INSA Toulouse

Innovative Smart Systems

Smart Forest

INSA IOT INNOVATIVE PROJECT

Promo 2018-2019 *Tutors :* Mr. Groc, Mr. Monteil, Mr. Shea

Team : Erwan Béguin, Waël Ben Jemaa, Jean-Baptiste Laffosse, Alex Noize, Mathieu Raynaud

October 4th 2018 — January 23th 2019

Contents

Abstract **1**

1 Architecture of the project **2**

2 Hardware composition **3**

2.1 Gateway 3

2.1.1 Bill of materials 3

2.2 Weather stations 3

2.2.1 Bill of materials 3

3 Dimensioning batteries of the weather stations **4**

3.1 Energy consumption of the weather station 4

3.2 Measurement duration in a weather station 5

3.3 Capacity needed 5

3.4 Calculating battery life 5

3.5 Battery choice 6

3.6 Solar panels 6

3.6.1 Solar panel sizing 7

4 Weather Station Design **8**

4.1 Conception of the sensor shields 8

4.2 Development of the Arduino Program 9

4.2.1 Implementation of functions 9

4.2.2 Preparation of the payload 10

4.2.3 Automation of the Arduino program 11

4.3 Gateway 11

5 Dashboard **12**

5.1 Overview 12

5.2 Technical aspect 13

5.3 Possible improvements 15

6 Packaging **16**

7 Planning and calendar **17**

7.1 Provisional Gantt 17

7.2 Actual planning 18

Conclusion 20

List of Figures 21

List of Tables 22

Bibliography 23

Appendices

A. Gantt diagram of the project A1

B. Gateway js code B1

C. Angular dashboard of the project C1

D. GitHub repositories D1

Abstract

Negative effects on the environment are one of the greatest threats that human-beings are facing. To measure them, in Banyuls-sur-mer, in the south of France, the Oceanographic Observatory monitors the La Massane forest, which is a 336 hectare wide protected area. More than 50,000 trees and 8,000 different species have been identified in the forest, and many measurements such as temperature, humidity rate, and luminosity, are taken in this area to observe its evolution.

However, currently researchers collect measurements in the forest manually. The problem is that the area is not easily accessible, and it is very time-consuming to measure a few parameters in some places of the forest. Moreover, no network is available in the forest except at a sole point, and no energy supply is available either.

Smart Forest is a collaborative project between the Oceanographic Observatory of Banyuls-sur-mer, and the National Institute of Applied Sciences of Toulouse (INSA) engineering school. In this project, we implement open-source hardware and sensors to create weather stations, low-energy communication protocols like LoRa to communicate, and solar panels with batteries to provide enough energy to supply our system.

Our objective for this project is to create a smart network of sensors inside the forest to regularly obtain information about the forest. The data collected has to be easily exploitable by researchers directly from the research center.

To attain our objective, we deploy self-powered smart weather stations, communicating through LoRa with a gateway and finally send the data to a server, before displaying it on a dashboard. We design each weather station using an Arduino, a battery, solar panels, sensors, and a LoRa antenna. The gateway will collect LoRa data, translate it into CSV files, and send it over Wi-Fi. Finally, the dashboard will be implemented in Angular.

Chapter 1

Architecture of the project

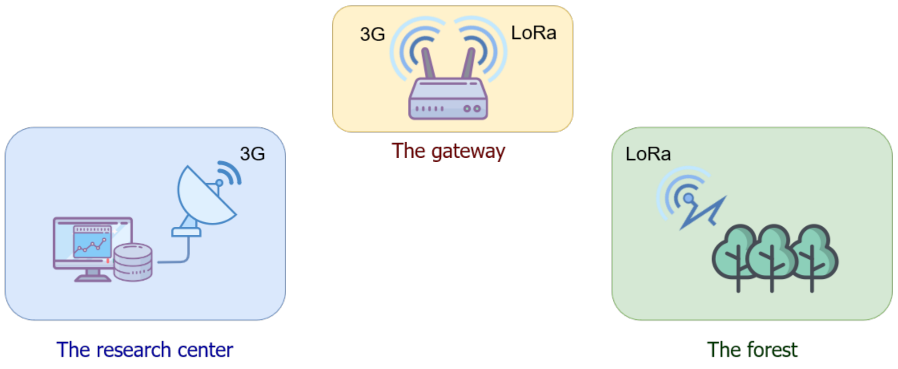


Figure 1.1: System overview

In the figure above, the architecture of the project is shown.

On the right, the forest contains weather stations. Each weather station implements several sensors, and sends data directly to the gateway. Each station is directly powered by a battery and a solar panel. It is important to know that each weather station is protected by a waterproof box, because they stay outside all year long, and have to resist to weather conditions.

