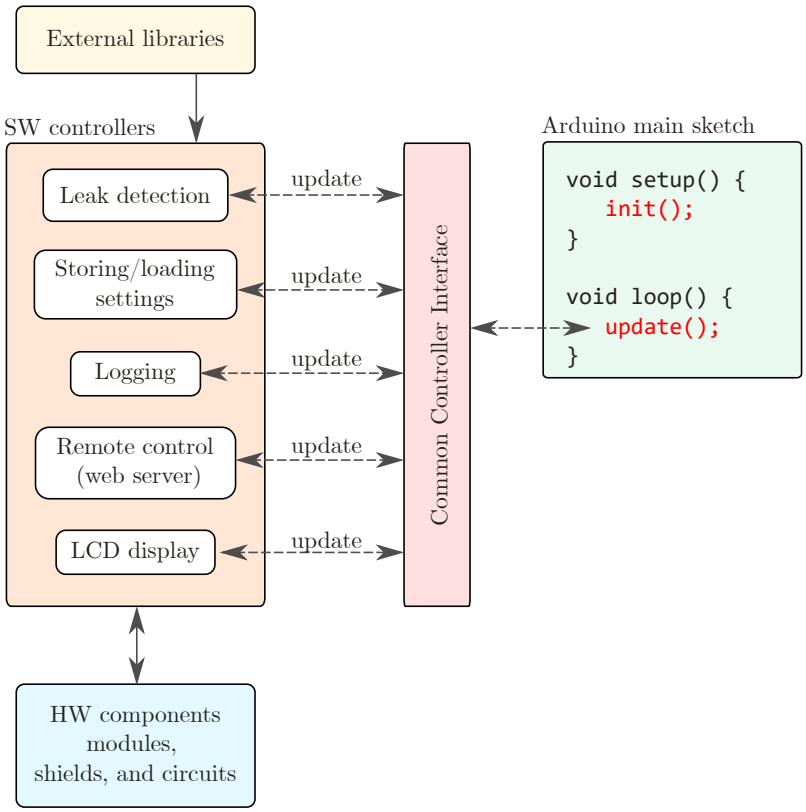


Figure 10.1: Block diagram of the SW structure of the project

As you can see in the figure above, the design has an OOP Bottom-Up structure, where the top is represented by individual SW controllers linked up together via a common interface. Each of the controllers implements a different functionality of the system. For example, the logging controller is used for debugging and troubleshooting. Some of the controllers require the use of an external library. For instance, when storing data on an SD card or when creating an instance of a web server. All the parts of the system are being periodically updated from the loop function in the main sketch. Therefore, the time complexity of the update methods plays a vital role as

the whole system should be updated, ideally, without delay. All the logical parts will be discussed in detail in the following sections.

10.1 Common Controller Interface - `IControllable`

This interface defines the common functionally for all the controllers. It is a layer of abstraction that allows the main sketch to treat all of them the same way without a need of knowing their internal implementation.

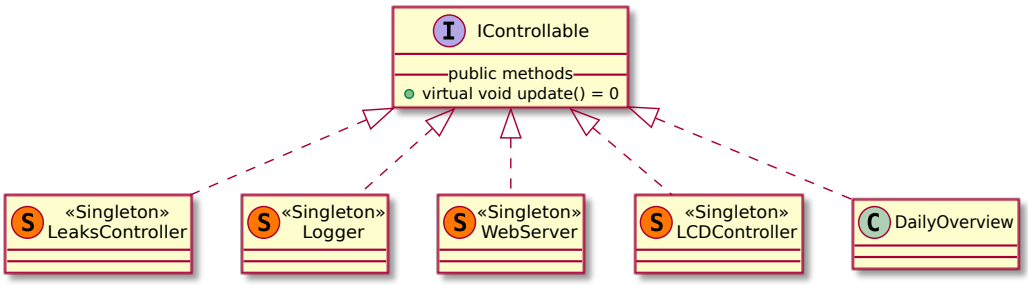


Figure 10.2: UML diagram of the interface `IControllable`[1]

Although a feature has already been implemented, it could be easily disabled if needed. For example, the `Logger` controller is primarily used for troubleshooting. Therefore, before deploying, it is usually disabled using macros after a bug has been fixed (see the User manual 14.13).

10.2 Leak detection controller

As far as the algorithms are concerned, for their entire explanation, see sections 6.2.3, 6.3.2, and 6.4. As the next step, I will explain the role each class plays in the hierarchy shown below in figure 10.3.

This part of the system takes care of water leak detection. It involves reading data from the water meter, analyzing them, and, if needed, closing the main valve.

Moreover, each of the water leak detection algorithms can be bypassed. For example, if the user expects a high-water consumption, and they do not want the device to close the main valve. For instance, when watering their backyard.

[1] All the UML diagrams were created using PlantUML

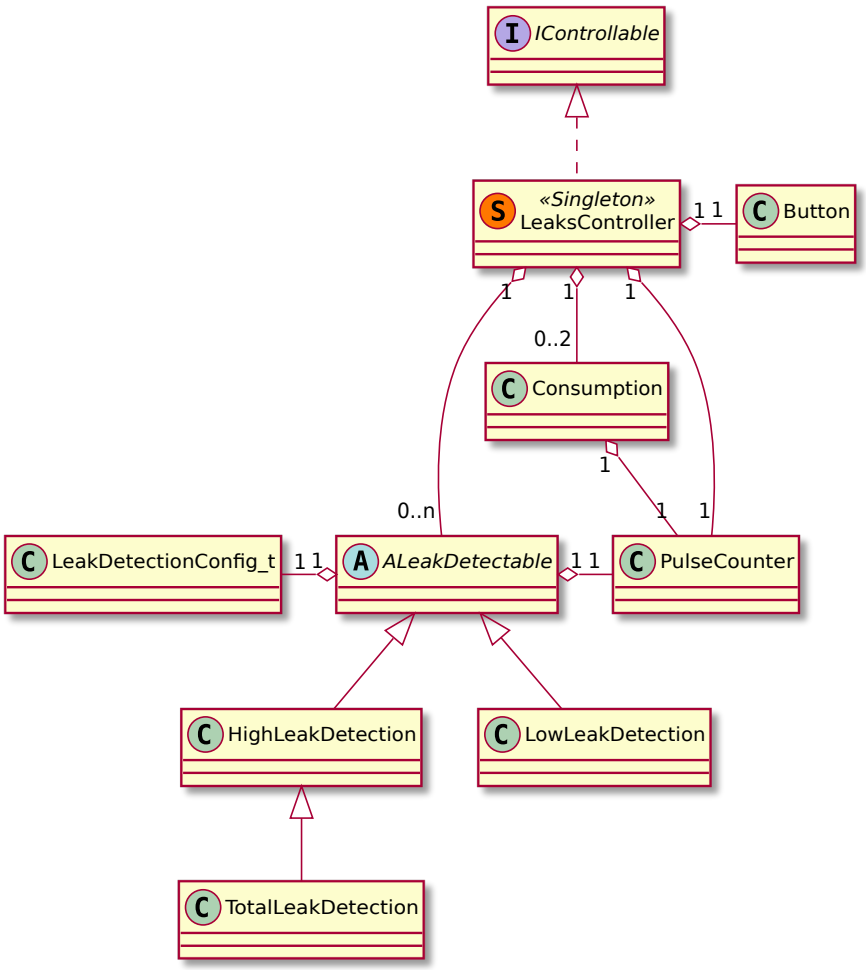


Figure 10.3: UML diagram of the leak detection system of the device

10.2.1 class `PulseCounter`

Since I decided to use the Iron Cyble sensor [6], the input of all the algorithms looks as shown in figure 6.4.

The sensor works as a simple one-button circuit, where a certain level of voltage must be provided on its input, for example 5V. When there is no flow of water in the pipe, the output is inactive (0V). After every 10 liters of water flown through the sensor, the sensor closes the circuit for a period of 70ms, letting the 5V appear on the output.

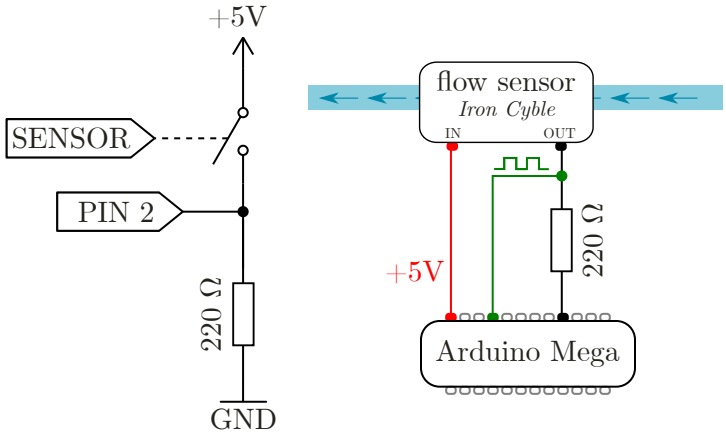


Figure 10.4: Electric circuit for reading pulses from the sensor

This class represents the source of data for all the detection algorithms. Also, it could be considered as the primary input of the whole system. The UML diagram of the class can be seen with its description in the attachments (14.1).

10.2.2 class `Button`

When a leak has been detected, the user is required to manually press the reset button as a confirmation of them having dealt with all the consequences. As soon as the button is pressed, the device will reset all the detection algorithms.

The hardware implementation is similar to `PulseCounter` (section 10.2.1). However, this time, it is not the flow sensor that closes the circuit but the user as they press the button. All the variables the class holds are described in the UML diagram in the attachments (14.2).

10.2.3 structure `LeakDetectionConfig_t`

Since every household is different, there is a requirement to have the ability to change the parameters of the water leak detection algorithms. This is where `LeakDetectionConfig_t` comes into place. It holds all the necessary variables that are mentioned in sections 6.2.3, 6.3.2, and 6.4.

However, not every variable held in this structure is used by every algorithm. It was designed in a more general way, so the structure would not have to differ for every kind of water leak detection algorithm. In particular, the low-water leak detection algorithm does not use the `limitPulseAction` variable for its functioning. The whole structure can be seen in the attachments (14.3).

It is also possible to change the parameters in real-time. This is meant to be done by the owner of the house through the interface (the web server) discussed later in the document (section 10.5).

To change the settings of a running algorithm, the new values need to be passed into the class using a particular method. For safety reasons, the new settings will not be applied immediately since a leak might be close to being detected. Instead, a flag will be set, and when the algorithm resets, the new settings will be put in place.

This feature is required in all the detection algorithm. Therefore, in order to follow the OOP approach, it is done entirely in class `ALeakDetectable`.

10.2.4 abstract class `ALeakDetectable`

This class represents an abstraction of a leak detection algorithm. It encapsulates the common features, variables, and functionality of all three algorithms, so they do not have to be implemented three times, but only once. It also implements features such as updating settings or changing them according to the state of the home alarm, which will be discussed in this section.

Settings of a leak detection algorithm

Each algorithm has its settings depending on the parameters of the house, such as size, type, etc. This is represented by structure `LeakDetectionConfig_t`, previously discussed in section 10.2.3.

Home alarm settings

When the house is empty, ideally, there should not be any water consumption whatsoever. Based on the state of the home alarm, which was used an indication of nobody being at home, the current settings could be changed to another set of values that have been adjusted for this particular situation.

It is up to the user how they want to set the parameters of the algorithms when the house is unoccupied. For instance, the limits could be twice as low in order to detect even tinier leaks. However, they should not be set entirely down to zero as there still might be some devices operating, such as a washing machine finishing up its cycle.

From the implementation point of view, the class will hold two instances of `LeakDetectionConfig_t`. One for ordinary use and the other for the situation when there is nobody at home. These are further referenced as `normalConfig` and `alarmConfig`.

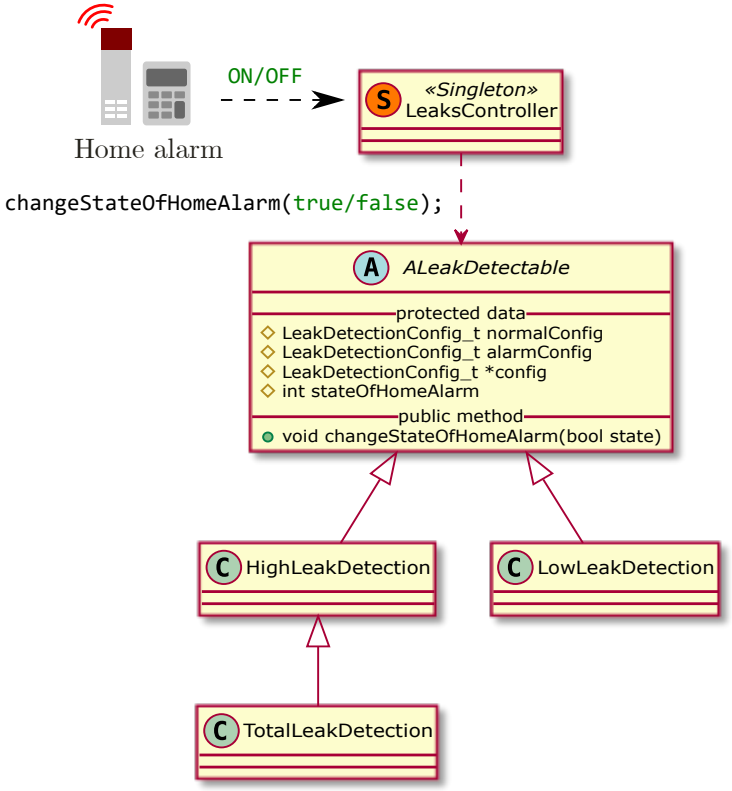


Figure 10.5: Diagram of changing settings by the state of the home alarm

The diagram above illustrates a simple principle of how changing settings by the home alarm works. When the user leaves the house, they usually set the home alarm before locking the door. The microcontroller reads the state signal in a form of 1/0 and changes the settings accordingly. As mentioned previously, the settings will not be applied immediately but with the restart of each algorithm.

Depending on the value of `stateOfHomeAlarm`, which is set up externally by `LeaksController`, the pointer `config` points either at `normalConfig` or `alarmConfig`. The UML diagram of `ALeakDetectable` shown in figure 10.5 does not show all the variables and methods contained in the class but only those used for changing settings of the water leak detection algorithms by the state of the home alarm. The entire class description could be found in the attachments (14.4).

Perhaps, this feature may not be required in every house. Therefore, it can be disabled using macros, and the `normalConfig` will be used at all times regardless of the state of the home alarm. How to disable it is shown in chapter 14.13.

10.2.5 class `HighLeakDetection`

This class mainly implements the methods regarding a high-water leak defined in its parent class, `ALeakDetectable`. In particular, the algorithm implemented within this class can be found explained in section 6.2.3.

To provide more information about the progress of the algorithm, two more methods were implemented returning percentage information about a leak being detected and the algorithm being reset.

10.2.6 class `TotalLeakDetection`

As explained in section 6.4, a total water leak could be interpreted as a high water leak with different parameters. In terms of implementation, almost all the functionality is the same as in `HighLeakDetection`. The only difference is the way this algorithm resets. Unlike the high-water leak detection, where the algorithm resets with pulse inactivity for a certain amount of time, this algorithm resets periodically every twenty-four hours. This value is set by the user through the user interface (section 10.5).

10.2.7 class `LowLeakDetection`

The `LowLeakDetection` class represents a particular implementation of the algorithm mentioned in section 6.3.2. It implements similar functionality as `HighLeakDetection` and `TotalLeakDetection`. All three water leak detection algorithms can be seen in the attachments (14.5).

Since all the water leak detection algorithms implement the same functionality, most of the methods and variables can be seen along with their descriptions in the `ALeakDetectable` class (see attachment 14.4), which is their parent class. They may only differ by their additional parameters related to the particular algorithm. These, however, can be found in sections 6.2.3, 6.3.2, and 6.4.

10.2.8 class `LeaksController`

`LeaksController` is the controller of the whole hierarchy taking care of water leak detection. It holds instances of the algorithms described above and periodically updates them. It also updates the `PulseCounter` as well as the `Consumption` class (section 10.2.9), which is used to keep track of how much water the user consumes over a certain amount of time.

The `LeaksController` class was designed as a singleton because there is no need to create multiple instances of this class within the project and thus, it is unique [20, chapter 10.5 The SINGLETON Pattern]. Most of the controllers and classes within this project are implemented similarly. The UML diagram of the class can be seen in the attachments (14.6).

Closing the main valve

This class also reads the state of the home alarm and passes it on to the algorithms to switch their settings accordingly. However, perhaps the most important feature of the class is closing the main valve. If any of the not currently bypassed algorithms detect a water leak, it will send a signal to close the main valve. The valve will then remain closed until the user presses the reset button.

