4th Year IoT Project  
Aquarium Monitoring System: HCFF

Julien Rose, Achille Bayart, Nicolas Rigaudy



Table of contents

[Introduction and context 3](#_Toc52832527)

[Completed work 4](#_Toc52832528)

[Hardware 4](#_Toc52832529)

[Raspberry Pi 4](#_Toc52832530)

[Temperature 5](#_Toc52832531)

[Brightness 7](#_Toc52832532)

[Class hierarchy 8](#_Toc52832533)

[Automatic scheduling of data collection 8](#_Toc52832534)

[Image Recognition 9](#_Toc52832535)

[Initial Idea 9](#_Toc52832536)

[OpenCV 9](#_Toc52832537)

[Android App 11](#_Toc52832538)

[Figma 12](#_Toc52832539)

[Android Studio 13](#_Toc52832540)

[Project perspective 16](#_Toc52832541)

[Planned additions 16](#_Toc52832542)

[Challenges 16](#_Toc52832543)

[Conclusion 17](#_Toc52832544)

[GitHub Project Link 17](#_Toc52832545)

[Annexes 18](#_Toc52832546)

# Introduction and context

As a 4th year engineering project in the IoT major, we were first given a list of sample projects we could work on. However, the idea of an aquarium fish health monitoring system was already in our minds and we decided to go forward with our own project after getting approval from our teachers. Achille’s father is passionate about fish and owns a big reef tank with 8 fish from 6 different species and this was the perfect test subject for our system. Being ourselves passionate with new technologies and connected systems, we wanted to make a smart monitoring system that can be adapted and scaled very easily to different aquarium tanks, for professionals or individuals.

The first information we wanted the system to give the user was through image recognition of the fish. Using a camera, we wanted to be able to count the fish and differentiate the species so that we could show the user each fish separately to keep track on its appearance (a lot of health problems can have visual symptoms on the fish).  
We then decided to add sensors to our project such as temperature and brightness because factors like those are very important for fish and coral health.   
Finally, we chose to create an Android app especially for individuals, but also usable by professionals, which would allow the user to access in a smart, intuitive and entertaining way all of the data and information about his aquarium.

To bring this project to life, we had all the necessary tools at our disposal, including:

* Raspberry Pi nano computer
* Camera module
* Temperature and brightness sensors
* Programming languages and software that we had already learned in class to build the code for the app and for the sensors
* Test aquarium

# Completed work

## Hardware

### Raspberry Pi

The Raspberry Pi (RPi) is a nano computer.

Figure 1. Raspberry Pi 4 Model B

We decided to choose the Raspberry Pi 4 Model B because it is the latest model of Raspberry Pi and it fit perfectly with what we aimed to achieve in this project. We chose a model with 2GB of random-access memory as this was sufficient for our use.

We installed Raspbian (Raspberry Pi OS) which is a free operating system based on Debian, a Linux distribution.

To setup the RPi efficiently, we connected to it:

* **A USB type-C power supply** to provide power to the board.
* **A microSD card**: the nano computer needs an SD card to store all its files and the Raspbian operating system. We selected a microSD of 64 GB to be safe as we knew we would have to store video footage of the camera on the RPi.
* **A computer screen**: this is used to view the Raspbian desktop environment. We used a standard HDMI cable plus a micro-HDMI to HDMI adapter, to connect Raspberry Pi 4 to the screen.
* **A USB keyboard and a USB mouse** to be able to configure and work on the RPi.



Figure 2. B01 Camera Module

Moreover, we bought a B01 camera module especially for the Raspberry Pi. This camera is used to detect and count the fish in the aquarium tank, using OpenCV image recognition (see page 9).

At first, we tried using a Withings and a Samsung camera for the project as we already had them on hand but, being smart cameras with a lot of features and security, they were very hard to connect to the RPi. Achille’s father normally uses those two cameras to check on the aquarium when he is away but they therefore both already have dedicated apps and it was too complicated for us to integrate them into our prototype.  
We thus decided to use a camera module specially made for the Raspberry Pi, it’s useful and not very expensive.

### Temperature

The first information we wanted to measure was the aquarium’s temperature as it is very important for the fish’s healthcare, especially certain species that need a precise temperature-controlled environment.

Figure 3. DS18B20 temperature sensor

We decided to choose the DS18B20 sensor which is very commonly used in conjunction with the Raspberry Pi. This sensor has a measurement range from -55°C to +125°C and it is waterproof.