In the middle, the gateway first collects all the data sent by the weather stations. This data is exchanged via a LoRa network. The gateway is located at the sole point of the forest where a connection can be established, on top of the La Massane tower. Once the data is received from the weather stations, it is computed, converted into a JSON file, and sent to The Things Network, which is a free cloud for LoRa devices on the Internet. The use of this cloud allows us to access the data easily from any point whenever we need it.

On the left, the research center has a dashboard capable of displaying graphically the data stored in The Things Network. It first downloads the data in JSON format, and read it to display the correct information. The dashboard is implemented in Angular 7, because it is a programming language which is executed on client side, and which is easily deployable on any server. Also, Angular is a raising language with a lot of support. This means that our dashboard will be sustainable for several years.

Chapter 2

Hardware composition

2.1 Gateway

2.1.1 Bill of materials

Name	Reference	Quantity	Price	Total cost
Raspberry Pi 3 B		1	35,36€	35,36€
LoRa Transceiver	RF-LORA-868-SO	1	18,62€	18,62€
TOTAL				35,36€

Table 2.1: Bill of materials of the gateway

2.2 Weather stations

2.2.1 Bill of materials

Name	Reference	Quantity	Price	Total cost
Arduino Micro		1	18,10€	18,10€
LoRa Transceiver	RN2483	1	14,70€	14,70€
GPS Module	ID: 746	1	34,22€	34,22€
Light sensor	TEMT6000X01	1	1,23€	1,23€
Temperature and humidity sensor	HIH8120-021-001	1	8,02€	8,02€
Digital gas sensor O3 module	968-042	1	65,58€	65,58€
TOTAL				141,85€

Table 2.2: Bill of materials of a weather station

We established the complete Bill of Materials (BOM) for the project at the beginning of the project. We based our decisions on the customer's needs, the compatibility with the Arduino Micro, and the energy consumption of the sensors. Then, we had to ask some advice from Mr. Monteil to choose the materials correctly. Finally, we had to decide which sensors we wanted according to the communications protocol they use (UART, I2C, etc ...), in matter of facts we have a limited number of pins on the Arduino Micro, and only one I2C interface.

Chapter 3

Dimensioning batteries of the weather stations

Since Smart Forest is an IoT based project, power supply is one of the most important aspects that must be dealt with. The goal of this part is to minimize the power consumption of our devices in order to have a longer battery life.

3.1 Energy consumption of the weather station

In order to calculate the total energy consumption of one weather station we began by exploring the power consumption of each of our components. The table below illustrates the current consumed by each component in both supply and sleep modes.

Component	**Supply current**	**Sleep mode current**
Arduino Micro	10 mA	62 µA
Temperature and Humidity Sensor	10 mA	0.6 µA
Light Sensor	3 mA	0.5 µA
O_3 Sensor	4 mA	0.05 mA
GPS Module	480 mA	260 mA
LoRa Transceiver	124.4 mA	2 mA
Total	622.05 mA	262.613 mA

Table 3.1: Consumption per component

As we can see, based on these calculations, the sleep mode current represents more than a third of the supply current which does not fit for an IoT project. This is due to high sleep mode current of the GPS module, so the solution is to completely shut this module off during sleep cycles. The solution that we proposed is to supply the GPS module with an external transistor, which will turn on the GPS module only in supply mode. If we turn the GPS module only once a day, this will help us to decrease the sleep mode current from 262.613 to 2.613 mA.

Another solution that we found suitable is to put the O_3 sensor on a supply mode all the time since it needs a lot of time to take a measurement (around 3 minutes).

Of course the supply current will decrease, but the sleep mode current will increase a little bit due to the O_3 sensor consumption, so finally we have:

- Supply current: 141 mA

- Sleep mode current: 6.563 mA

Now that we have the total consumption of the weather station, we have to determine the periods of supply-sleep we shall work with.

3.2 Measurement duration in a weather station

In order to determine the supply mode of a weather station we need to know duration of every measurement of each sensor, and add it to the transmitting duration of the LORA device.

$Supply\ duration = Measuring\ duration + Transmitting\ duration$

The table below illustrates this:

Component	**Measuring/Transmitting duration in seconds**
Temperature and Humidity Sensor	3
Light Sensor	3.5
GPS Module	60
LoRa Transceiver	5
Total	10

Table 3.2: Measuring and transmitting duration of each sensor

As we can see the total duration of the supply mode is 8.5 seconds. We will approximate this duration to 10 seconds in order to give more flexibility if an error occurs.