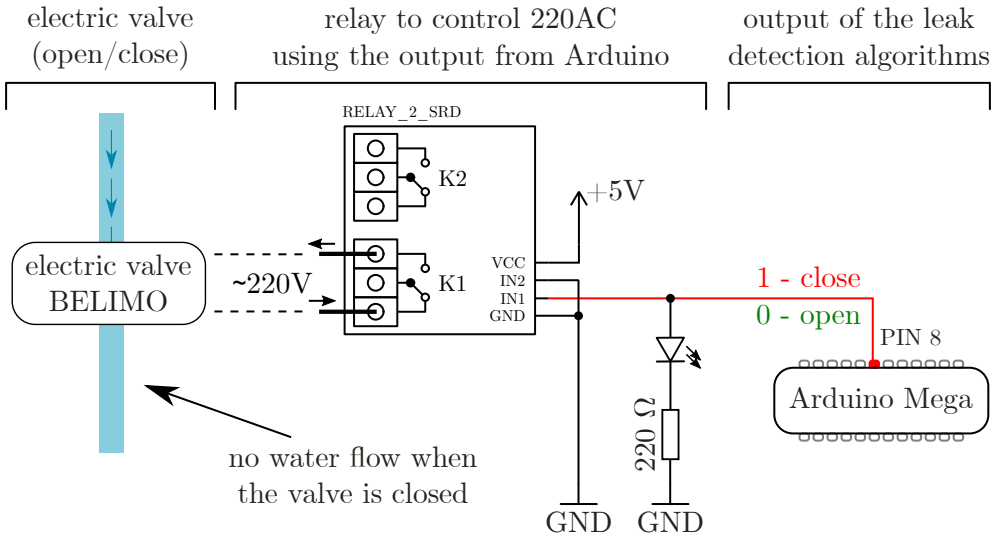


Figure 10.6: Diagram of controlling the main valve from the microcontroller

Once a signal is sent from the Arduino board, it takes a couple of seconds for the valve to change its state. As the valve needs 200AC voltage for its operating, a relay needs to be used since the microcontroller provides only 5V on its output.

Manual close of the main valve

Also, the user has the option to open or close the main valve manually by pressing one button if they need to (see the User manual 14.13). This feature might be a convenient way of turning water off for the entire house.

10.2.9 class `Consumption`

The class `Consumption` is used to monitor how much water has been consumed within a day, week, or month. It uses the `PulseCounter` class to keep counting the pulses generated by the flow sensor. The counter will be reset back to 0 after the amount of time set through the constructor when creating an instance has passed. The content of the class is described in the attachments (14.7).

10.3 Logging controller

Logging was used throughout the process of development of this project. It is used for visualization, validation, and troubleshooting of various kinds of functionality. For instance, it may be used for printing out the current state of the algorithms and their progress as the pulses are being read off the input pin. In order to follow the OOP approach, a hierarchy regarding the logging process was implemented in the project (figure 10.7).

If a class is supposed to be included in the logging process, it needs to implement two methods as it is shown in the UML diagram below. The first method, `getLogID()`, returns an identification of the class. For instance, `"HIHG_LEAK"`. The other method, `getLogDescription()`, is the content to be printed out on the screen. The log itself has then the following structure:

```
[d:h:m:s:ms][LOG_ID][LOG_DESCRIPTION]
```

Or in particular, it may look like:

```
[0:00:00:05:618][RAM][FREE=5212B]
```

As you can see above, each log being printed out on the screen is consist of a time stamp, its ID, and the content of the log itself. The time stamp is the up-time of the Arduino board.

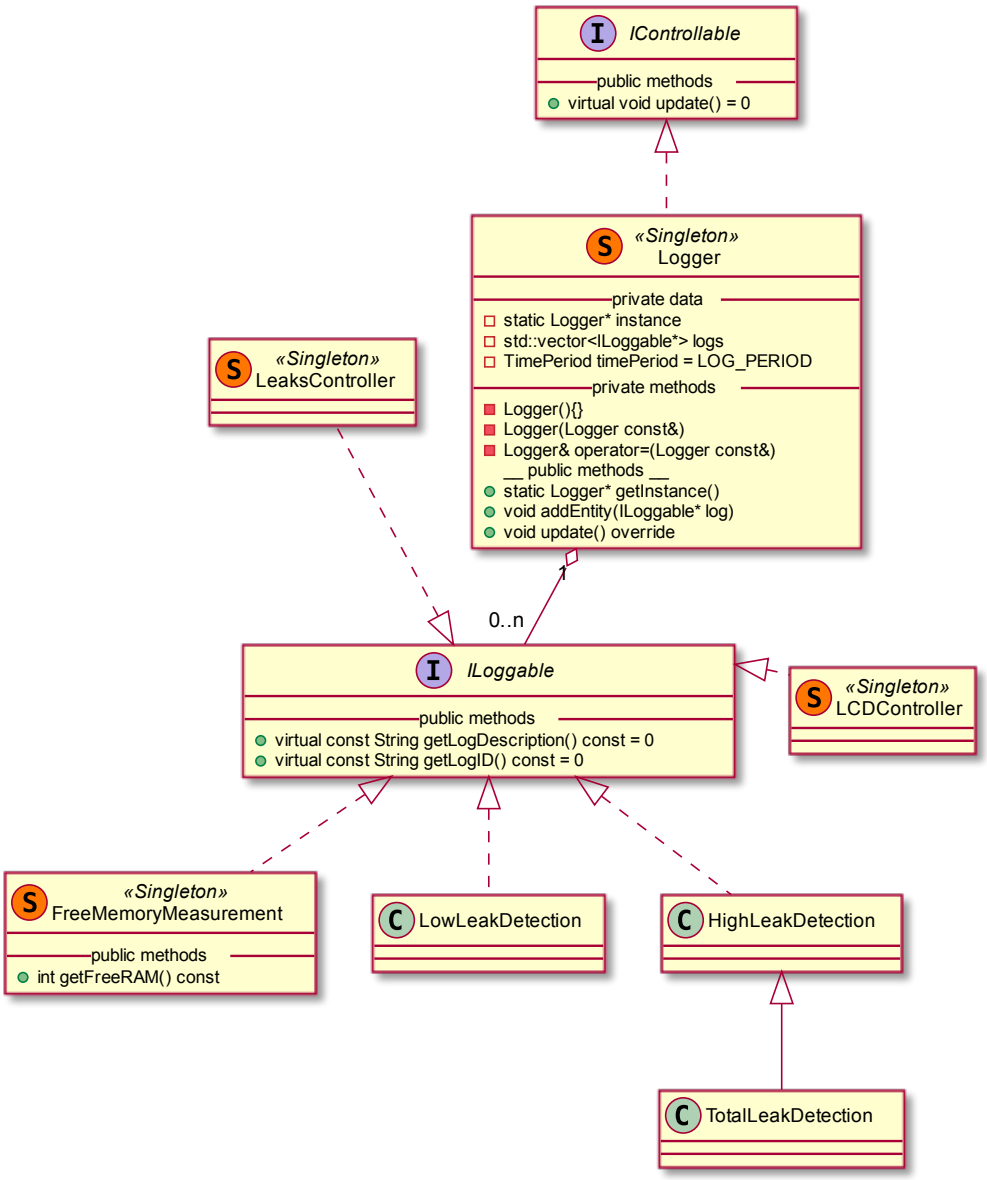


Figure 10.7: UML diagram of the logging system of the project

The UML diagram does not contain all the methods of all the classes but only those supposed to be mentioned as a part of the logging system. Each class implementing the `ILoggable` interface must explicitly implement both methods defined by this interface.

The process of logging (printing out information to the terminal) works periodically. It means that at the beginning, a logging period is specified, e.g. 3 seconds. Then every 3 seconds, the latest information about the classes registered to the logging system will be printed out.

This structure provides an easy way of both adding and removing classes

from the logging system. Also, the user can entirely disable this feature of the system or change the logging period. More information about how it is done could be found in the User manual 14.13.

Listing 1: An example of a log being printed out on the screen

```
[0:00:00:04:089][HIGH LEAK][BYPASS=0 | ACTIVE=0 | ACTIVE=3.33%
↪  | RESET=4.69% | CONFIG=NORMAL | PULSE_COUNT=1]
[0:00:00:04:140][LOW LEAK][BYPASS=0 | ACTIVE=0 | ACTIVE=1.37%
↪  | RESET=4.55% | FLIP_FLOP=1 |
↪  FLIP_FLOP_SET_TIME=0:00:00:03:621 | CONFIG=NORMAL]
[0:00:00:04:290][TOTAL LEAK][BYPASS=0 | ACTIVE=0 |
↪  ACTIVE=2.50% | RESET=0.23% | CONFIG=NORMAL |
↪  PULSE_COUNT=1]
[0:00:00:04:406][LEAKS_CONTROLLER][VALVE_STATE=0 | HOME
↪  ALARM=0 | MANUAL_CLOSE=0]
[0:00:00:04:493][RAM][FREE=5322B]
```

Additionally, there is another piece of information printed out on the screen. However, this type is not being printed out periodically but only when an event occurs. In particular, this regards the boot-up process of the system or when the system receives an HTTP request (see section 10.5).

Listing 2: Information regarding the boot-up process of the system

```
pins configuration...
loading settings...
adding instances to logging...
adding instances of all the controllers...
web server initialization...
server is at 10.10.2.118
SUCCESS - SD card initialized.
INDEX~1.HTM exists.
program is now running...
```

10.3.1 class `Logger`

This class is the main SW controller of the logging part of the system. It implements the same design pattern, singleton, as `LeaksController`. Thus, almost all the methods have the same description as in section 10.2.8.
All instances registered in the logging system are held in a `vector` data structure, which was added as an external library to the project [4].

10.3.2 class `FreeMemoryMeasurement`

As the SW structure of the project grows, it is useful to keep track of how much of the SRAM memory is still free. Arduino Mega comes with 8KB of SRAM, but this value reduces with each additional library added to the project. Therefore, it comes in handy to have this value printed out as a part of the logging system. It is also implemented as a singleton, and besides the compulsory methods, it contains one method that returns the current value of the free SRAM memory. The compulsory methods are indeed the pattern-related methods and the interface this class implements.

10.4 LCD controller

There are essentially two ways of displaying information about the system to the user. If the user, for their own reasons, does not want to have the web server functionality enabled in the system, they will still be able to see what is going on via the LCD display.
The hierarchy of this part of the system is similar to the logging system 10.3, If a class is meant to print some information on the LCD display, it needs to implement the `IDisplayable` interface.

10.4.1 intereface `IDisplayable`

As you can see in the UML diagram shown below (figure 10.9), the interface defines only one method which is used to return a particular row of the content the class is supposed to print out on the LCD display. The number of rows depends on the size of the display. In this case, a 20x4 LCD [18] display was used, so each class can print out content up to four rows, where each row is made of twenty columns.

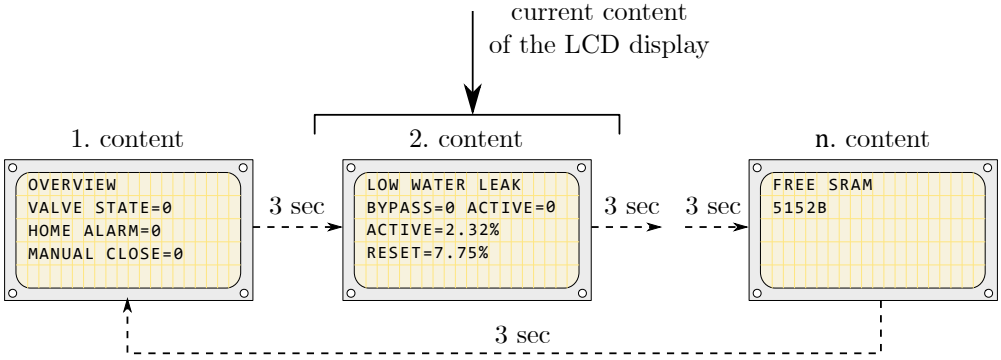


Figure 10.8: A principle of displaying multiple pages on the display

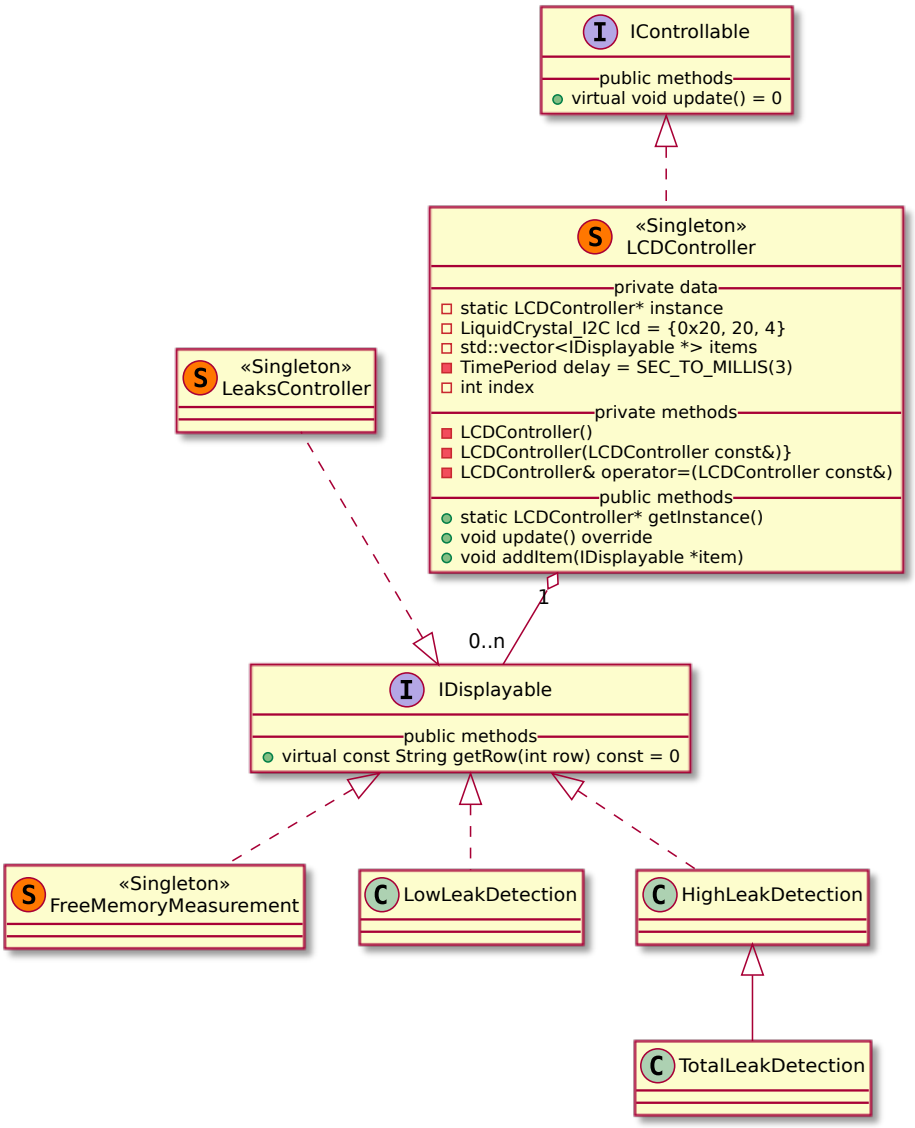


Figure 10.9: UML diagram of the hierarchy used for printing information on the LCD display

10.4.2 class `LCDController`

The `LCDController` class deals with printing information out on the LCD display. It uses a paging system where each class represents a single page consist of four rows, as explained in section 10.4.1. There is also a delay of three seconds between two pages, so the user can read the status from the display effortlessly. Its principle is shown in figure 10.8.
This class, as any other SW controller within the project, is implemented as a singleton.

10.5 Web server

The web server represents the main interface the user uses for interaction with the system. Not only does it provide various kinds of information about the system, but it also allows them to change the parameters of the water leak detection algorithms.

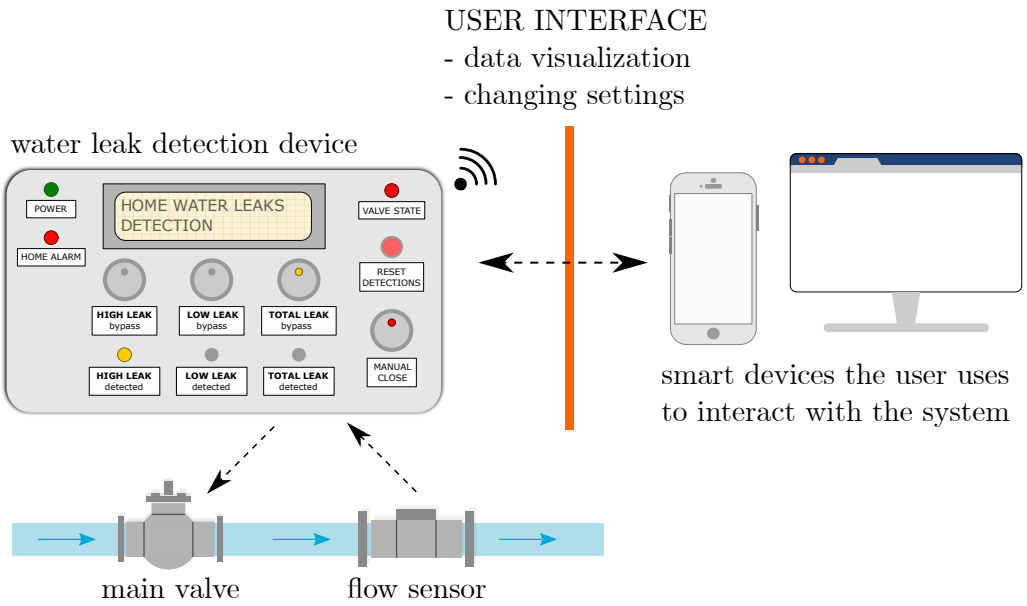


Figure 10.10: The concept of the user interface

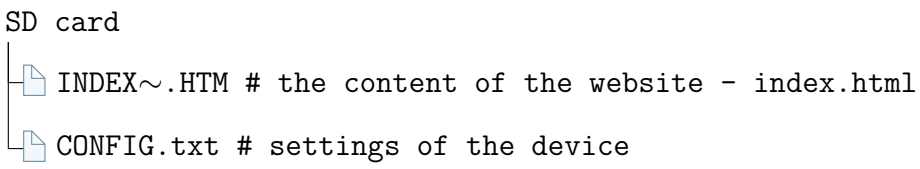
10.5.1 Providing HTML content

The interface is implemented using the Ethernet library and an Ethernet shield, which is attached to the Arduino board. Using the ethernet shield, the microcontroller board works as a simple web server capable of providing static HTML content.