We started our tests with only one temperature sensor. The temperature sensor was quite easy to connect and read from as we used a preexisting Python library to control it. We also took the result of the temperature measures and saved them into a text file.

We then added another temperature sensor to the prototype breadboard, to get a more precise reading of the water temperature in the tank, with the two sensors spread out to opposite corners.

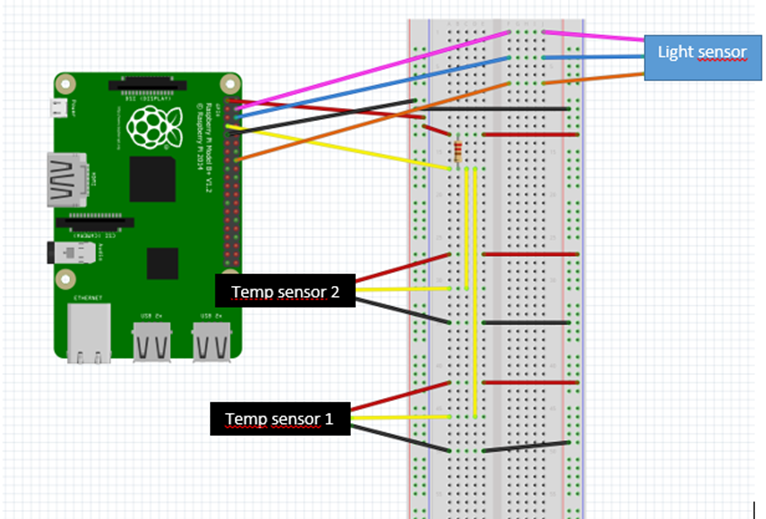


Figure 4. Prototype board wiring diagram (made using Fritzing)

We used a 4,7 kΩ resistor to connect the sensors and the cabling is done as follows:

* red wire to pin 1 (3,3V power)
* yellow wire to pin 7 (GPIO 4)
* black wire to pin 9 (Ground)

This wiring allowed us to measure the average temperature of the two sensors. We can see in the annex *(Figure 17)* the Python code used to save the average temperature to a text file.

The average temperature is saved every 30 minutes (1800 seconds). As you can see on the previous picture, we leave the sensors in the aquarium during one whole night. We have a maximum difference of 0,5°C and the temperature stays around 26,5°C. This is exactly what we expected.

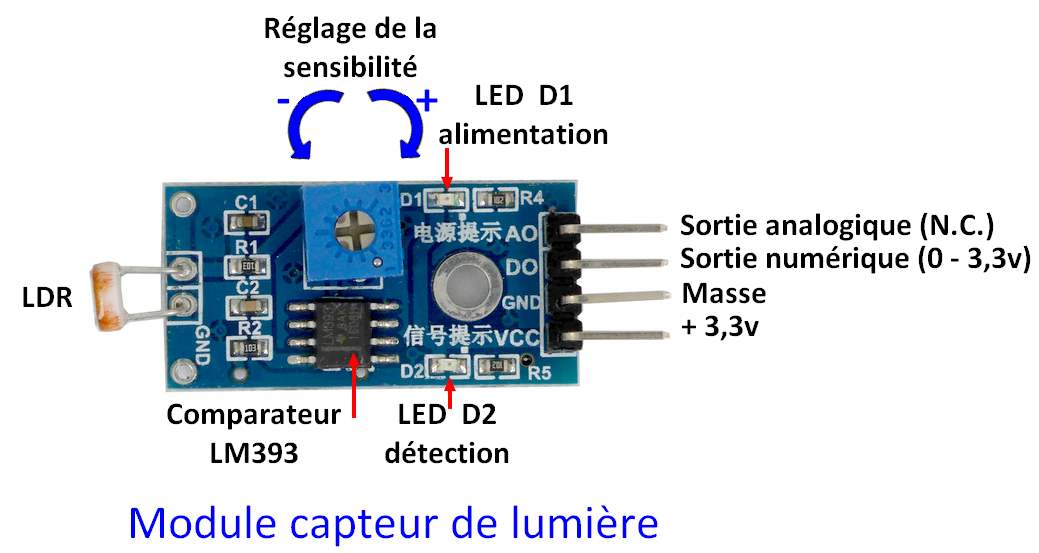


Figure 5. Full setup of Raspberry Pi with all sensors and camera module

### Brightness

The second information we wanted to get from our device was the brightness inside the aquarium, to be able to check if the tank lights are on or off. We decided to add another sensor to the Raspberry Pi: the LDR (Light Dependent Resistor) light detector module.