3.3 Capacity needed

Before calculating the capacity needed we have to establish the number of measurements each day. The research center requires 24 measurement per day, which means we will send the weather station into measurement mode once per hour.
For the GPS module we will take one measure per day.
The capacity per hour needed for the weather station can be calculated with the following formula:
$C = \frac{supply\_mode\_duration}{3600} * supply\_mode\_current + \frac{sleep\_mode\_duration}{3600} * sleep\_mode\_current$
We apply this formula to our values, and we obtain:
$C = \frac{10}{3600} * 141 + \frac{(3600-10)}{3600} * 6.563 = 6.563mAh$
The result is a total capacity per hour of **7.223 mAh**. This is of course after adding the consumption of the GPS module which is **0.66 mAh**.

3.4 Calculating battery life

The following formula will give us the exact battery life per hour for our weather station [6]:
$Battery\_life = (Battery\_capacity)/(Load\_Current) = result\ (hours)$
In order to choose a suitable battery for our weather station we have to look into the sun exposure data of the forest.

Since we choose to recharge our battery with a solar-panel system, we have to know approximately during which periods the sun is not available to charge a battery. This will help us to choose the correct battery to supply our components.

Based on some weather statistics [5], In Banyuls-sur-Mer during some periods we can have up to 8 days without sufficient sunshine to charge our batteries.

This means that our battery life must last up to 8 days which is equivalent to 192 hours.

Appliying this to the previous formula we obtain :
$Battery\_capacity = 192 * 7.223 = 1386.816 mAh$

Therefore we must choose a battery with a capacity equal or greater than 1500 mAh.

3.5 Battery choice

Now that we have the minimum capacity of the battery we can move on to the choice. The main properties of the battery dimensioning will be the cost and the voltage/current supplied.
Our choice is limited to lithium batteries with 3.7 V for each cell, due to the fact that this type of battery is the only suitable choice to provide more than 1500 mAh.
The table below indicates some batteries with their different specifications [2].

Battery	**Output voltage**	**Capacity**	**Cost**
TCB RC LiPo Battery 3S	11.1 V	3500 mAh	18.61 €
High Capacity 3.7 V	3.7 V	2200 mAh	4.56 €
2S RC LiPo Battery	7.4 V	5000 mAh	20.01 €

Our choice was immediately directed to the second battery because it suits all the specifications needed and costs less than the other two. For the nominal voltage (3.7 V) which is less than the nominal voltage used in the weather station (5V), we will fix this by combining two batteries in series to have 7.4 V.

Battery properties:

- Lithium-Ion battery
- Battery Capacity: 2200 mAh
- Voltage: 7.4 V 2S
- Size: 21 mm * 70 mm
- Weight: 120 g

3.6 Solar panels

Now, that we have finished sizing our batteries we need to know how we will charge them.
As we previously mentioned, the weather station can be supplied with the batteries for around 12-13 days (since we have a 2200 mAh battery).

Meanwhile, a solution for charging these batteries presents itself in order to have 100% autonomous system. The most efficient solution is solar panels, but the question is how to choose these solar panels.

3.6.1 Solar panel sizing

In our case there are two main things to consider in order to choose a solar panel [4]:

- How much energy can our battery store
- How much energy can a Solar panel generate over a period of time

This is of course, taking in to consideration the period of time our weather station can work autonomously without any charging (approximately 12 days).

We will begin by calculating the quantity of energy a solar panel can generate over a period of time:
In the south of France the average winter's day has 3 hours of sunshine [8] (we choose winter because it has the least sunshine in the year). In winter a 10 watt panel for example will provide 30w worth of energy back into our battery.

Now for the energy storage of a battery, taking in to consideration the most critical weather conditions(10 days without sun), which means that our battery will be at 20%, so if we need to charge a battery which needs a power of 16.28 watts. Our panel size can be obtained by applying the following equation:

$$Panelsize = \frac{Watts\ required}{Time\ of\ year\ sunshine\ hours} \tag{3.1}$$

Then, our panel size would be around 6 W panel (16.28/3).
Finally we can deduce that 6W/9V solar panel would fit our weather station perfectly even during the most critical periods of the year.

Chapter 4

Weather Station Design

In this section, we explain the design of the weather stations. Including all the steps we followed to measure the key physical variables and to send them through the LoRa network.

4.1 Conception of the sensor shields

When we ordered all the sensors for the weather stations, we have chosen low consumption sensors to be more efficient in terms of autonomy for the weather station. But this implies that two of the sensors required a shield to be used. For example, the light sensor uses the Surface-Mount Technology (SMT) and the Temperature/Humidity sensor is too small, so we cannot plug them directly into the Arduino Micro or on a breadboard.