However, there are a couple of issues that need to be tackled. The HTML code cannot be stored directly in the source code as there is only 8KB of SRAM available. The entire content has over 30KB in total, which exceeds the maximum Arduino Mega provides. As a solution, the code is stored on an SD card plugged into the SD card slot that is a part of the Ethernet shield. The SD card is formatted to FAT16 file system since the library supports only FAT16 and FAT32.

Loading the page content

When a client connects to the web server, the program will load the content of the `index.html` file stored on the SD card. The content of the file is not read all at once, but instead, it is read line by line, so the web page could be provided to the client without using too much of the SRAM memory. Since the longest line in the HTML code is 161 characters long, it could be assumed that the loading process does not exceed more than 161B of SRAM at a time. However, since the entire HTML code consists of 1063 lines, it could consequently slow down the updating process of other parts of the system. Therefore, the other SW controllers are being updated simultaneously with each line read off the card.



Due to the size of the web page and the overall performance of Arduino Mega, it takes a few seconds for the site to load up. Nevertheless, the most important part of the system, the water leak detection algorithms, is being updated frequently regardless of the website being read off the SD card.

Replacing static HTML with real-time values

As the content is being read line by line, some of the lines contain a piece of information about the system. For example, the state of the home alarm, daily water consumption, state of the main valve, etc. In order to identify these lines, there was a unique sequence of characters defined within the code, indicating that this part should be replaced with a value. This value can either represent an actual value, such as the current water consumption, or it could be used to specify a CSS class, for example, when changing colors.

Listing 3: Unique sequence of characters identifying a value in HTML

```
%*<value_id>*%
```

Listing 4: An example of a line that needs to be replaced with a value

```
<td>time since the last pulse was detected</td>
<td class="table-border-left">%*12*%</td>
```

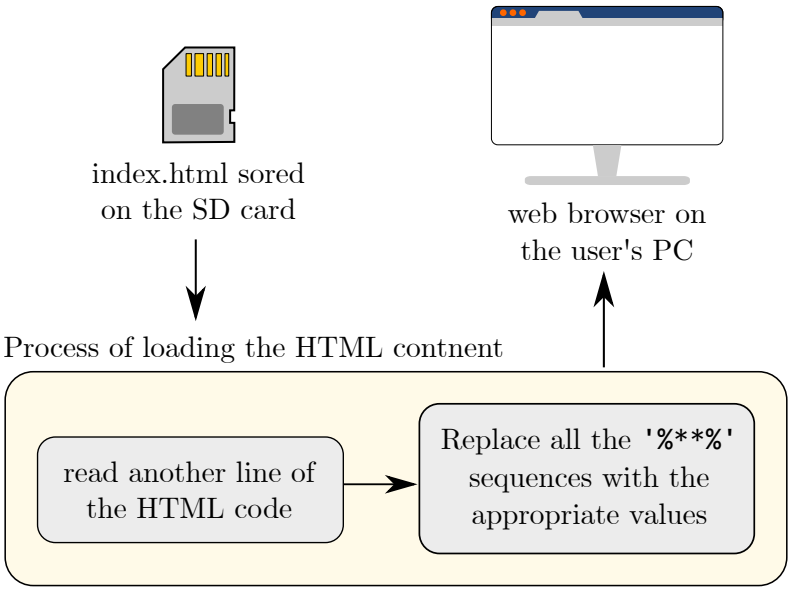


Figure 10.11: The process of loading index.html from the SD card

Different kinds of data belong to different parts of the system. Therefore, if a class should work as a source of data for the HTML code, it needs extend `HTMLDataSource` (see attachment 14.8). The web server controller then goes over individual classes registered as a source of data until it finds the one that holds the data matching the particular number. For instance, the number 12 represents the time since the last pulse was detected and is held in `PulseCounter`.
Each class internally uses a `map` taking advantage of its quick lookup times. The map itself is consists of the `value_id` and a pointer to a function that returns the associated `String` value. This brings the advantage of implementing extra logic within the function. For instance, when requesting a background color for a progress bar, multiple colors can be returned based on how close a leak is to being detected. The process of finding the class holding the particular data is visualized in figure 10.12.

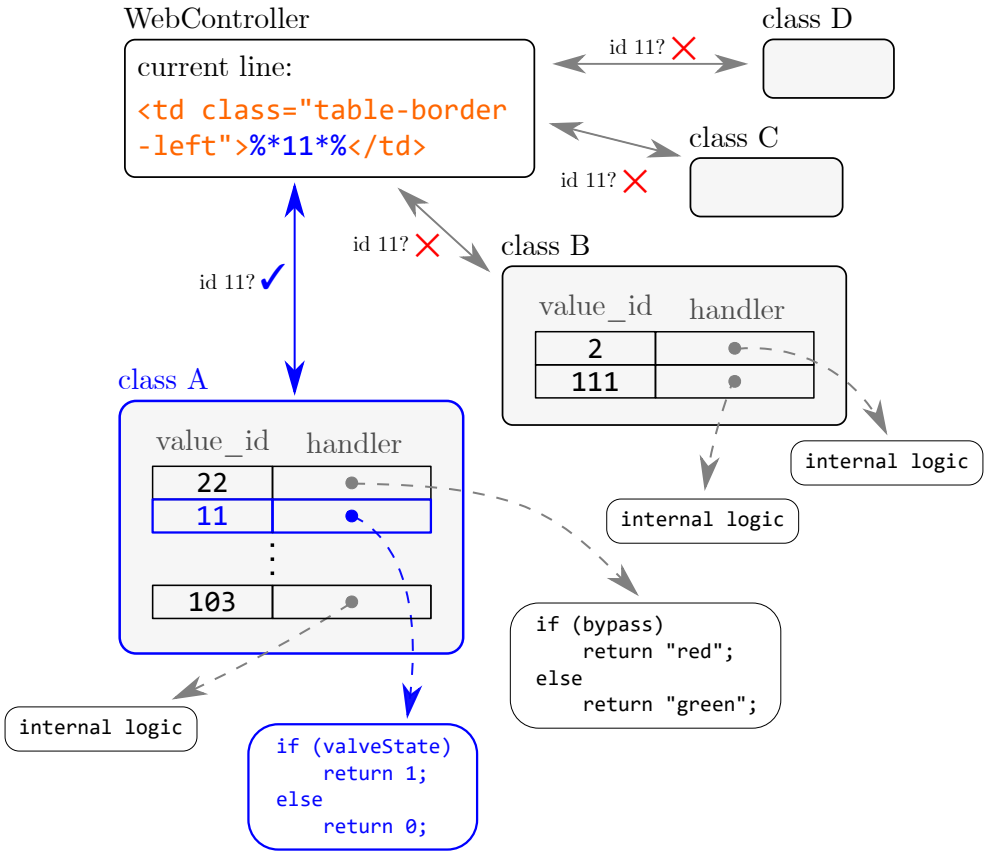


Figure 10.12: The process of inserting data into the HTML code

The UML structure of this part of the system is reminiscent of the previously discussed ones. The `HTMLDataSource` class works as an interface defining the method which each class needs to implement. The method returns a value according to the id given as a parameter. In case the value with this id is not held within the class, an `"UNDEFINED" string` will be returned.

10.5.2 Sending e-mail notifications

Sending e-mails is a way of notifying the user about different changes within the system. As an example, if there has been a leak detected, the user will be immediately notified via e-mail. There are several types of e-mails the user can receive regarding the state of system (see table 10.1).
The Ethernet library defines an EthernetClient which could be used for connecting to a mail server. This part of the system was inspired by a setup article for Arduino that could be found at *smtp2go.com* [14]. SMTP2Go is an e-mail service provider through which the Arduino board sends e-mails directly to the user's e-mail account.

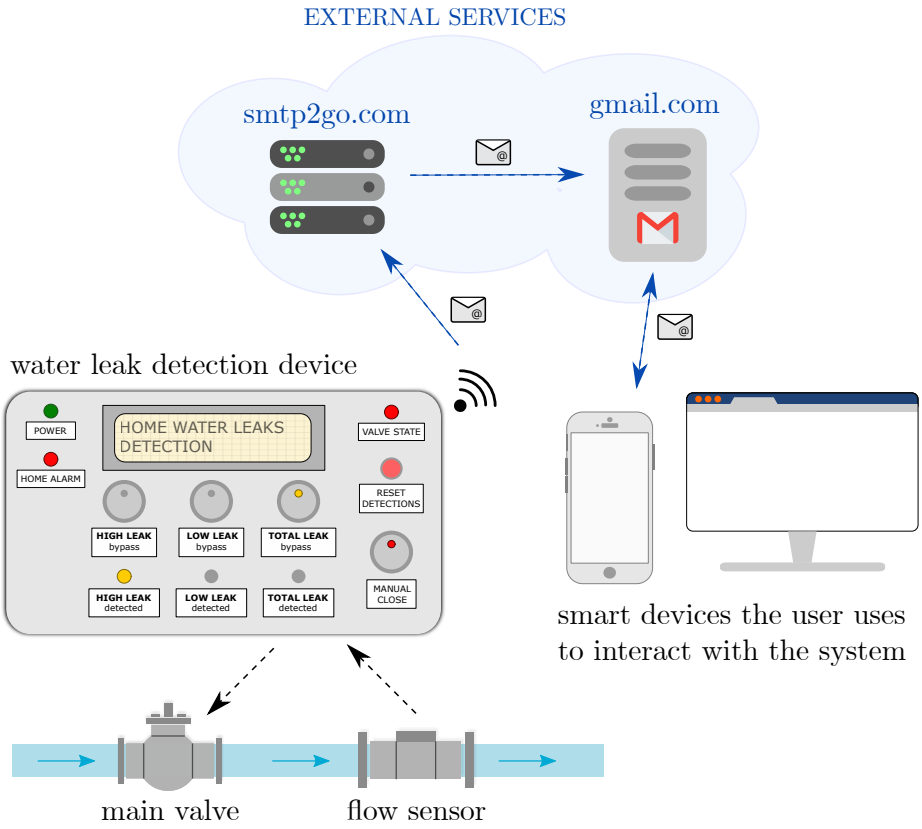


Figure 10.13: The e-mail notification system of the system

In order to keep the project structured, there was a singleton class defined within this project taking care of sending e-mail notifications. This class holds a lookup table of different kinds of e-mails. The UML description of the class can be found in the attachment (14.9).
If a class wants to send off an email, it will pass the type, subject, and content of the e-mail on to the `EmailSender` class. Then, using the lookup table, `EmailSender` will check if the particular type of e-mail has been enabled by the user (see the User manual 14.13). If the particular type of e-mail is enabled, the device will connect to the smtp2go server in order to send the e-mail to the user. The whole process of sending an e-mail does not take more than 3 seconds based on the information printed out on the screen as a part of the logging system.

	type of e-mail	state
1)	the device boots up	OFF
2)	a leak has been detected	ON
3)	daily overview	ON
4)	state of the valve has changed	ON
5)	device has been reset	OFF
6)	state of the home alarm changes	OFF
7)	the user has changed settings	OFF
8)	new settings have been applied	OFF
9)	bypass of a water leak detection has changed	OFF

Table 10.1: The lookup table stored within the `EmailSender` class

10.5.3 Changing settings

The user has the option to change two kinds of settings. The first type is the parameters of individual water leak detection algorithms. The device has pre-defined settings, which are supposed to be changed by the user after installation. The second type of settings is a selection of different types of e-mail notifications the user wants to receive.
When the user wants to change either of the settings, they enter the new parameters through the web interface of the device. As soon as they click on the update button, a piece of JavaScript code built-in the website will generate an HTTP request, so the data could be passed to the device. Due to memory limitation as well as security reasons, the device was programmed in a way, so it does not store more than 160B of the HTTP request. However, this number of bytes is enough for all possible variations of the settings the user may require.

10.5.4 Changing e-mail notifications

Listing 5: An example of an HTTP request to change e-mail notifications

```
1 http://10.10.2.118/?notification=0;0;0;0;0;1;0;0;0;
  ↪ jakub.sil@seznam.cz;
```

This type of an HTTP request is defined by its starting sequence of characters, which is `"/?settings="`. If the device finds this sequence at the beginning of the request, it will treat the rest as the new parameters of the

settings. There are currently nine different kinds of e-mails the user may receive (see table 10.1). Each type of notification is represented in the request either as 1, the notification is enabled, or 0, this type of notification will not be sent to the user. The parameters are supposed to be separated by a semicolon, and their order is the same as in table 10.1. The last parameter is meant to be the user's e-mail address, to which all future e-mails will be sent.

10.5.5 Changing parameters of the water leak detection algorithms

Listing 6: An example of an HTTP request for changing parameters of the water leak detection algorithms

```
1 http://10.10.2.118/?settings=400;30000;200;15000;
  ↪ 26000;40000;13000;20000;600;180000;300;90000;
```

The fundamental structure of this type of HTTP request is the same as when changing e-mail notification settings. The request begins with its defining sequence, `"/?settings="`, and has the parameters separated by semicolons. There are twelve values in total. These could be divided into three groups, where each group represents a water leak detection algorithm. Each of the algorithms requires two values - the thresh-hold value of a leak discovery and the reset value of the algorithm. Additionally, there are two sets of setting for each algorithm - `normalConfig` and `alarmConfig` (see section 10.2.4).

Listing 7: The order of the parameters in an HTTP request

```
1 http://10.10.2.118/?settings=HIGH_NORMAL_ACTION;
  ↪ HIGH_NORMAL_RESET;HIGH_ALARM_ACTION;HIGH_ALARM_RESET;
  ↪ LOW_NORMAL_ACTION;LOW_NORMAL_RESET;LOW_ALARM_ACTION;
  ↪ LOW_ALARM_RESET;TOTAL_NORMAL_ACTION;TOTAL_NORMAL_RESET;
  ↪ TOTAL_ALARM_ACTION;TOTAL_ALARM_RESET;
```

The values are supposed to be in liters, in case they represent an amount of water, or in milliseconds, in case it is a time value. The meanings of the variables are mentioned in sections 6.2.3, 6.3.2, and 6.4

As far as security is concerned, the device analyzes all the parameters before applying them. In particular, it checks if the values are numbers, if they do not overflow nor underflow, etc. Multiple different scenarios have been carried out as a part of the testing process (chapter 12).

10.6 Storing setting on an SD card

The main purpose of storing the current settings is to have a backup in case the power goes out. When the electricity is turned back on, the user will not have to set up all the parameters again as they will be read off the SD card on the start of the device.

10.6.1 Reading data from the configuration file

When the device boots up, it will check if there is a file called `CONFIG.txt` on the SD card. If the file is found, the device will read its content. Upon successful validation of the content, all the water leak detection algorithms will be assigned their appropriate parameters. On the other hand, if the file does not exist or its content does not have the right format, default settings will be used, which are defined in `include/LimitsDefinition.h`.

Listing 8: An example of content of `CONFIG.txt` [2]

```
30000;30;15000;20;       # high water leak
40000;26000;20000;13000; # low water leak
180000;30;90000;15;      # total water leak
1;0;0;0;0;0;1;1;0;jakub.sil@seznam.cz;
```

As far as the structure is concerned, the values on each line have the same meaning. For example, the first two values on the first line represent the reset time and limit of the high-water leak detection algorithm when `normalConfig` is applied. The other two values are for the `alarmConfig` settings. All the time values are in milliseconds, and all values representing an amount of water are in pulses, where one pulse corresponds to ten liters of water. The structure of the last line has been described in section 10.5.4.

10.6.2 Storing data to the configuration file

When the user changes settings, the whole process of storing the new values works in four steps. First of all, a new temporary file called `backup.txt` is created, which works as a backup before overwriting the main configuration file. As the second step, new parameters are stored in this backup file. Thirdly, the main configuration file is overwritten with the new data. And lastly, the temporary backup file is deleted.

[2] The comments are only for explanation purposes. They are not stored within the file.

10.7 Remote connection

If the user is on a business trip or vacation, they can still keep track of their water consumption by visiting the website of the device. As mentioned previously, the website provides information about daily as well as monthly water consumption. Since the device is accessible only from within the LAN, first, they need to connect to their home network using a technology such as SSH or VPN to be able to get onto the website.

10.7.1 API for data analysis

Additionally, the system could define an API for third-party applications allowing them to gain read-only access to data for statistical analysis. This data could be, for example, how much water has been consumed so far, the states of the bypass buttons, how frequently the pulses occur, etc. Such applications would periodically store data off of the device into a database where additional look-ups and filters could be performed. The outcome would be in a form of charts with a user-friendly look giving the user a summary of how their water consumption has been evolving.
However, the idea of API and external compatibility was left as a thought for a future extension. Also, it would be worth considering using a more equipped board, such as Raspberry PI, that would have more stable support of working with databases, data analysis, providing external connections, and other such features than Arduino-based microcontrollers.