This sensor is used by setting a light threshold by adjusting the sensitivity of the LDR. The output of the module then depends on the threshold as explained here:

* Light intensity < threshold: output = 0
* Light intensity > threshold: output = 1

**Power**

**Detection**

**+5V**

**Ground**

**Digital output**

**Analog output**

**Sensitivity adjustment**

**Comparator LM393**

Figure 6. Light Sensor Module

As you can see in the wiring diagram (see page 5), this sensor is connected as follows:

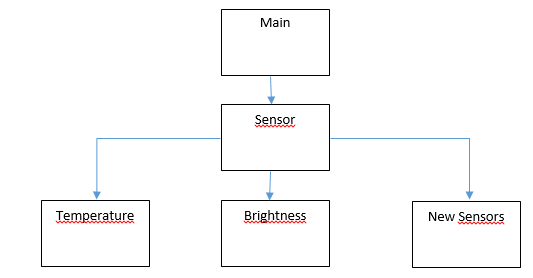
* pink wire to pin 4 (5V power)
* blue wire to pin 6 (Ground)
* orange wire to pin 16 (GPIO 23)

You can find the python code of the light sensor in the annex. *(Figure 18)*

A new line is printed when the brightness changes. When the light intensity is inferior to the fixed threshold, the code prints « it’s off » and when the light intensity is superior to the fixed threshold, the code prints « it’s on ».

### Class hierarchy

To improve the Python code and make it more organized and scalable for future additions of other sensors, we decided to create the following class hierarchy:



Thanks to this code architecture, we can add new captors easily.

You can find all the codes in the annex. *(Figures 19, 20, 21, 22, 23)*

### Automatic scheduling of data collection

To get continuous data from the sensors, we had to find a way to run our code at regular intervals, for example every minute. To do so, we used a software in Unix called Cron, which is a task scheduler. This allowed us to not need an infinite loop in our python code to run it forever, but instead use a lighter and safer process that would only run the code when necessary.

Cron uses a crontab (cron table) file, a configuration file that dictates which actions to run at which intervals. We thus entered our own cron command into this file, so that our python code, which retrieves and sends the data from the sensors to the database, is executed every minute, from the moment the RPi is booted up.

No description available.

Figure 7. Cron command to run main.py every minute

## Image Recognition

### Initial Idea

One feature we wanted to work on was image recognition. This was first intended as a way to count and detect the different fish and show them according to their species, also allowing to analyze in detail their behavior to alert of any signs that could warn the user of potential health problems.  
However, to be able to recognize the fish this accurately, we would have needed to use TensorFlow, a very detailed and thus quite complex machine learning platform and use a custom library of data (a large amount of images of the fish with boundary boxes and labelled according to their species). This path was very interesting but could have been a complete AI project, and we did not have the time or resources to go through with this idea in its entirety.

OpenCV  
We thus opted for simpler image recognition, using a Python library called OpenCV, which is a basic yet powerful too that can be used in a variety of ways with images and videos. It enabled us to detect fish movement in short videos and have an approximate count of the number of fish visible on camera during that video.

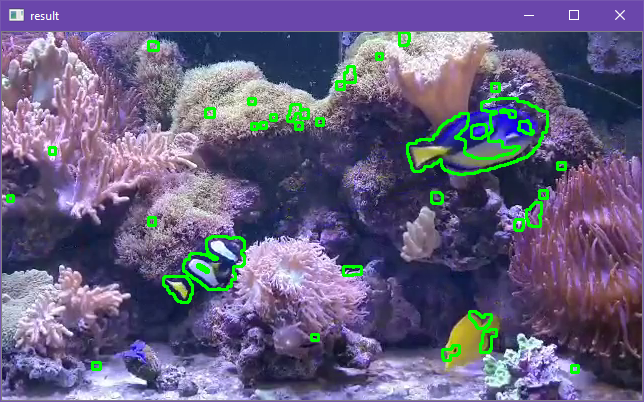


Figure 8. Detection of contours with movement

In this image, we can see the contours the program detects. This is done by comparing frame by frame a black and white version of the image. This image is also modified in various ways (blurs, thresholds, etc.) to accentuate contrast between shapes to identify the contours more easily. The program looks at contours that change position between two frames and displays them on screen.