Therefore, we designed two shields on KiCAD to use both these sensors (Figure 4.1). We tried to integrate them on a small shield, then we added pins we can plug on a breadboard, as this is why is it easier to use them. Finally, we followed the circuits shown on the datasheet to make our shields.

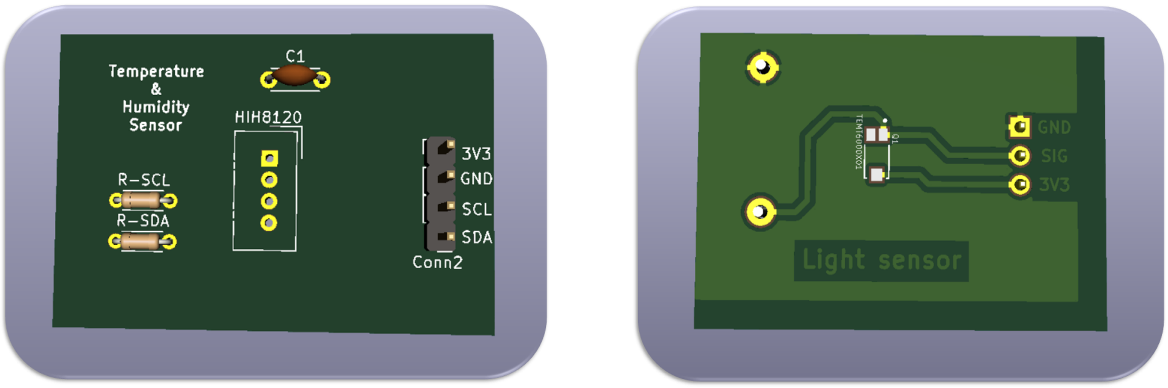


Figure 4.1: KiCAD visualization of the Temperature/Humidity sensor and the Light Sensor shields

We designed the two shields on KiCAD, and then we manufactured them with the help of Catherine Crouzet in the Physics Department. The results of our shield are shown on the following picture Figure 4.2.

After manufacturing them, and soldering all the components, we had to test the two shields to make sure they worked, and we satisfactory could use them later on the weather station.

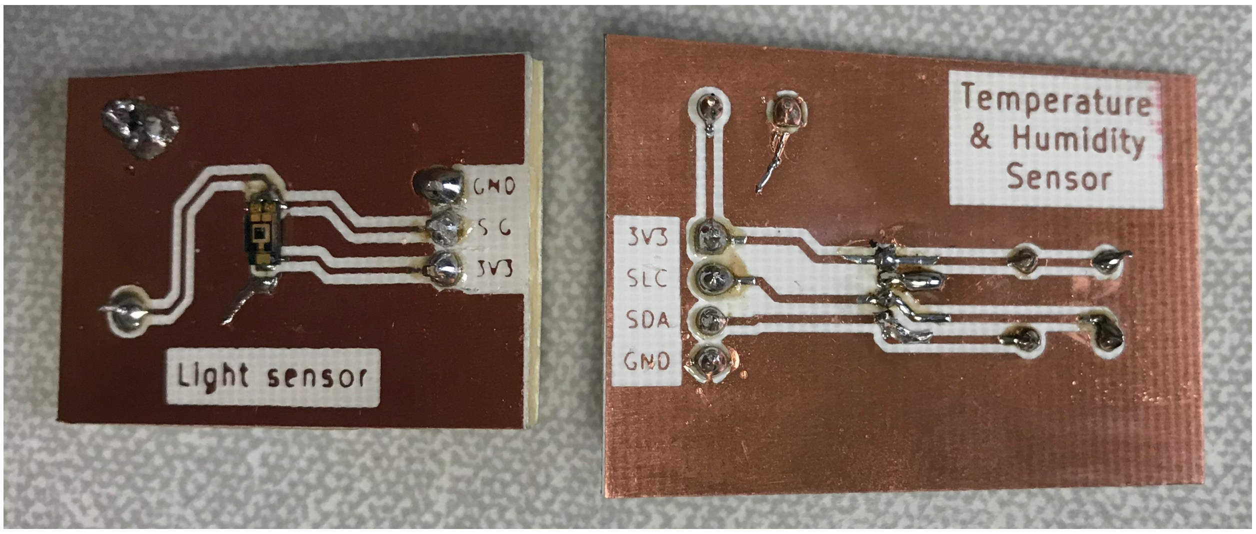


Figure 4.2: Temperature/Humidity sensor and the Light Sensor shields

4.2 Development of the Arduino Program

4.2.1 Implementation of functions

One by one we added all the functionalities specified by the customer. First, we tested the specific functions separately to obtain a sensor value, and then we added the function to the main program. We can find the following part in the Arduino program, and they can work independently:

- **Light sensor:** Measure brightness thanks to the Analog to Digital Converter (ADC) of the Arduino, and take the average over 100 values.
- **Temperature/Humidity sensor:** Open the I2C wire, request the value, read the humidity and the temperature value on 4 different bytes and close the communication.
- **Ozone sensor:** Open the software serial, ask for one measurement, wait for the response of a proper ozone value, parse the response of the sensor and close the software serial.