11 HW structure of the system

The system is composed of several HW modules that will be discussed in this chapter. The main board of the device is represented by Arduino Mega 2560 with an Ethernet shield attached to it. This board was mainly chosen for its 8KB of SRAM and great library support.

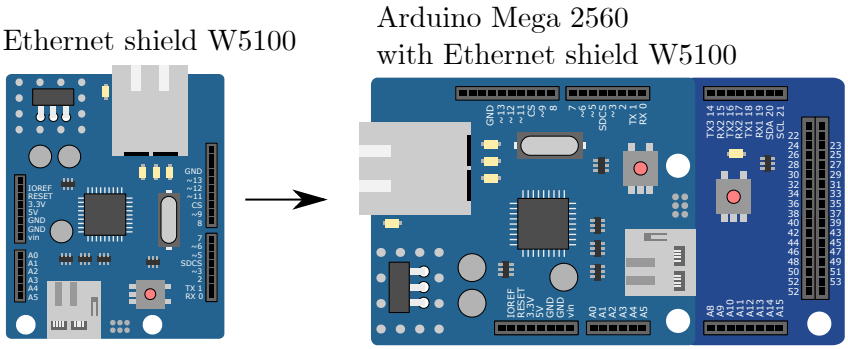


Figure 11.1: The Ethernet shield attached to Arduino Mega 2560

The Ethernet shield is used as a web server, e-mail sender, and storage for settings with its SD slot. After plugging an UTP cabel into the port on the Ethernet shield, the device will be assigned an IP address, assuming there is a DHCP server in the local network. Otherwise, an IP will have to be set up manually in the source code.

11.1 I2C bus communication

There are two devices in the system controlled via the I2C communication protocol. The first one is a DS3231 real-time module, which is used for replacing the Arduino time with date-time in order to visualize information in a more user-friendly way. The second device is the LCD display used for visualizing information regarding the current state of the device. The devices are distinguished by their assigned addresses on the bus. The entire circuit can be seen in figure 11.2.

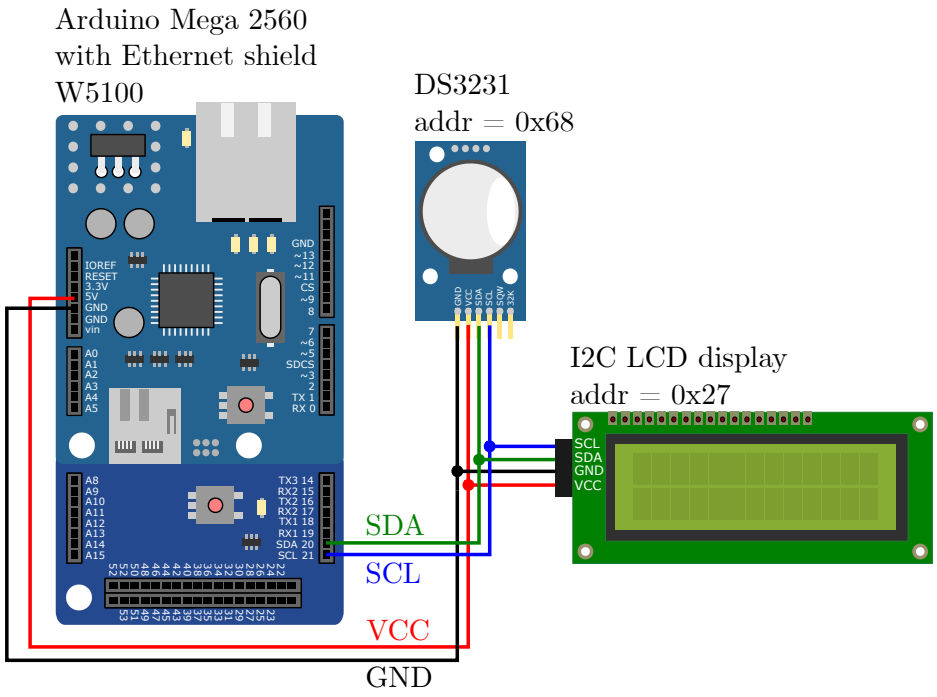


Figure 11.2: Devices connected to the I2C bus

11.2 LED signalization

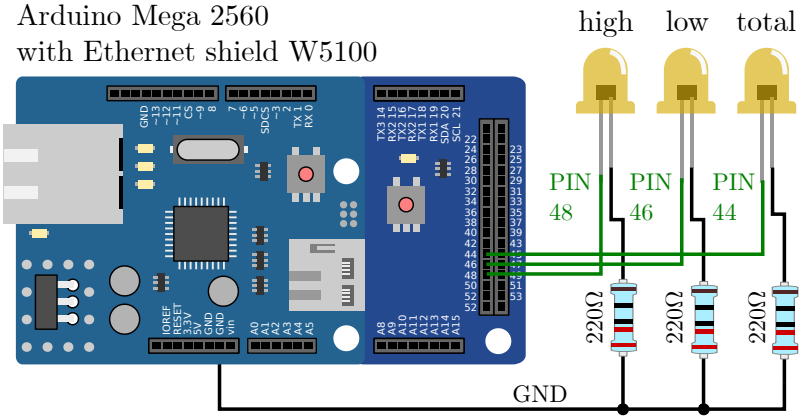


Figure 11.3: LED signalization of different kinds of leaks

When a leak is detected, a simple yet efficient way of visualization is using LEDs. This works as a first signal that something is out of the ordinary. It gives the user a brief picture of what is happening without the need of walking up to the device to read the state from the LCD display.

In order to limit the flow of current thought an LED, a 220Ω resistor was used. Having 5V on the output of the Arduino board, it reduces the current down to 22,72mA ($I = \frac{U}{R}$)[1]. The states of the LEDs correspond to the states of the water-leak detection algorithms. If a leak is detected, the appropriate LED turns on and vice versa.

11.3 Buttons and switches

As mentioned previously, the user has the option to bypass every type of a water-leak detection algorithm if they want to. This might find its reason when they expect an abnormal water consumption that is not meant to be caught by one of the water leak detection algorithms. Also the user can manually change the state of the valve using the appropriate switch button. The last component is the reset button, which is used to reset all three detection algorithms.

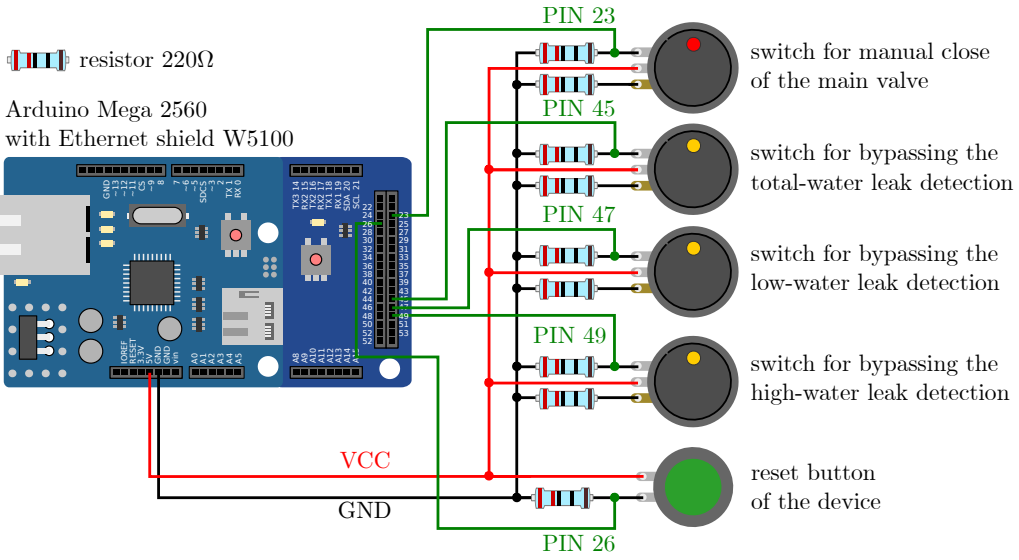


Figure 11.4: Input of the system in a form of buttons and switches

11.4 Output of the main valve

Whenever a water leak is detected, or the user wants to close the main valve manually, the system needs to send a signal to close the main valve. However, the BELIMO valve used in this project works with ~220AC voltage. Therefore, a relay module was used to control the higher voltage from the

[1] I is electric current, U is voltage, and R is resistance.

Arduino board. The relay closes when 5V is provided on its input. For visualization purposes, an extra LED was connected in parallel to the input of the relay.

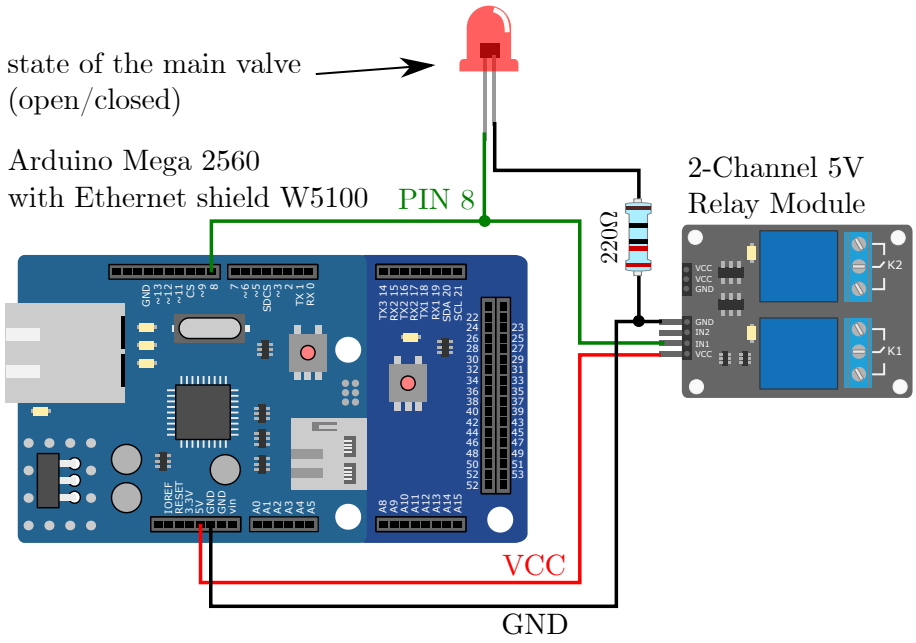


Figure 11.5: Controlling the main valve

11.5 Reading pulses from the flow sensor

The electronic circuit for reading pulses could be thought of as a simple one-button circuit where the button is pressed whenever ten liters of water have flown through the sensor. When this happens, the circuit closes for a period of 70 milliseconds allowing the Arduino board to register the change of states.

For testing purposes, the Arduino itself can work as a pulse generator, so different water-leak detection scenarios can be tested out. The pulses are generated on pin A15 and read on pin A14. This circuit is only used when testing different kinds of water leak-related scenarios.

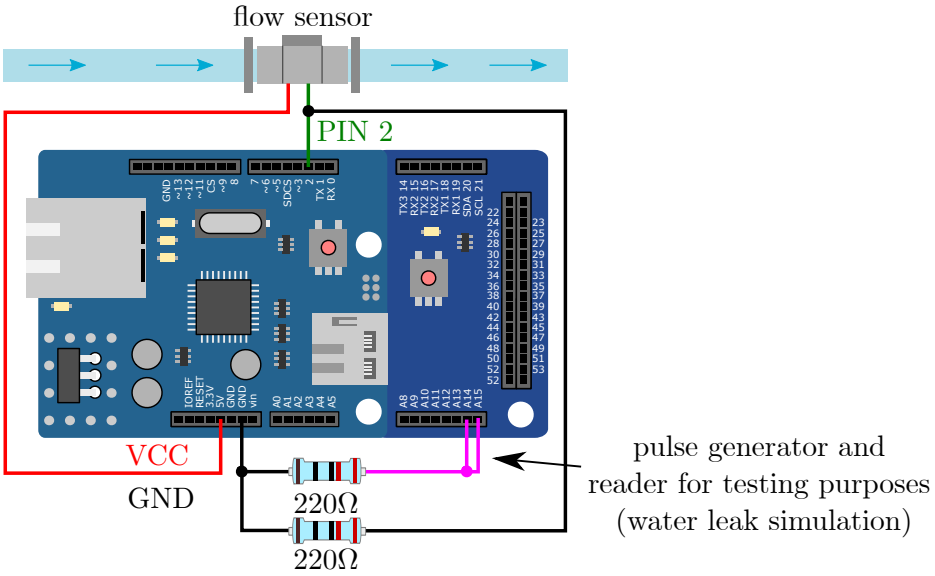


Figure 11.6: The circuit for reading pulses from the flow sensor

11.6 Reading the state of the home alarm

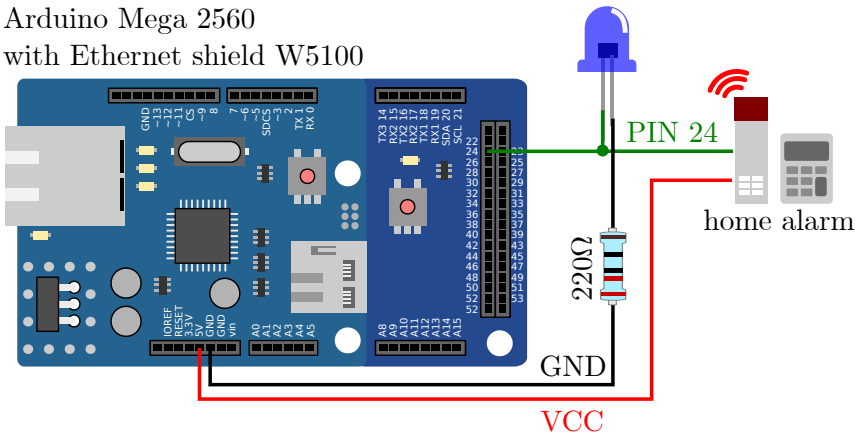


Figure 11.7: Reading the state of the home alarm

Reading the state of the home alarm works essentially the same way as reading pulses from the flow sensor. A certain level of voltage is provided on the alarm's input, and when the user leaves the house, the alarm closes the circuit, letting the voltage appear on the output where it can be read by Arduino.

The settings of the water leak detection algorithms will be changed according to the state of the home alarm. The principal of switching between two set of settings has been explained in section 10.2.4.

12 Testing

The testing part is arguably one of the most crucial steps of the whole process of development. The tests were designed to test both the SW implementation and proper functioning of the electronic circuits. The device was tested in multiple different ways, all of which will be described in this chapter.

12.1 Unit testing

Unit testing mainly focuses on separate fundamental parts of the project. In particular, this involves simulating multiple water leak-related scenarios and receiving an HTTP request when the user changes settings.

12.1.1 Water leak detection algorithms

Scenarios

For clarity, all scenarios were defined in a JSON file that is then loaded into a template test file for its execution. Each type of water leak detection algorithm has its JSON file containing different scenarios related to the particular kind.

Listing 9: An example of a high-water leak scenario [1]

```
{
  "positive": [
    {
      "number_of_generated_pulses": 15,
      "delay_high": 100,
      "delay_low": 100,
      "pulses_limit_action": 15,
      "reset_time": 300
    }
  ],
  "negative:" []
}
```

The file essentially holds two arrays. The array `positive` defines all the scenarios that result in a water leak. The second array called `negative`

[1] The variables `delay_high` and `delay_low` define the shape of each generated pulse

contains scenarios that should not cause a water leak. The array in which a scenario is defined determinates the expected result of the test.

Test templates

The template file is a sketch test file with marked blank spots in it, which are meant to be filled with the parameters of the particular scenario. The template file has the same structure for all three algorithms. All tests are generated automatically by a `Python` script that parses all the JSON files, and for each scenario, it creates a test file off of the template file.

Running tests

PlatformIO comes with unit testing support which was used to run and validate all the tests. PlatformIO provides a guide on how to carry out unit testing along with some examples on their website [9].

Listing 10: Summary of unit testing

```
Test                  Environment     Status   Duration
--------------------  --------------  ------   ------------
test_high_leak_NE_1   megaatmega2560  PASSED   00:00:23.442
test_high_leak_NE_2   megaatmega2560  PASSED   00:00:18.189
test_high_leak_NE_3   megaatmega2560  PASSED   00:00:16.599
test_high_leak_PO_1   megaatmega2560  PASSED   00:00:19.310
test_high_leak_PO_2   megaatmega2560  PASSED   00:00:18.967
test_high_leak_PO_3   megaatmega2560  PASSED   00:00:16.569
test_low_leak_NE_1    megaatmega2560  PASSED   00:00:19.242
test_low_leak_NE_2    megaatmega2560  PASSED   00:00:17.996
.
.
.
test_total_leak_NE_3  megaatmega2560  PASSED   00:00:17.400
==============19 succeeded in 00:05:54.570 ================
```

All tests are supposed to be located in the `test` folder, which was created by the PlatformIO extension in VSCode. When all tests are finished, a final overview is printed out on the screen, so the user can see which tests have failed and which ones have successfully passed.

12.1.2 Receiving an HTTP request

This functionality was tested similarly to the water leak detection algorithms. The only difference in how the tests are executed is the absence of JSON files. Instead, all scenarios were defined directly in the test file, as the initialization is usually a few lines long.