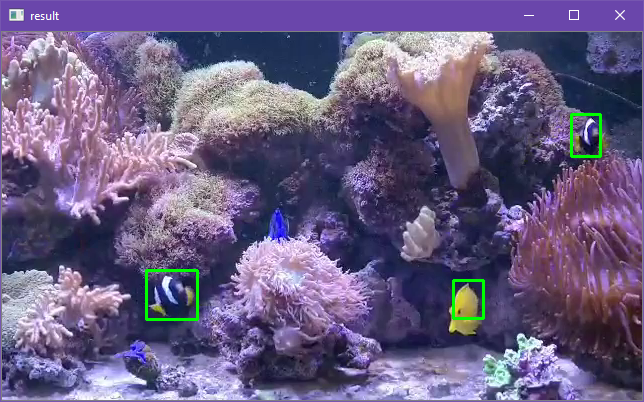


Figure 9. Drawing boxes from contours

In this second image, we can see that most of the small contours identified in the first step are not recognized as only the fish have boxes drawn around them. This is done by limiting the size of the contours that should get a boundary box to a minimum to eliminate most false positives coming from moving debris or from the anemones.

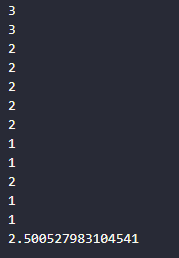


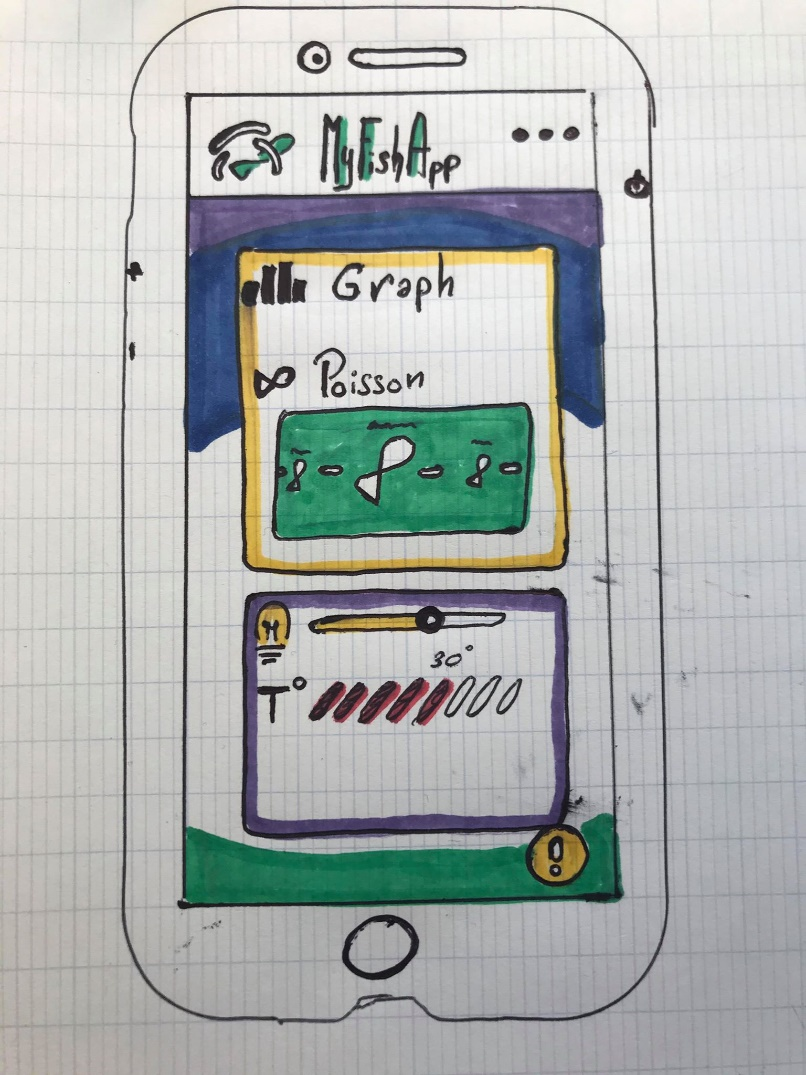
Figure 10. Number of fish for each frame and average for the video segment

The program can then count the number of boxes present on screen for each frame to have an approximate amount of fish visible by the camera. When the program is exited, an average is calculated of the number of fish present during the whole video segment.

## Android App

Our goal was to be able to check all the information of the aquarium that could affect fish health. We decided to use an Android application as our user interface.