Changing e-mail notifications

Listing 11: A valid HTTP request to change e-mail notifications

```
/?notification=1;1;1;0;0;1;0;0;0;jakub.sil@seznam.cz;
```

The structure of this kind of HTTP request is explained in section 10.5.4. A valid HTTP request of this kind must follow several rules:

1) The number of parameters must be ten.
2) Each of the parameters must end with a semicolon.
3) The first nine parameters must be either a zero or one.
4) The user's email must not be empty.

If any of these rules are broken, the whole HTTP request is ignored and discarded.

Listing 12: Examples of invalid HTTP requests

```
/?notification=0;0;00;0;0;1;0;0;0;jakub.sil@seznam.cz;
/?notification=0;0;0;0;0;1;0;0;0;;
/?notification=0;0;0;0;0;165;0;0;0;jakub.sil@seznam.cz;
/?notification=
/?notification=AA;0;0;0;0;1;0;0;0;jakub.sil@seznam.cz;
```

Changing parameters of the detection algorithms

Listing 13: A valid HTTP request for changing parameters of the water leak detection algorithms

```
/?settings=400;30000;200;15000;26000;40000;13000;20000;600;
↪  180000;300;90000;
```

The meaning of the values is explained in section 10.5.5. Whether an HTTP request of this type is valid or not comes down to several rules:

1) The total number of parameters must be twelve (three water leak detection algorithms and two kinds of settings).
2) Each of the parameters must end with a semicolon.
3) Each number must be an integer less than $2^{32} - 1$ (`UINT_MAX`).
4) Additionally, each number must be a positive number greater than 0.

Listing 14: Examples of invalid HTTP requests

```
/?settings=400;30000;0;15000;26000;40000;13000;20000;600;
↪ 180000;300;90000;
/?notification=
/?settings=400;30000;200;15000;26000;5294967295;13000;20000;
↪ 600;180000;300;90000;
/?settings=400;30000;200;15000;26000;40000;13000;20000;600;
↪ 180000;-300;90000;
/?settings=15.968;30000;200;15000;26000;40000;13000;20000;600;
↪ 180000;300;90000;
/?settings=400;30000;200;abcd;26000;40000;13000;20000;600;
↪ 180000;300;90000;
```

12.1.3 Running Unit tests

If the user wants to run the unit tests described above, they first need to enable testing. To do so, they need to uncomment line `"// define UNIT_TEST"` in the `include/Setup.h` file. As the next step, they need to save the file and go to the `unit_tests` folder. Once in the folder, all they need to do is run the following commands.

```
python test_http_rqst.py # test detection algorithms
python test_leaks.py     # test receiving HTTP request
```

12.2 System testing

System testing was carried out once all the unit tests had been successful and all the parts of the device were wired up and put in an enclosure. This part of testing did not focus on the internal implementation as much as on the overall required functionality.

12.2.1 Scenarios

1) When a water leak is detected and the bypass is off, the valve must be closed.
2) When a water leak is detected and the bypass is on, the valve must not be closed.
3) If the valve is closed due to a water leak, using the switch for manual closing of the valve must not affect the state of the valve – it must remain closed.
4) When pressing the reset button, all the water leak detection algorithms must be reset.
5) If the valve is manually closed, pressing the reset button must not change the valve's state.
6) After a leak has been detected, and the valve has not been manually closed, pressing the reset button must open the valve again.

All the scenarios listed above were carried out manually by pressing the buttons and observing the output. The LEDs indicating a detected water leak should work regardless of the state of the bypass switch.

12.3 Interface testing

The last part of the testing process gave attention to the user interface. This testing was carried out at the end of the project. Its main purpose was to test if the website shows up-to-date information regarding the system accurately. All the information shown on the website should match both the values displayed on the LCD and the states of LEDs. Also, when the user changes settings, they should see the update once the website refreshes.

12.3.1 Testing e-mail notifications

Whether or not e-mail notifications are working correctly was tested by manually invoking circumstances under which an e-mail would be sent off. This was done for all the e-mail notifications defined within this project. All of them can be seen listed in table 10.1.

13 Conclusion

The purpose of this project was to design a device that would detect water-leak occurrences in a family house, such as a cracked or leaking pipe.
In the first part of the thesis, different kinds of water leaks were taken into consideration, along with possible ways of detecting them. All the options were then narrowed down to using a flow sensor attached to the main pipe entering the house. Having the flow sensor as the main input of the device, three types of water leaks were defined, covering the most common scenarios of what could happen.
The core of the system is represented by Arduino, which implements all three water leak detection algorithms. By reading data from the sensor, the micro-controller analyzes whether or not there is a water leak taking place somewhere in the house. If a water leak has indeed been detected, the device will respond by closing the main valve of the house in order to avoid any potential damages. Also, taking advantage of the state of the home alarm, the device offers the option to adjust the parameters when there is nobody at home, as there should be no water consumption whatsoever.
The user is provided information regarding the state of the device in multiple different ways. A simple yet efficient way was implemented by using LEDs, which visualize two-state information such as the state of the main valve, which type of water-leak has been detected, state of the home alarm, etc. More detailed information about the system can be then seen on an LCD. Also, the system has a built-in web server that works as a user interface, providing a variety of information regarding the system, such as the daily water consumption or the current settings of the device. Through the interface, the user can set up multiple kinds of e-mail notifications that will be sent off to them whenever the associated event happens.
As for the interaction with the system, the user can manually close the main valve, reset the device, or bypass any of the water leak detection algorithms. Bypassing the algorithms be may useful when they are expecting an abnormal water consumption that should not be detected, such as when watering their backyard. They can also adjust the parameters of the algorithms on the website according to their needs.
In the end, multiple tests were carried out to test the system thoroughly before installation. This part included unit testing of the fundamental parts of the device as well as overall system testing by scenarios once the device had been put into its enclosure.

Bibliography

[1] *Arduino - tutorials* [online]. [cit. 2021/9/2]. Available at: https://www.arduino.cc/en/Tutorial/HomePage.

[2] *Arduino Products (Arduino Nano, Arduino Mega 2560)* [online]. [cit. 2021/9/2]. Available at: https://www.arduino.cc/en/Main/Products.

[3] *ArduinoDS3231 (external library)* [online]. Free Software Foundation, Inc. [cit. 2021/12/4]. Available at: https://github.com/jarzebski/Arduino-DS3231.

[4] *ArduinoSTL (external library)* [online]. Free Software Foundation, Inc. [cit. 2021/15/3]. Available at: https://github.com/mike-matera/ArduinoSTL.

[5] *BELIMO water applications technical brochure* [online]. BELIMO, 2020. [cit. 2021/27/2 - chapter 9 - Ball valves; page 56]. Available at: https://www.belimo.at/mam/europe/technical-documentation/technische_brosch%C3%BCren/water_solutions/belimo_water-applications_technical-brochure_en-gb.pdf.

[6] *Cyble Sensor data-sheet* [online]. Itron, 2011. [cit. 2021/24/1]. Available at: https://www.itron.com/-/media/feature/products/documents/brochure/cyble_sensor_pb_en_1211.pdf.

[7] *Key factors to consider when choosing a microcontroller* [online]. John Koon, 2019. [cit. 2021/8/2 - the list of the key factors]. Available at: https://www.microcontrollertips.com/key-factors-consider-choosing-microcontroller/.

[8] *PlatformIO (platform for embedded development)* [online]. [cit. 2021/5/3]. Available at: https://platformio.org/.

[9] *PlatformIO (Unit Testing)* [online]. [cit. 2021/2/4]. Available at: https://docs.platformio.org/en/latest/plus/unit-testing.html.

[10] *Raspberry Pi (Remote Access)* [online]. [cit. 2021/9/2]. Available at: https://www.raspberrypi.org/documentation/remote-access/.

[11] *Renesas FS2012 data-sheet* [online]. Renesas Electronics Corporation, 2018. [cit. 2021/22/1 - 9. Analog Output]. Available at: https://www.renesas.com/us/en/document/dst/fs2012-datasheet.

[12] *Residential End Uses of Water Version 2* [online]. The Water Research Foundation, 2016. [cit. 2021/3/1 - Figure 4. Average daily indoor per household water use REU1999 and REU2016]. Available at: `https://www.circleofblue.org/wp-content/uploads/2016/04/WRF_REU2016.pdf`.

[13] *Residential End Uses of Water Version 2* [online]. The Water Research Foundation, 2016. [cit. 2021/3/1 - Figure 1. Indoor household use by fixture]. Available at: `https://www.circleofblue.org/wp-content/uploads/2016/04/WRF_REU2016.pdf`.

[14] *Setting up Arduino with SMTP2GO* [online]. [cit. 2021/22/3]. Available at: `https://www.smtp2go.com/setupguide/arduino/`.

[15] *STM32 32-bit Arm Cortex MCUs* [online]. STMicroelectronics. [cit. 2021/9/2 - the first paragraph]. Available at: `https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html`.

[16] *STM32Cube Ecosystem* [online]. STMicroelectronics. [cit. 2021/9/2 - the first paragraph]. Available at: `https://www.st.com/content/st_com/en/stm32cube-ecosystem.html`.

[17] *Sliding Window Algorithm* [online]. Said Sryheni, 2020. [cit. 2021/4/2 - 3.3. Sliding Window Algorithm]. Available at: `https://www.baeldung.com/cs/sliding-window-algorithm`.

[18] *eses I2C 20x4 display* [online]. eses. [cit. 2021/16/3]. Available at: `https://dratek.cz/docs/produkty/0/756/eses1474620659.pdf`.

[19] *House Meter Check for Leaks* [online]. Home Water Works. [cit. 2021/7/1 - section - Leaks; paragraph - Whole House Meter Check for Leaks]. Available at: `https://www.home-water-works.org/indoor-use/leaks?fbclid=IwAR17yanHGw56pPT-5pdO4N7f7NZEAETIKkHnZvxLmJFNZ6WhP3DNgw8I7-k`.

[20] Horstmann, C. *Object-Oriented Design and Patterns.* John Wiley & Sons, 2005. ISBN 0471744875.

List of Abbreviations

LDD - leak (water) detetection device
RAM - random-access memory
SRAM - static random-access memory
IDE - integrated development environment
OOP - Object Oriented Programming
SW - software
HW - hardware
IoT - Internet of Things
UML - Unified Modeling Language
MAC - Media Access Control
I/O - input/output
LCD - liquid crystal display
SD - Secure Digital
HTTP - Hypertext Transfer Protocol
HTML - Hypertext Markup Language
LED - light emitting diode
SSH - Secure Shell
FTP - File Transfer Protocol
SCP - Secure, Contain and Protect
SCCM - standard cubic centimeters per minute
I2C - Inter-integrated circuit bus communication
UTP - Unshielded Twisted Pairs
DHCP - Dynamic Host Configuration Protocol
IP - Internet Protocol
JSON - JavaScript Object Notation
ID - Identification
API - Application Programming Interface
LAN - Local Area Network
VPN - Virtual Private Network

14 Attachments

14.1 `PulseCounter` class

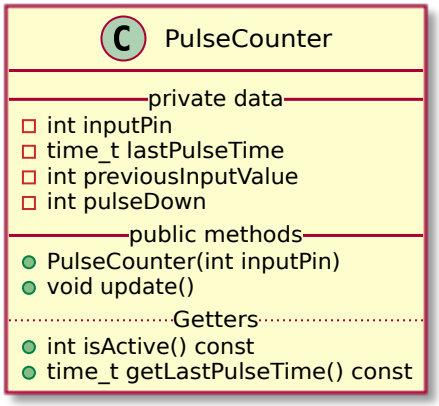


Figure 14.1: UML diagram of class PulseCounter

- `PulseCounter` - private data
  - **`int inputPin`** - The number of the input pin where the pulses are supposed to occur.
  - **`time_t lastPulseTime`** - The last time when a pulse wast detected on the input pin.
  - **`int previousInputValue`** - The last state of the input pin. A pulse is detected when the state of the pin goes from low to high.
  - **`int pulseDown`** - A flag indicating if a pulse has been detected.
- `PulseCounter` - public methods
  - **`PulseCounter(int inputPin)`** - The constructor of the class. It takes one parameter, which is the number of the input pin.
  - **`void update()`** - This method is used for updating the class. It includes reading the current state of the input pin and determining whether or not there is a pulse on the input pin.

- `PulseCounter` - getters
  - **`int isActive() const`** - If there is a pulse on the input pin, it returns 1. Otherwise, 0 will be returned.
  - **`time_t getLastPulseTime() const`** - It returns the Arduino time when the last pulse was detected. This information is used by the algorithms in order to find out if there has been a water leak detected.

14.2 `Button` class

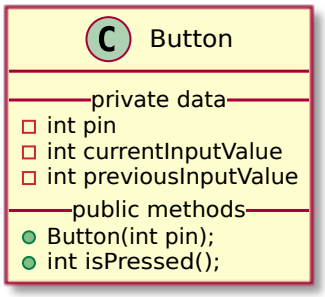


Figure 14.2: UML of the reset button

- `Button` - private data
  - **`int pin`** - The number of the pin the button is connected to.
  - **`int currentInputValue`** - The current voltage value read off the input pin. This value is either high or low.
  - **`int previousInputValue`** - The previous state of the input pin.
- `Button` - public methods
  - **`Button(int pin)`** - The constructor of the class. It takes one parameter, which is the number of the pin the button is connected to.
  - **`int isPressed()`** - It reads the voltage on the input pin and compares it to the previous one. If the previous value is low and the current one is high, 1 will be returned as the button is currently being pressed. Otherwise, the method will return 0.

14.3 `LeakDetectionConfig_t` class

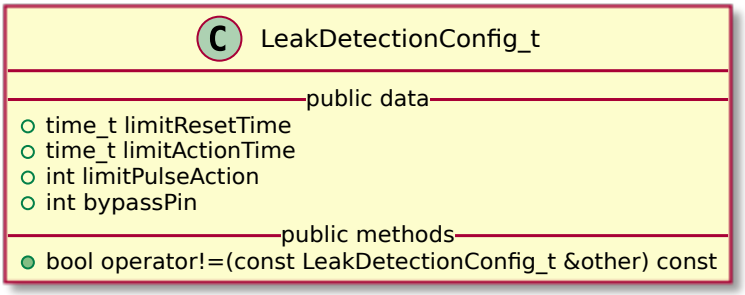


Figure 14.3: UML of the `LeakDetectionConfig_t` structure

- `LeakDetectionConfig_t` - public data
  - **`time_t limitResetTime`** - The time limit determining if the algorithm should be reset. This depends on the particular implementation of the algorithm. For example, `limitReset` time is treated differently in a high-water leak detection algorithm and a low-water leak detection algorithm.
  - **`time_t limitActionTime`** - The time limit defining a low-water leak (see section 6.3.2).
  - **`int limitPulseAction`** - The number of pulses defining a high-water leak and total total-water leak (see section 6.2.3).
  - **`int bypassPin`** - The number of the pin that the bypass button is connected to. The user can manually bypass (switch off) the detection algorithm for as long as they want to.
- `LeakDetectionConfig_t` - public methods
  - **`bool operator!=(const LeakDetectionConfig_t &other) const`** - Overloaded operator for comparison of two different sets of settings.

14.4 `ALeakDetectable` class

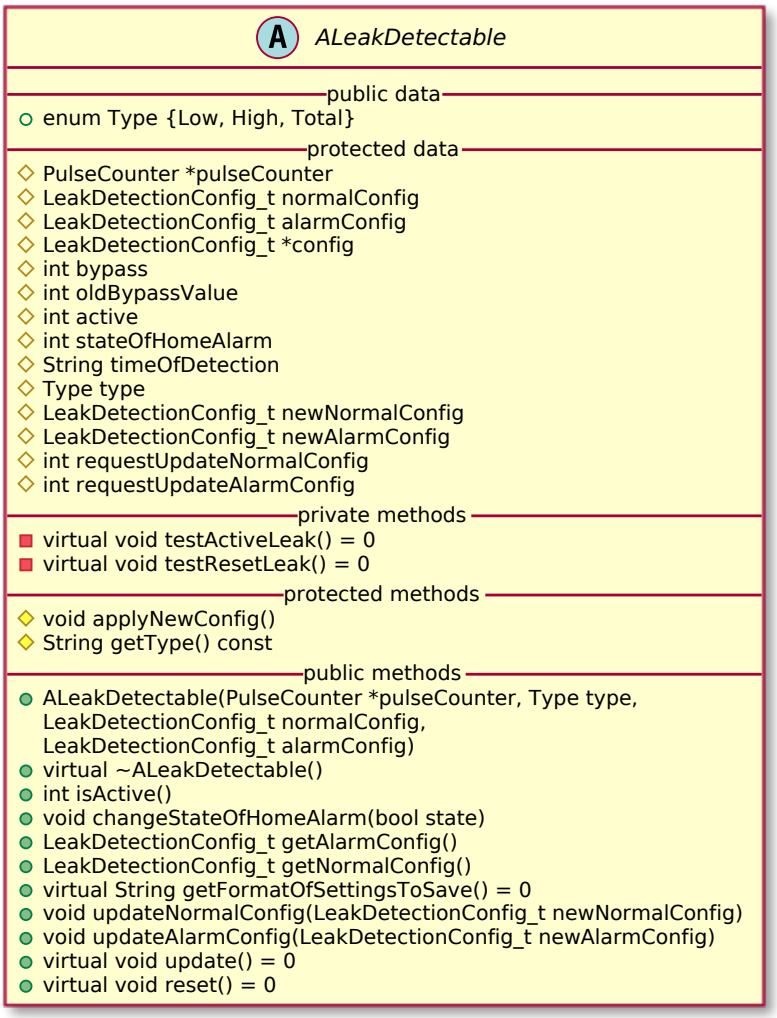


Figure 14.4: UML diagram of abstract class `ALeakDetectable`

- `ALeakDetectable` - public data
  - **`enum Type`** - An enumeration for distinguishing different types of water leak detection algorithms - high, low, and total.
- `ALeakDetectable` - protected data
  - **`PulseCounter *pulseCounter`** - A pointer to an instance of `PulseCounter` (section 10.2.1) used for counting pulses generated by the flow sensor.

- **`LeakDetectionConfig_t normalConfig`** -
  The settings of the algorithm when the house is occupied.
- **`LeakDetectionConfig_t alarmConfig`** -
  The settings of the algorithm when the house is empty.
- **`LeakDetectionConfig_t *config`** - The current settings being used by the algorithm at the moment. This depends on the state of the home alarm, as explained in section 10.2.4.
- **`int bypass`** - The current state of the bypass pin. If the bypass is on, the output of the algorithm has no affect on the state of the main valve.
- **`int oldBypassValue`** - The previous state of the bypass pin. This value is used to keep track of the change of voltage on the input pin (from high to low or from low to high).
- **`int active`** - The value indicating whether or not a leak has been detected by the algorithm (1/0).
- **`int stateOfHomeAlarm`** - The current state of the home alarm. This value is used to decide which of the two set of configurations should be used for the algorithm.
- **`String timeOfDetection`** - The time when a leak was detected stored in a `String` format.
- **`Type type`** - The type of the water leak detection algorithm. This is used when debugging or sending e-mail notifications to distinguish different types of algorithms.
- **`LeakDetectionConfig_t newNormalConfig`** - New normal settings to be applied when the user decides to change some of the current parameters.
- **`LeakDetectionConfig_t newAlarmConfig`** - New alarm settings to be applied when the user decides to change some of the current parameters.
- **`int requestUpdateNormalConfig`** - A flag indicating the user has changed the settings, and they need to be put in place when the algorithm resets.
- **`int requestUpdateAlarmConfig`** - A flag indicating the user has changed the alarm settings, and they need to be put in place when the algorithm resets.

- `ALeakDetectable` - private methods
  - **`virtual void testActiveLeak() = 0`** - This method tests if the algorithm has detected a water leak. This is determined by the logic of the algorithms described previously.
  - **`virtual void testResetLeak() = 0`** - This method tests if the algorithm should be reset. This is determined by the logic of the algorithms described previously.
- `ALeakDetectable` - protected methods
  - **`void applyNewConfig()`** - This method applies the new settings according to the value of the flags `requestUpdateNormalConfig` and `requestUpdateAlarmConfig`. This method is called when the algorithm resets.
  - **`String getType() const`** - It returns the type of the particular water leak detection algorithm in a `String` format.
- `ALeakDetectable` - public methods
  - **`ALeakDetectable(PulseCounter *pulseCounter,`**
    `Type type,`
    `LeakDetectionConfig_t normalConfig,`
    `LeakDetectionConfig_t alarmConfig)` -
    This is the constructor of the class. As far as the parameters are concerned, it takes a pointer to an instance of `PulseCounter` as the input of the algorithm along with default settings and the type.
  - **`virtual ~ALeakDetectable()`** - The descructor of the class.
  - **`int isActive()`** - It returns 1 if a leak has been detected. Otherwise, 0 will be returned.
  - **`void changeStateOfHomeAlarm(bool state)`** - This method is used to notify the class about the new state of the home alarm.
  - **`LeakDetectionConfig_t getAlarmConfig()`** -
    After calling this method, the current alarm configuration will be returned.
  - **`LeakDetectionConfig_t getNormalConfig()`** -
    After calling this method, the current normal configuration will be returned.

- `virtual String getFormatOfSettingsToSave() = 0` - It returns both types of settings of the algorithm in a `String` format, so they could be stored on the SD card.
- `void updateNormalConfig(LeakDetectionConfig_t newNormalConfig)` - This method is called when the current normal settings are required to be changed from the outside of the class.
- `void updateAlarmConfig(LeakDetectionConfig_t newAlarmConfig)` - This method is called when the current alarm settings are required to be changed from the outside of the class.
- `virtual void update() = 0` - This method is called to update the whole class. The class is supposed to be updated periodically as frequently as possible.
- `virtual void reset() = 0` - This method is called to reset the algorithm from the outside of the class. For instance, when the user presses the reset button (see section 10.2.2).

14.5 Hierarchy of the water leak detection algorithms

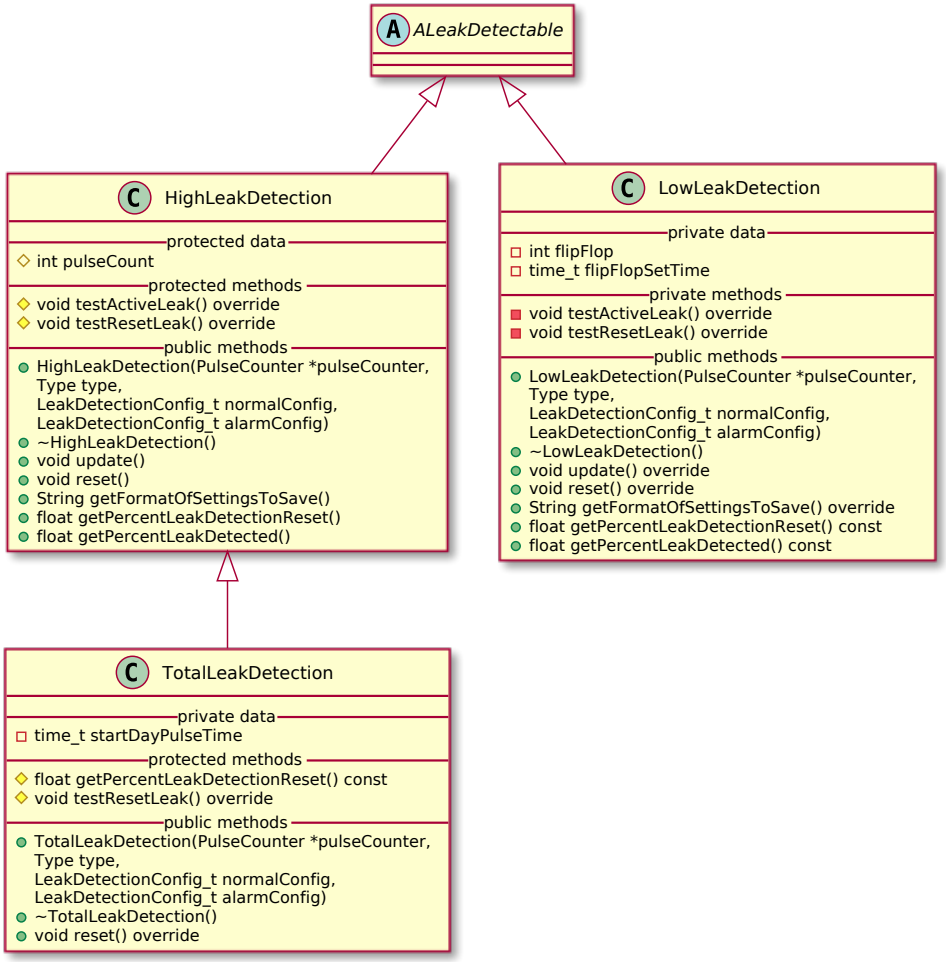


Figure 14.5: UML diagram of the water leak detection algorithms and their hierarchy

14.6 `LeakController` class

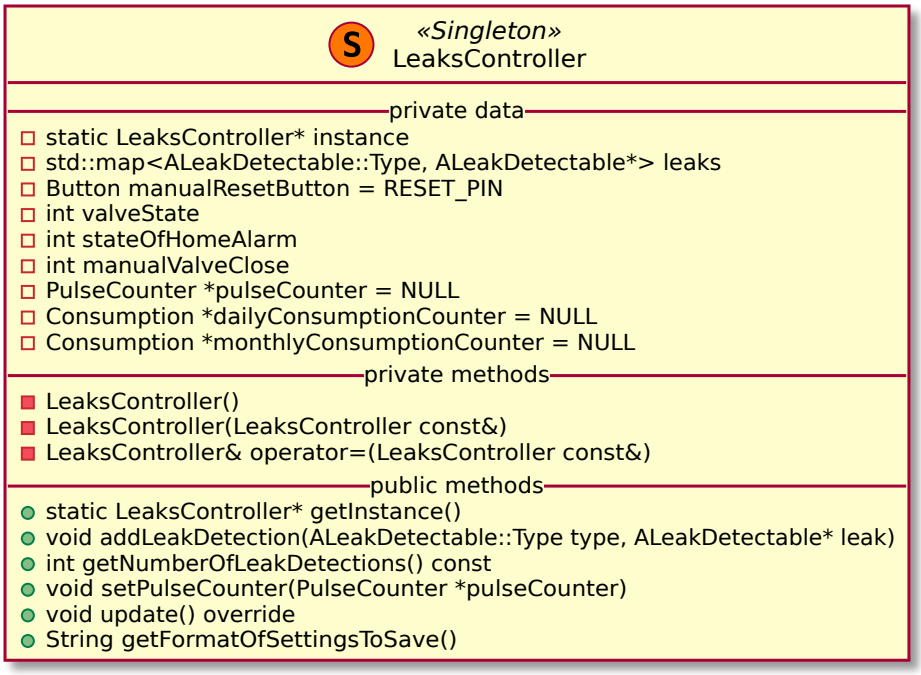


Figure 14.6: UML diagram of the class `LeakController`

- `LeakController` - private data
  - **`static LeaksController* instance`** - The only existing instance of the class. This is a part of the singleton design pattern.
  - **`std::map<ALeakDetectable::Type, ALeakDetectable*> leaks`** - A map of instances of different water leak detection algorithms. In order to have quick access to the instances of the algorithms by their type, an `std::map` [1] was used. Taking advantage of this data structure, the look-up time reduces, in average, down to $\mathcal{O}(1)$. For example, if the user changes the parameters of the high-water leak detection, new settings need to be passed on to the particular instance.
  - **`Button manualResetButton = RESET_PIN`** - The reset button the user is required to press after a leak was detected and they have dealt with all the consequences.

[1] `std::map` was added as a part of an external library, `ArduinoSTL` [4]

- **`int valveState`** - The state of the main valve. This value is sent from the microcontroller to the valve as shown in figure 10.6.
- **`int stateOfHomeAlarm`** - The state of the home alarm. This value determines which of the two settings should be used in the detection algorithms.
- **`int manualValveClose`** - An indication if the user has manually closed the main valve (1/0).
- **`PulseCounter *pulseCounter = NULL`** - A pointer to an instance of the class `PulseCounter`, which is used for counting the pulses generated by the flow sensor.
- **`Consumption *dailyConsumptionCounter = NULL`** - A pointer to an instance of the `Consumption` class used for keeping track of total daily water consumption.
- **`Consumption *monthlyConsumptionCounter = NULL`** - A pointer to an instance of the `Consumption` class used for keeping track of total monthly water consumption.

- `LeakController` - private methods
  - **`LeaksController()`** - The constructor of the class. A private constructor is one of the major characteristics of the singleton design pattern.
  - **`LeaksController(LeaksController const&)`** - The copy constructor of the class.
  - **`LeaksController& operator=(LeaksController const&)`** - An overload assignment operator.

- `LeakController` - public methods
  - **`static LeaksController* getInstance()`** - The instance of the class will be returned.
  - **`void addLeakDetection(ALeakDetectable::Type type, ALeakDetectable* leak)`** - This method is used to add another water leak detection algorithm to the class.
  - **`int getNumberOfLeakDetections() const`** - It returns the total number of leak detection algorithms held within the class.
  - **`void setPulseCounter(PulseCounter *pulseCounter)`** - This method is used to set the pointer to an instance of `PulseCounter`.

– **`void update() override`** - This method is called to update all the instances and variables held in the class. For example, `stateOfHomeAlarm`, `manualValveClose`, or `manualResetButton`. It cascadingly updates all the instances of the algorithms as well.

– **`String getFormatOfSettingsToSave()`** - This method is used when storing settings of all the algorithms on the SD card.

14.7 `Consumption` class

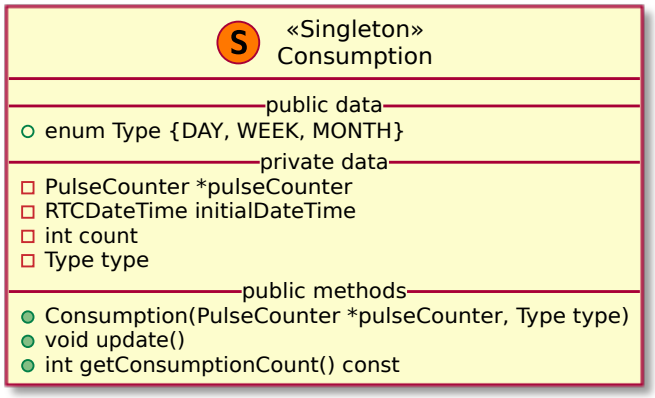


Figure 14.7: UML diagram of the class `Consumption`

- `Consumption` - public data
  - **`enum Type DAY, WEEK, MONTH`** - An enumeration defining the period of monitoring water consumption.
- `Consumption` - private data
  - **`PulseCounter *pulseCounter`** - A pointer to an instance of the `PulseCounter` class, which is used for counting pulses generated by the flow sensor.
  - **`RTCDateTime initialDateTime`** - Initial date-time of the monitoring period. It is used as a reference to when the monitoring process began. [2]
  - **`int count`** - This variable holds the number of pulses detected during the monitoring period.

[2] `RTCDateTime` is a part of the Arduino-DS3231 library [3]

  - **`Type type`** - The type of the monitoring period. This can be set to either a day, week, or month.

- `Consumption` - public methods
  - **`Consumption(PulseCounter *pulseCounter, Type type)`** - This method is the constructor of the class. It takes as a parameter a pointer to an instance of `PulseCounter` and the `Type` of the monitoring period.
  - **`void update()`** - This method is called from the outside of the class when it needs to be updated. The process of updating includes checking if there is a pulse generated by the sensor, and if so, incrementing the counter by 1.
  - **`int getConsumptionCount() const`** - It returns the number of pulses detected so far within the monitoring period.

14.8 `WebServer` class

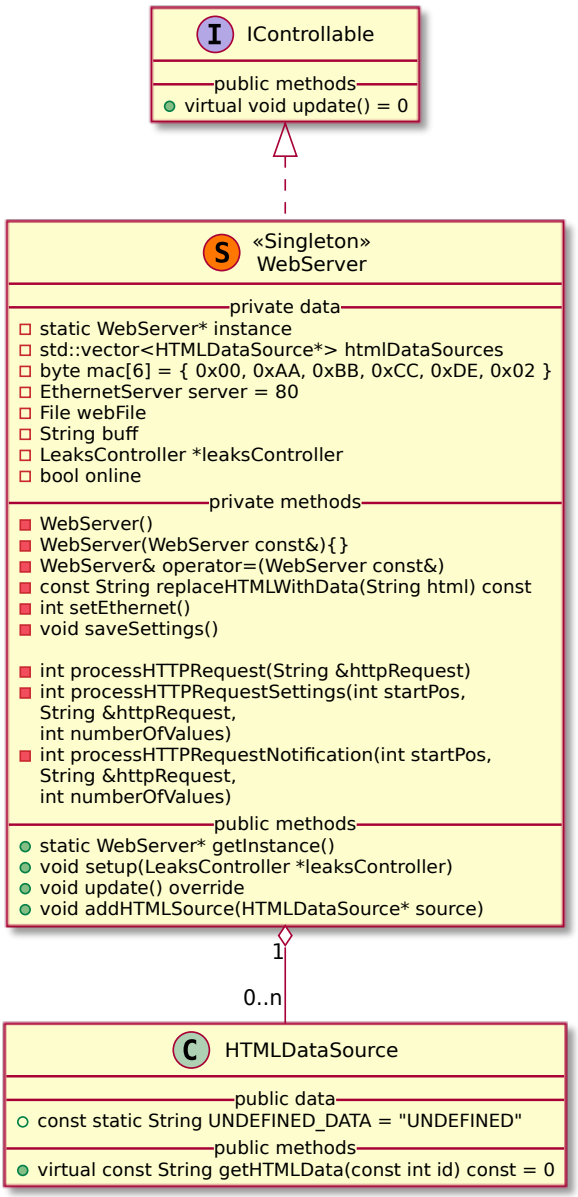


Figure 14.8: UML diagram of the `WebServer` class

- `WebServer` - private data
  - **`static WebServer* instance`** - The instance of the class.
  - **`std::vector<HTMLDataSource*> htmlDataSources`** - A list of classes registered as a data source for the HTML code.

  - **byte mac[6] = { 0x00,0xAA,0xBB,0xCC,0xDE,0x02 }** - The chosen MAC address of the Ethernet shield. This is supposed to be a unique address within the network.
  - **EthernetServer server = 80** - An instance of an Ethernet server running on port 80.
  - **File webFile** - An instance of a file stored on an SD card. This could represent `index.html` when loading the web site.
  - **String buff** - A `String` buffer used to store an HTTP request when changing settings.
  - **LeaksController *leaksController** - A reference to `LeaksController`, so it could be updated simultaneously with each line being read off the SD card.
  - **bool online** - A flag indicating if the server is up and running.