The app needs to show the main elements of information, such as the different measures done by the sensors (temperature, light, etc.) and the last image of the fish. The interface must be designed in a smart way that is intuitive to use and as informative as possible for the user it.

To establish our vision of a “smart interface” we made this first drawing:

The main idea of our app is shown here, with, in the center, a fish information box, where you can choose which fish you want to see and that brings you to a page with more information on the fish.  
Below, there is the main information box, which displays all current sensor readings and brings to different pages of more detailed graphs for each sensor.

### Figma

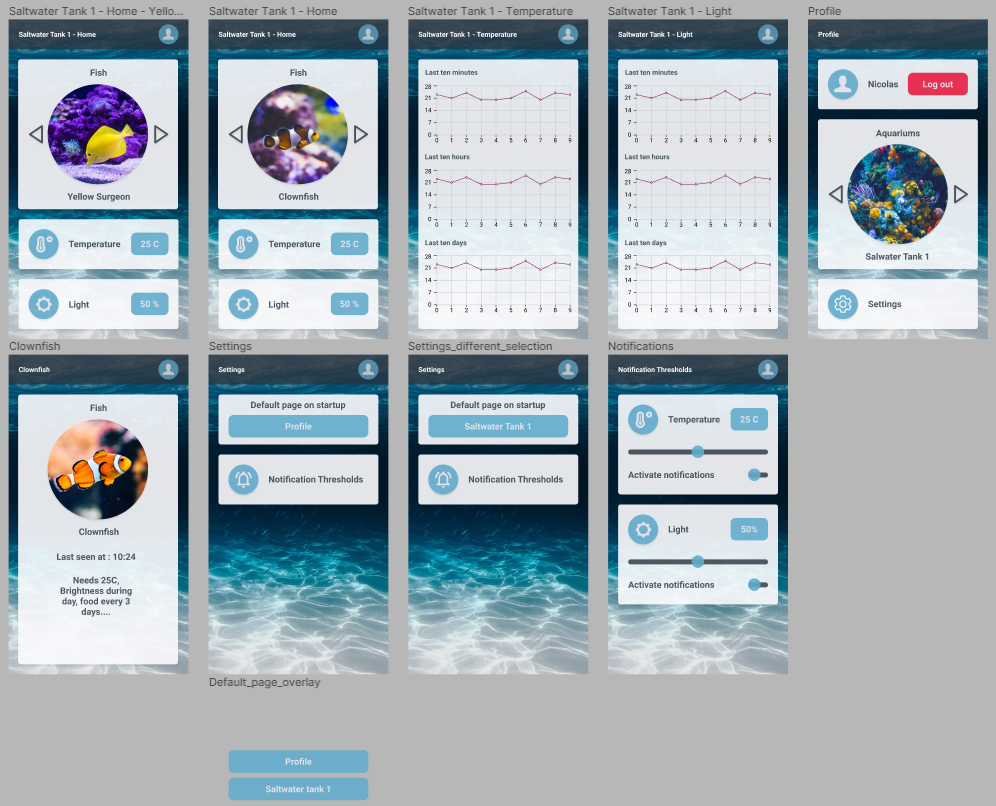
To design our interface, we first used a prototyping software called Figma. Figma helped us a lot to recreate our vision and drawing in a visual prototype. Figma is an interface design application, and thus also gives code for each element of the interface to use in Android Studio. This code was very helpful to understand how android interfaces work and when it is necessary to use smart interface tools.

Figure 11. Figma App Prototype

In this image, we can see the full prototype of the app, with a home page to select the aquarium tank we want to look at, as well as change the settings and also an option to log out of our profile. We can also see the different pages for individual fish, for the data graphs and for the different settings.

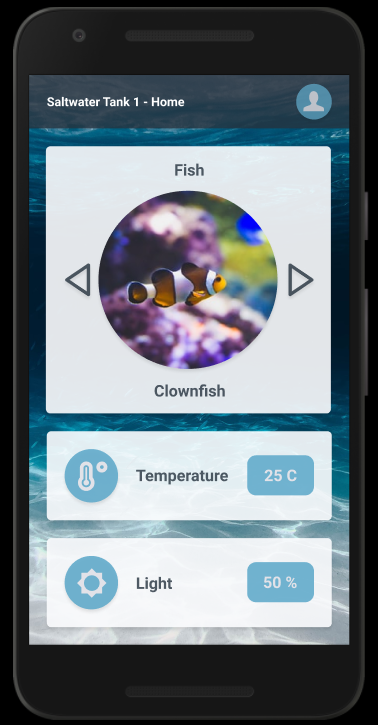


Figure 12. Prototype App on Phone

### Android Studio

We started our Android code with a REST API Android app base which we had created before during our Java Development for Android course. We then developed the interface of the app in parallel to this base model, following the Figma prototype as much as was possible with Android Studio tools.

Figma was a good tool for our code as it gave us Android code for each element, but it was also a trap for beginners like us. The code provided by Figma was difficult to piece together and adapt it to work as a whole and with the same result. It was timesaving to have this functionality but some of the elements did not work and some were just superfluous. This meant that we had to put a lot of effort to adapt this code into our code, and we quite quickly used less and less of the code it provided to us, using only the parts we knew would work.

For example, Android Studio does not have an easy, built-in option for creating circle images. We therefore had to use an external library called Circle Image View to create a circle image like in our Figma prototype.  
We also had issue when we superposed multiple layouts and boxes and some of our text disappeared.

Finally, we stopped using Figma code and used the information given to adapt our interface more correctly.

We deleted a lot of .xml files that Figma had given us which were unused, and we recreated our interface in a purer and more organized form, and this solved the layout and text bug.

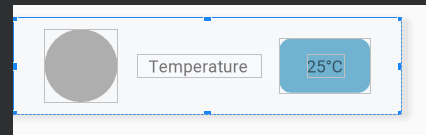
To manage the different information given by the sensors, we used a Card view layout that allowed us to have a template card with variables for the data name and value. The cards are then added to the interface with Java code and therefore it is very easy to add other sensors in the future, by just adding those sensors to a list and they will appear under the other cards.

Figure 13. Data Card Template

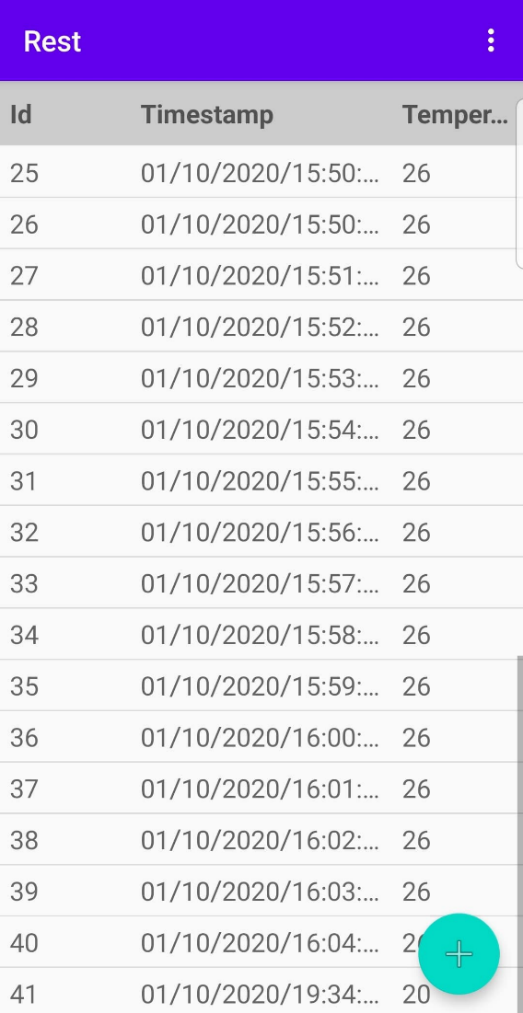
  
To help us test our connections, we used a TableVIew model which shows all the information of the database in a table.

Figure 14. List of data cards added with the Java code

Figure 15. Table showing data in the App

We can see here three parameters, the id, the timestamp (the date and time at which the data was collected) and the value of the temperature. We first used this interface to test our connection between the app and the database to check if the data was transferred correctly. After testing was done, we also used this table in the app itself. It is shown when you click on the different values shown in the main page. There is one table for the temperature data and one for the brightness data.

Once we had the link between the database and the app, we had the issue of refreshing the app, to get the new data we had to restart the app. We therefore put a refresh button which restarts the activity and then shows the new data.

The final step was to set up the fish information part:



Figure 16. Information page about Royal Gramma in the App

For each fish we added useful information like “Aquarium area” or the temperature at which each species thrive which helps us to see if everything is correctly configured for the species inside the aquarium.