- `WebServer` - private methods
  - **WebServer()** - The constructor of the class.
  - **WebServer(WebServer const&)** - The copy constructor of the class.
  - **WebServer& operator=(WebServer const&)** - An overload assignment operator.
  - **const String replaceHTMLWithData(String html) const** - This method inserts appropriate data into the `String` given as a parameter using the technique explained previously. The parameter represents one line of the HTML code.
  - **int setEthernet()** - This method is used to initialize the Ethernet shield. It is called only once when the system boots up. If everything goes well and the server starts running, 1 will be returned. Otherwise, 0 will be returned as a flag that something went wrong.
  - **void saveSettings()** - This method saves the current settings of the device. This is discussed more in detail in section 10.5.3.
  - **int processHTTPRequest(String &httpRequest)** - This method processes a received HTTP request. The method will analyze the request and validate it. This HTTP requests are used when changing settings of the device. If the request is invalid, the method will return 0. Otherwise, 1 will be returned.

- **`int processHTTPRequestSettings(int startPos, String &httpRequest, int numberOfValues)`** - This method is used to process an HTTP request sent to change the current settings of the leak detection algorithms. The meaning of the return values is the same as in method `processHTTPRequest`.
- **`int processHTTPRequestNotification(int startPos, String &httpRequest, int numberOfValues)`** - This method is used to process an HTTP request sent to change the settings of the current e-mail notifications. The meaning of the return values is the same as in method `processHTTPRequest`.

- `WebServer` - public methods
  - **`static WebServer* getInstance()`** - It returns the instance of the WebServer.
  - **`void setup(LeaksController *leaksController)`** - This method is used to set the `LeaksController`, so it could be updated regardless of a file being read off the SD card.
  - **`void update() override`** - This method is called in order to update the controller.
  - **`void addHTMLSource(HTMLDataSource* source)`** - Another source of data for the website can be added to the system using this method.

14.9 `EmailSender` class

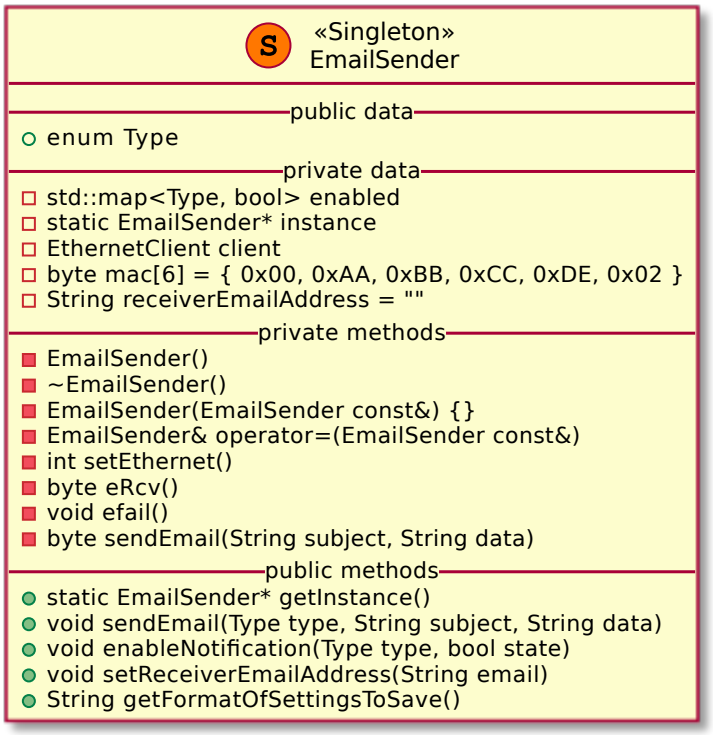


Figure 14.9: UML diagram of the `EmailSender` class

- `EmailSender` - public data
  - **`enum Type`** - An enumeration defining different types of e-mails.
- `EmailSender` - private data
  - **`std::map<Type, bool> enabled`** - The lookup table used for finding out whether or not the particular type is enabled.
  - **`static EmailSender* instance`** - The instance of the class.
  - **`EthernetClient client`** - An Ethernet client used for connecting to the smtp2go server when sending an e-mail.
  - **`byte mac[6] = {0x00,0xAA,0xBB,0xCC,0xDE,0x02}`** - The MAC address of the Ethernet shield. This is used for initialization of the shield in case it has not been initialized yet.
  - **`String receiverEmailAddres`** - The user's e-mail address.
- `EmailSender` - private methods

- **`EmailSender()`** - The constructor of the class.
- **`~EmailSender()`** - The destructor of the class.
- **`EmailSender(EmailSender const&)`** - The copy constructor of the class.
- **`EmailSender& operator=(EmailSender const&)`** - An overload assignment operator.
- **`int setEthernet()`** - Initialization of the Ethernet shield in case if has not been initialized yet. Upon successful initialization, 1 will be returned. Otherwise, the method will return 0.
- **`byte eRcv()`** - This method is used for reading data using the Ethernet client. If the reading fails, 0 will be returned. Otherwise, the method returns 1.
- **`void efail()`** - This method is used for closing the Ethernet client after the connection has failed.
- **`byte sendEmail(String subject, String data)`** - This method is used to send an e-mail to the user's e-mail account. If sending the e-mail fails, 0 will be returned. Otherwise, the method will return 1.

- `EmailSender` - public methods
  - **`static EmailSender* getInstance()`** - It returns the instance of the class.
  - **`void sendEmail(Type type, String subject, String data)`** - This method is used externally to send an e-mail of a specific type.
  - **`void enableNotification(Type type, bool state)`** - This method is called to either enable or disable an e-mail notification.
  - **`void setReceiverEmailAddress(String email)`** - This method is used to change the user's e-mail address.
  - **`String getFormatOfSettingsToSave()`** - It returns a `String` formatted so it could be stored on the SD card (see section 10.6).

14.10 Assembly process of the device

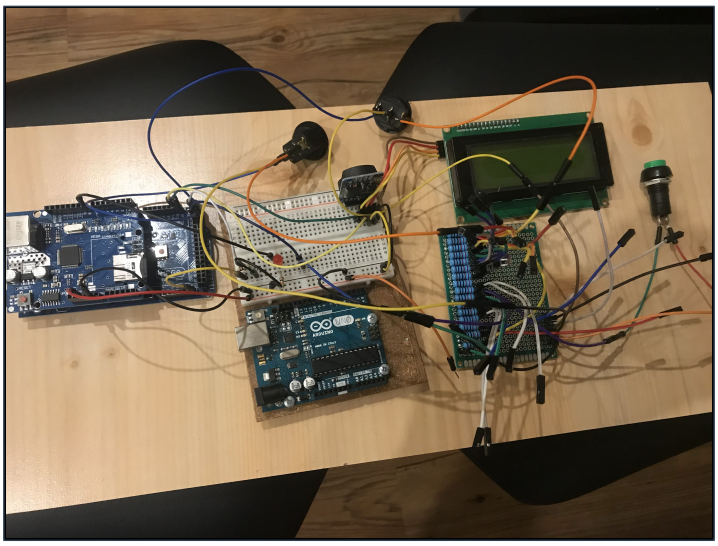


Figure 14.10: Process of assembling the device (1)

Figure 14.11: Process of assembling the device (2)

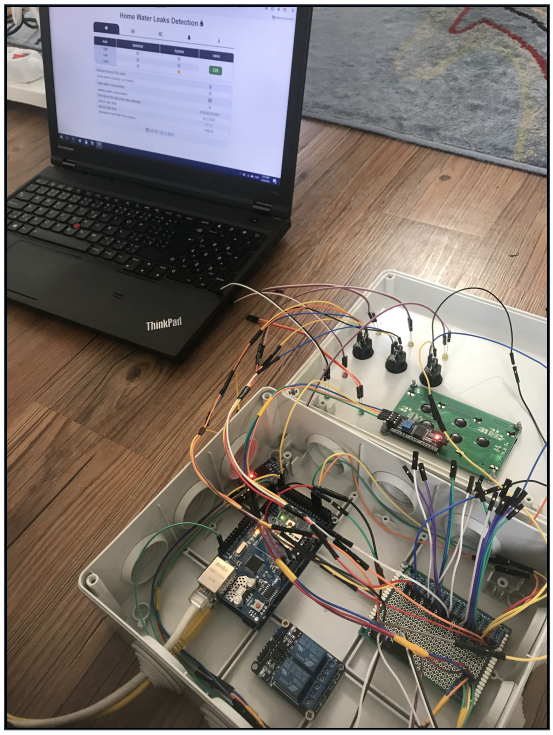


Figure 14.12: Process of assembling the device (3)

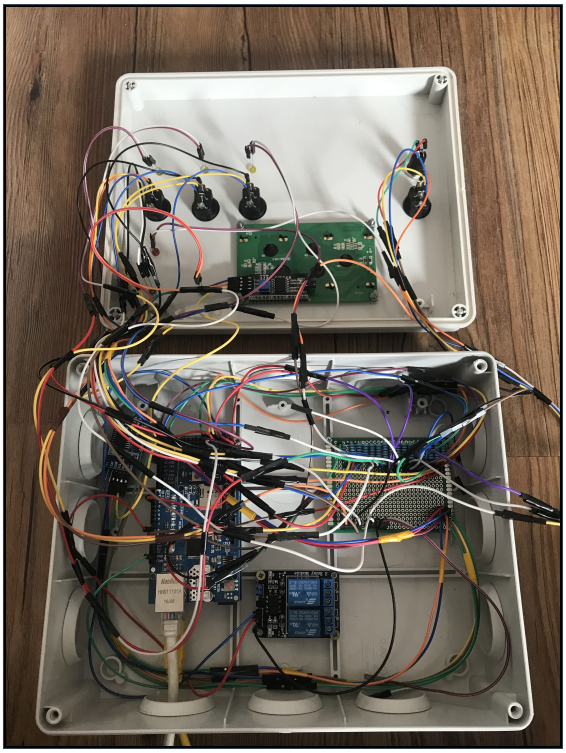


Figure 14.13: Process of assembling the device (4)

Figure 14.14: Process of assembling the device (5)

Figure 14.15: Process of assembling the device (6)

14.11 E-mail notifications

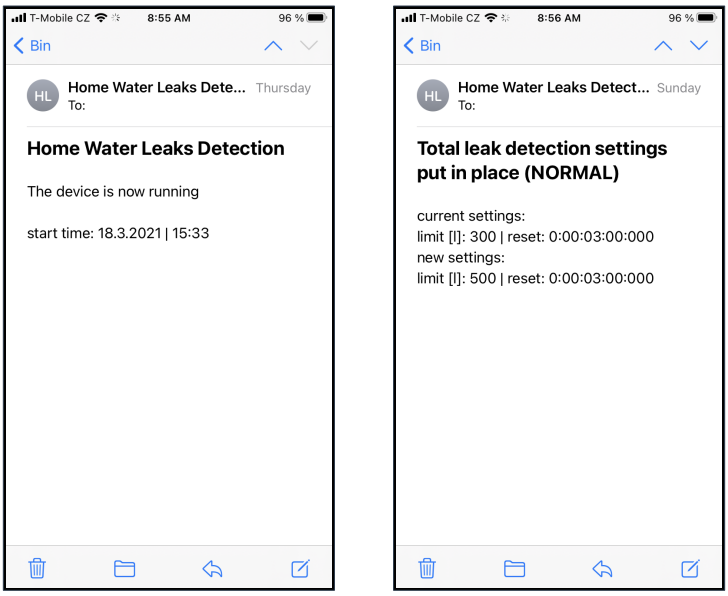


Figure 14.16: Examples of e-mail notifications the user may receive (1)

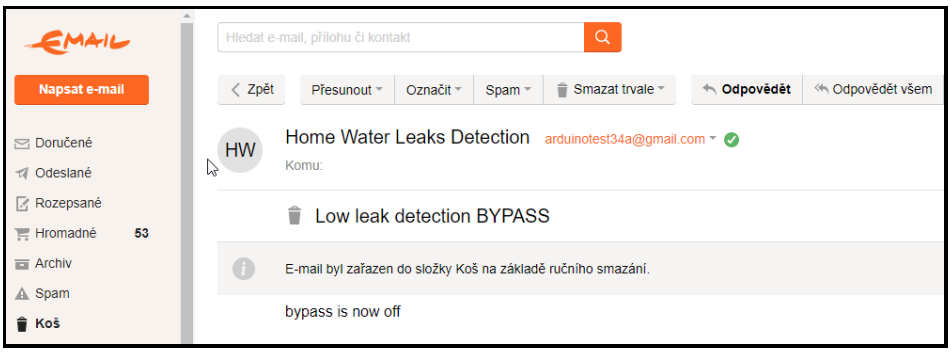


Figure 14.17: Examples of e-mail notifications the user may receive (2)

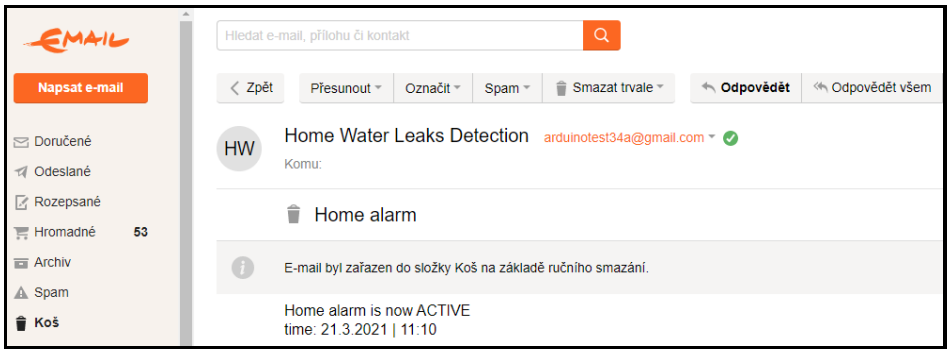


Figure 14.18: Examples of e-mail notifications the user may receive (3)

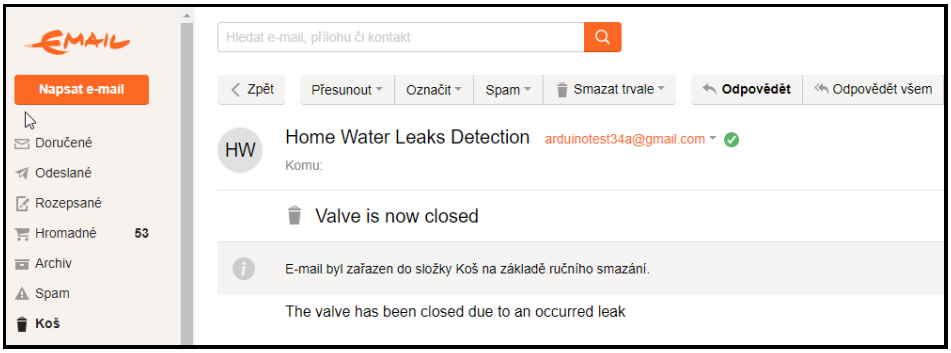


Figure 14.19: Examples of e-mail notifications the user may receive (4)

14.12 The user interface in a web browser

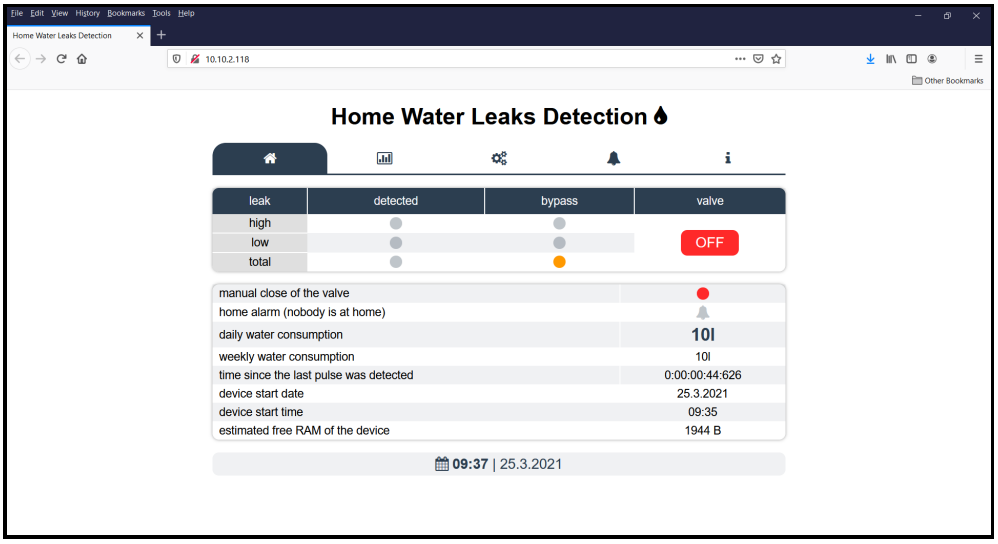


Figure 14.20: the user interface tab (1)