# Project perspective

## Planned additions

Instead of tables with all the sensor data, we would prefer to create graphs which will represent the last data as is the case in our Figma prototype. This will allow a simpler and more intuitive reading of the data.

We would like to add new sensors to our project to monitor different measures that can be important to fish and coral health, such as water pH level, water level to check for leaks, etc.

This will be possible thanks to the class architecture, as it is very scalable and allows us to easily add more sensors to be controlled by the RPi.

We will want for next year to expand our Aquarium Monitoring System by putting an automatic food dispenser above the aquarium. This kind of system is very useful when the owner is absent during a period.

Finally, we could create another system for professionals. It would be a large number of HCFF systems which would all be connected together and to the same database host. This would cover a large number of basins. This could for example be useful in public aquariums to monitor all the tanks at the same time. It would transform our project from a simple connected device to a real full-fledged IoT system.

All the new features mentioned above would also be added to the Android application, as well as new elements, such as notification control and user connections.

## Challenges

One of the first major challenges we encountered was knowing where to set our limits in terms of project scope. At first, we had a lot of ideas of a more complex system starting with more sensors to get as much information as possible. We quickly realized that it was easier to start with only two different sensors but to build our project in a way that more sensors could be added very easily in the future instead of going straight ahead with a lot of different measurements.  
We also wanted to have a much more powerful image recognition section which could even go as far as differentiate fish species and maybe even detect fish health problems from, for example, the color of their scales. We saw that this would demand the conception of a whole AI system with a custom data library and we simply did not have the time or resources to take on such a task.

One challenge we found was the connection between the different elements of the project. On the Raspberry Pi side, the REST API client was giving us trouble as we it sent back many errors. We were also quite confused at first on how exactly all the tools at our disposal should be used and how to implement each one together.

Another major challenge we had to overcome this semester was of course due to the Coronavirus lockdown. Our semester started with virtual classes, so we were never able to work on the project together physically. This meant that Achille could be the only one to work on the hardware while Julien and Nicolas could only work on the software. In the end, we managed to share the work efficiently, but we are still impatient to be able to work together next semester when we finish this project.

# Conclusion

As a conclusion, we can say that it was a pleasure to work on this project together even during this unusual semester. This project was very complete as it made us work on hardware and software, designing and building the system from end to end. This therefore allowed us to use a lot of different skills and technical knowledge we learnt from our classes and from our own research.

We also got to use new tools and methods to enable working as a team even if we were not together. This included Git, which is a code versioning program that allowed us to keep track of the progress of our project and on all the different changes that were made within it. It kept us from having errors when combining different documents modified by each of the three of us.

We hope to be able to keep working on this project during our fifth year, as we still have many ideas on how to expand and improve our system, while making it smarter, more intuitive, and more entertaining.

# GitHub Project Link

<https://github.com/Nicolas-Rigaudy/Aquarium-Monitoring-System>

# Annexes

Figure 17. Python code that saves the average temperature in the "MoyTemperature\_NUIT" text file

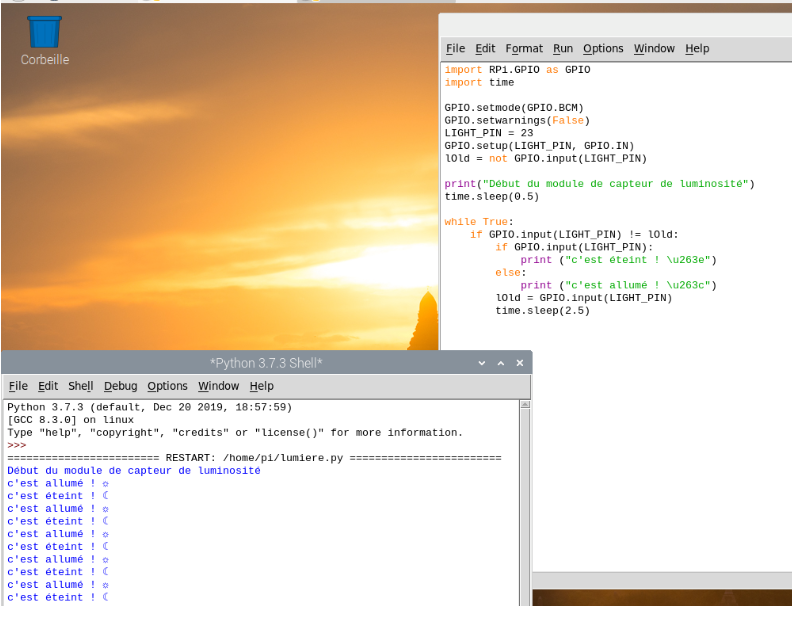


Figure 1. Brightness sensor Python code



Figure 19. Python code “main.py”