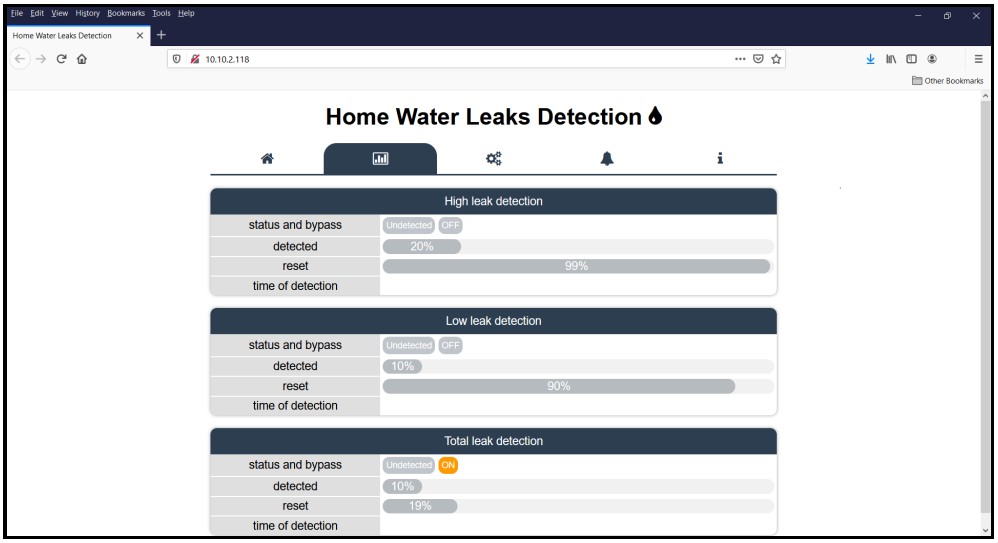


Figure 14.21: the user interface tab (2)

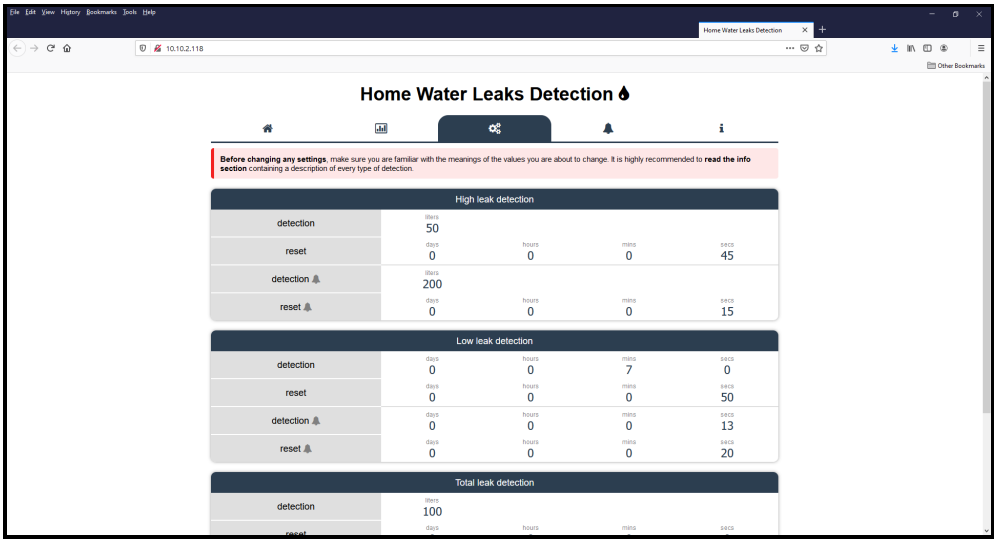


Figure 14.22: the user interface tab (3)

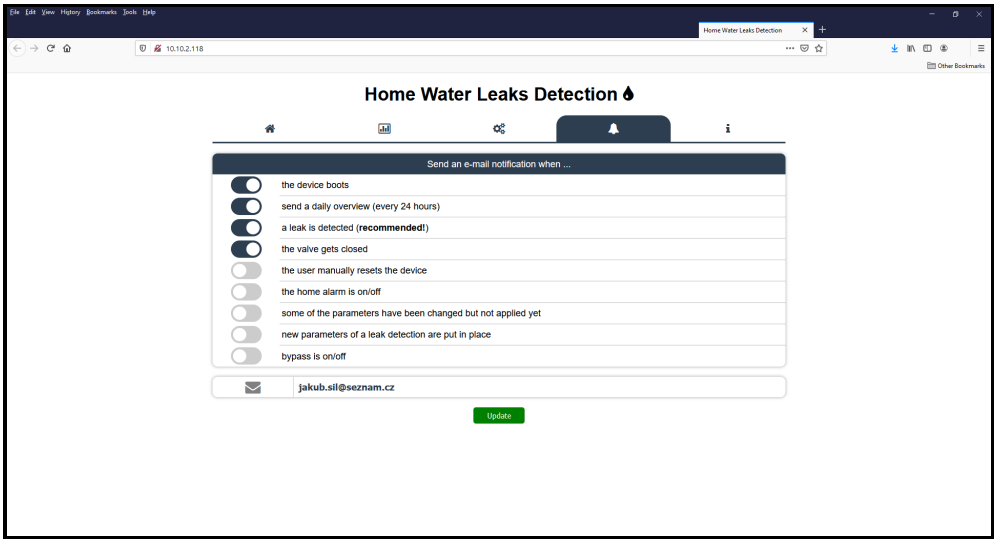


Figure 14.23: the user interface tab (4)

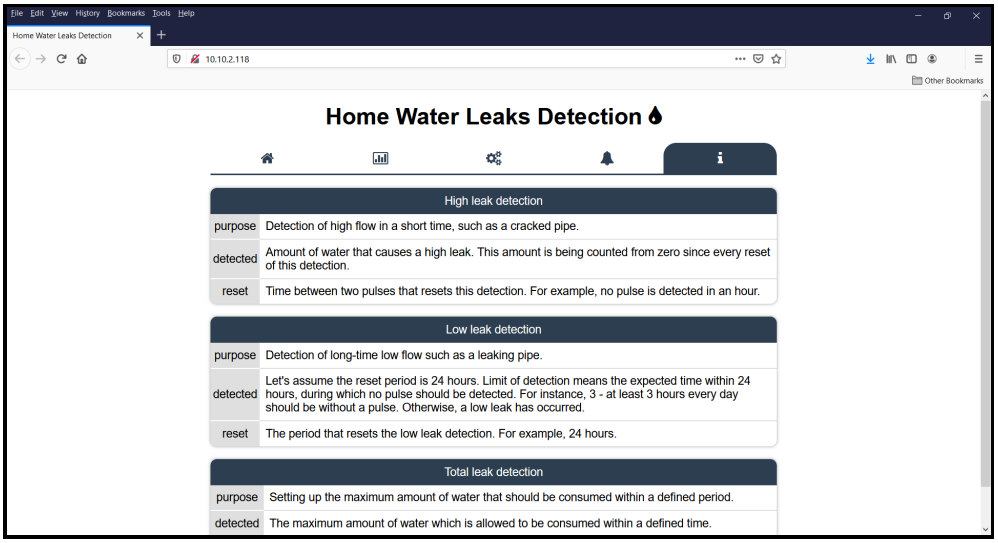


Figure 14.24: the user interface tab (5)

14.13 User manual

The user manual describes the content the user can see on the website, the meaning of individual components on the lid of the box, which the device has been put in, and some fundamentals instructions on how to make changes within the source code.

14.13.1 Website of the device

The website works as a user interface through which the user can see various kinds of information about the whole system as well as change settings if needed.

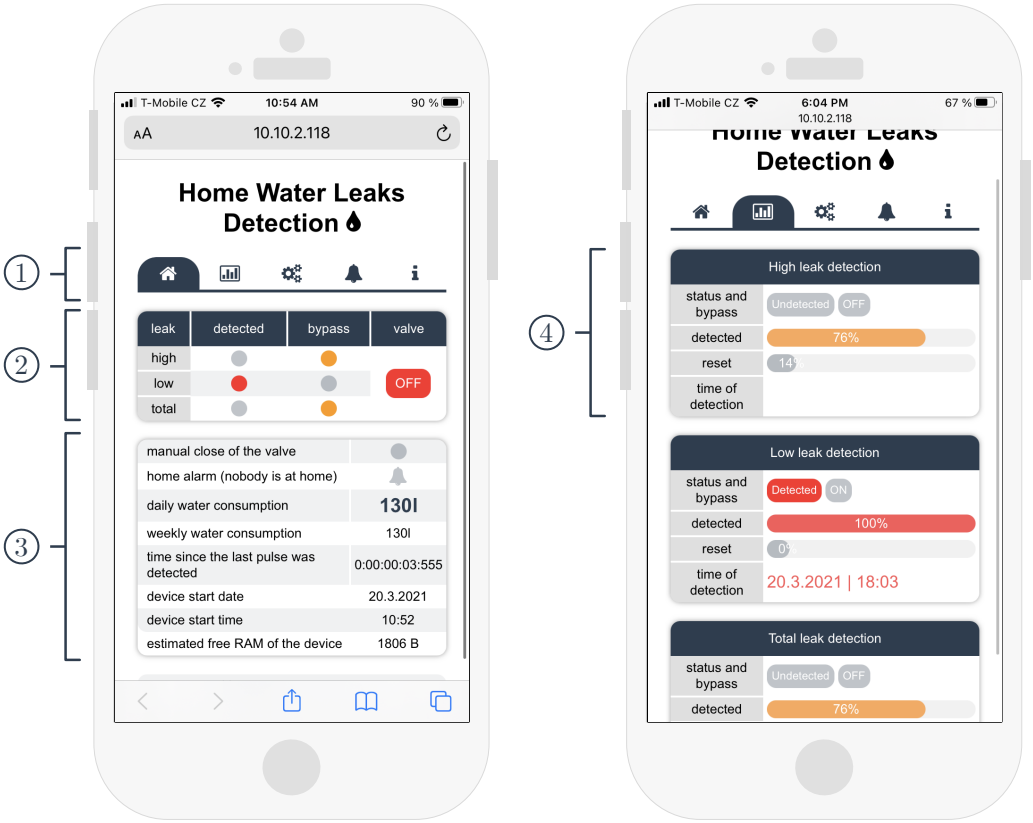


Figure 14.25: The website user interface (1)

1) **Tabs with different content**

   (i) **Home** - A tab containing an overview about the current state of the device.

   (ii) **Statistics** - Once this tab is selected, the user can see the progress of all three water leak detection algorithms.

   (iii) **Settings** - On this tab, the user can change the parameters of all the water leak detection algorithms.

   (iv) **Notifications** - A tab for changing e-mail notifications.

   (v) **Information** - This tab contains a brief description of the parameters of the water leak detection algorithms.

2) **States of all three water leak detection algorithms** - The states are represented as circles. If a water leak detection algorithm is bypassed, the appropriate circle will turn yellow. If a leak is detected, its associated circle will turn red.

3) **Other information**

   (i) An indication whether or not the valve has been manually closed.

   (ii) The current state of the home alarm.

   (iii) Daily and monthly water consumption.

   (iv) Time since the last pulse was detected. This could be used as an indication that the device is detecting the pulses correctly.

   (v) The time when the device started.

   (vi) Remaining free SRAM of the device.

4) **The progress of a water leak detection algorithm** - This tab provides more detailed information about water leaks, including states of the bypasses, time of detection, etc.

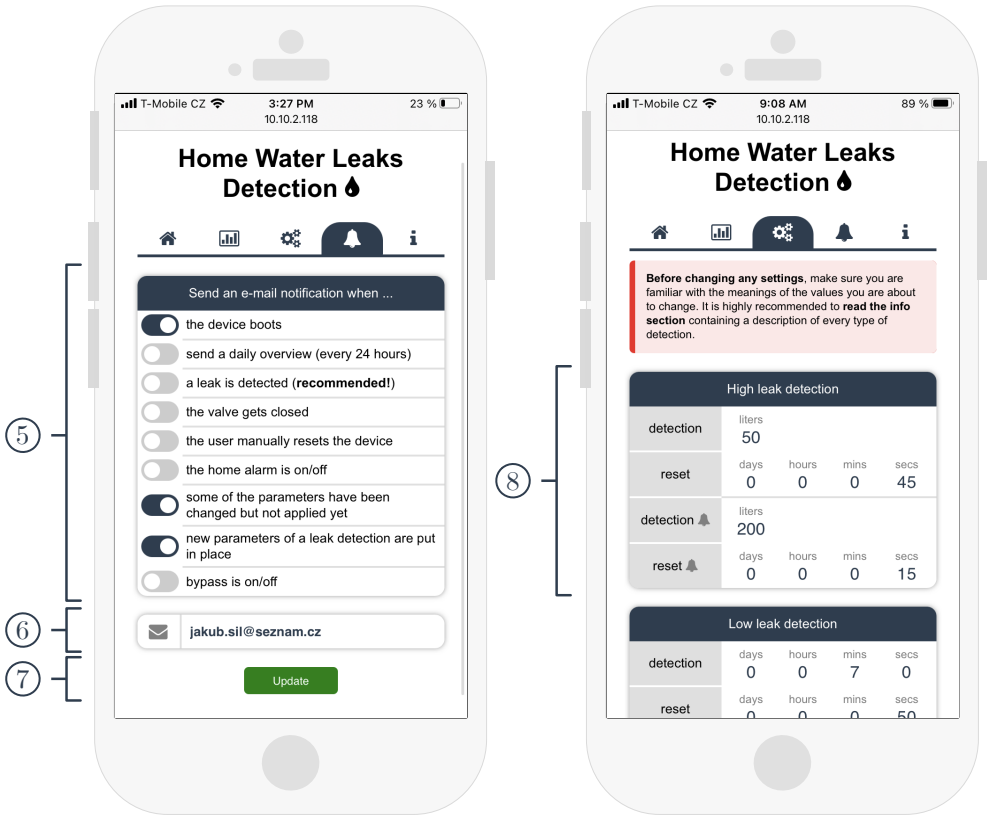


Figure 14.26: The website user interface (2)

5) **A list of e-mail notifications the user can receive**

6) **The user's e-mail address** - All selected e-mails will be sent to this e-mail address.

7) **The update button** - As soon as the user clicks on the button, all changes will be applied.

8) **Settings of a water leak detection algorithm** - Each algorithm has two kinds of settings depending on the state of the home alarm. The home alarm settings are marked with a bell symbol.

14.13.2 Enabling and disabling functionality

Before the code is compiled and uploaded to Arduino, the user can either enable or disable several different features of the device. For instance, they may want to turn on debugging or entirely disable the home alarm option if they do not have it. All changes are supposed to be done within `include/Setup.h`, which works as an initialization configuration file.

Listing 15: the configuration file

```
1  //#define UNIT_TEST
2
3  #ifndef UNIT_TEST
4  //# define DEBUG
5  //# define HOME_ALARM_SETTINGS
6  # define WEB_SERVER
7  # define LCD_DISPLAY
8  #else
9  # define GENERATE_PULSES
10 # define EMAIL_NOTIFICATION
11 #endif
12
13 #ifdef WEB_SERVER
14 # define EMAIL_NOTIFICATION
15 #endif
```

All changes will be applied after the code is re-built and uploaded to the Arduino board. In the listing above, the red-highlighted lines represent functions that have been disabled, and the green lines represent the currently enabled functionality of the device. For example, if the user wants to disable email-notifications, they need to comment out line 14.

14.13.3 Description of the physical device

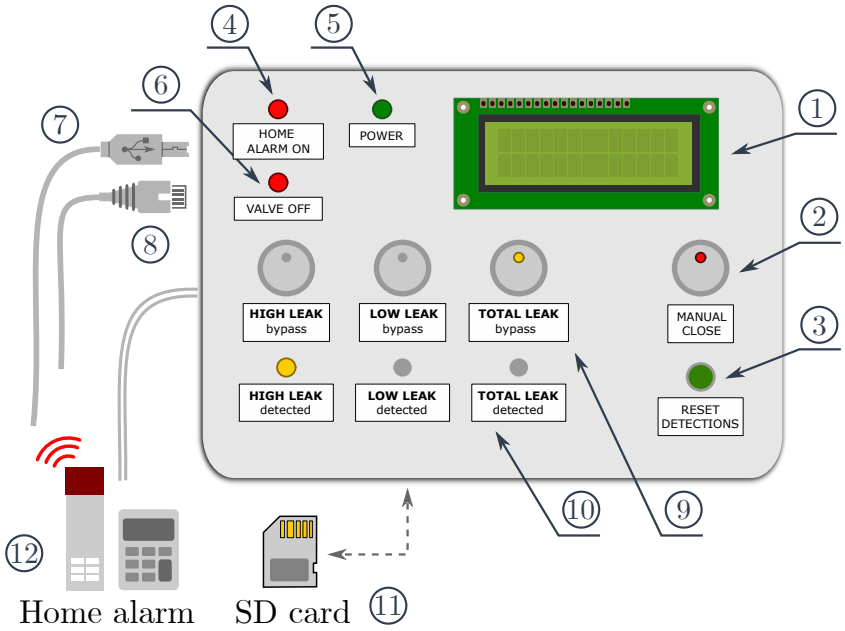


Figure 14.27: Final description of the device

1) The LCD display visualizing information about the system.
2) The switch for manual closing of the main valve of the house.
3) The reset button of the device. As soon as the button is pressed, all the water leak detection algorithms will be reset.
4) The LED indicating the current state of the home alarm.
5) The LED indicating that the device is up and running.
6) The LED indicating the current state of the main valve. It will turn on when the valve closes.
7) The power cable of the Arduino board.
8) The UTP cable plugged into the Ethernet shield attached to the Arduino board.
9) The bypass buttons of all three kinds of water leak detection algorithms.
10) The LEDs indicating whether or not a water leak has been detected.
11) The SD card, which works as storage for the HTML code as well as the settings of the device.
12) The cable connected to the home alarm.