Figure 20. Python code “sensor.py”



Figure 21. Python code “tools.py”

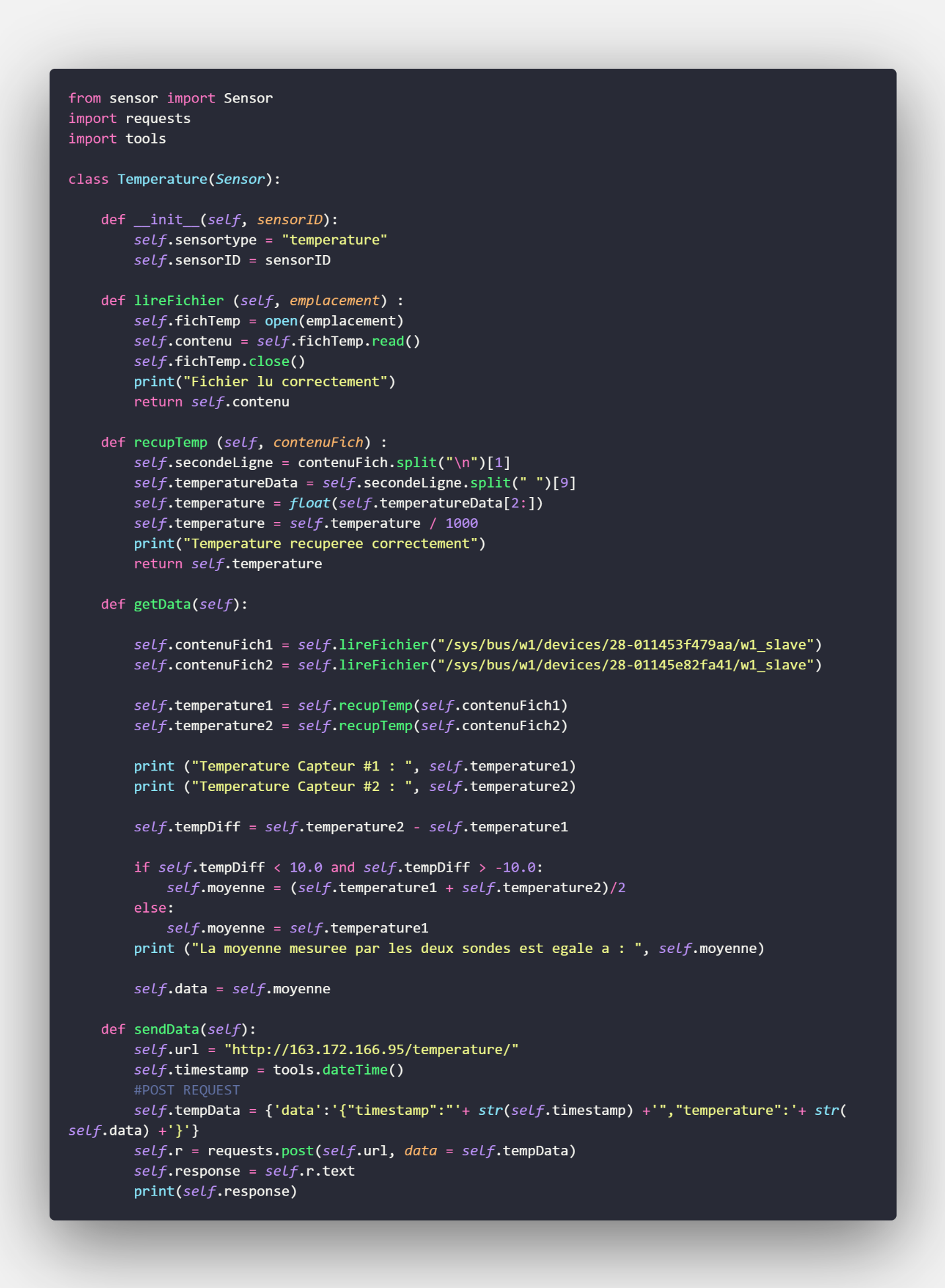


Figure 22. Python code “temperature.py”



Figure 23. Python code “brightness.py”