WARNING! the logical values of the sensor (RX and TX) and those of the Arduino are not the same, we need to add a tension divider bridge between the TX of the Arduino and the RX of the sensor.

- **GPS Module:** Turn on the GPS, initialize the GPS, wait until the GPS find the position, parse the response of the GPS into longitude, latitude and altitude variables, and shutdown the GPS module.
- **LoRa antenna:** Open the software serial, initialize the RN2483 chip, prepare the payload, send the sensor values or the position and close the software serial.

Everytime we used a software serial, we first need to open it and then to close the serial. Like this, we can avoid any conflicts between several software serials, and, we can use the GPS, the Ozone sensor and the LoRa antenna for the weather station.

Then, as we said before, to manage the battery lifetime we used a transistor to shutdown completely the GPS module when not in use. Moreover, because the GPS consumes too much energy, we decided to send the position of the weather station only once a day. We need to send the position of the weather station in order to verify the integrity of the sensor values sent throughout the day, for example people could steal the weather station, or animals could destroy it. We think that sending the position every 24 hours is enough for our application.

4.2.2 Preparation of the payload

The next part deals with the communication of the weather station with the gateway thanks to the LoRa antenna. Now all the sensor values are available, we can send them through the LoRa network. We can use a library to send the position or the sensor values and receive the information on The Things Networks (TTN). To do so, we need to prepare the payload we want to send.

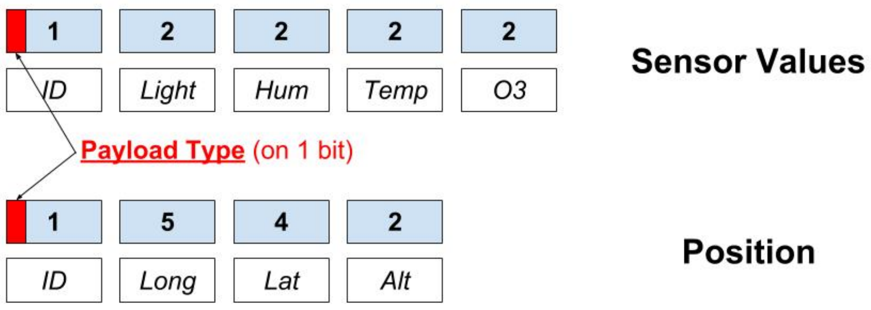


Figure 4.3: Sensor values and weather station position payloads
All the numbers correspond to the size of the part in byte.

- **Sensor value payload:** The Figure 4.3 shows the format of the sensor value payload. We have first the ID of the weather station, it is on one byte, then we have the light, the humidity, the temperature and the ozone values. Therefore the size of the sensor value payload is of ***9 bytes***. To make the difference between both payloads, we can also add a bit at the beginning, when it is a sensor value payload the bit is equal to **1**, and when it is a position payload the bit is equal to **0**.
- **Position payload:** The Figure 4.3 shows the format of the position payload. As for previous payload, we have the ID of the weather station and the type of the payload (**0**). Then we have the longitude, the latitude and the altitude. We can explain in detail the payload content, the 4.4 shows the longitude and the latitude parts.
  The longitude value is DDDMM.MMMM and the latitude DDMM.MMMM, with D the degrees, and M the minutes. Since the longitude is longer than the latitude, we need one more byte for the longitude. According to the directions, we can deduce the sign of the coordinates, for the latitude is North (+) or South (-), and for the longitude is East (+) or West (-). To indicate the sign, we use the first bit of the latitude and longitude parts. Finally, thanks to this format, we can have a position understood by the dashboard. The size of the position payload is ***12 bytes***.

Now the payloads are ready, we can send them through LoRa network using the library *<TheThingsNetwork.h>* and the RN2483 chip (available in the Physics Department).

After sending the message, we can see the payload on the TTN console. We will see in the following section how we parse and extract all the information from the payloads.

We decided to send the sensor values every hour, as we can save battery life using the sleep function of the Arduino between two transmissions. As we said before, we send the position every 24 hours.

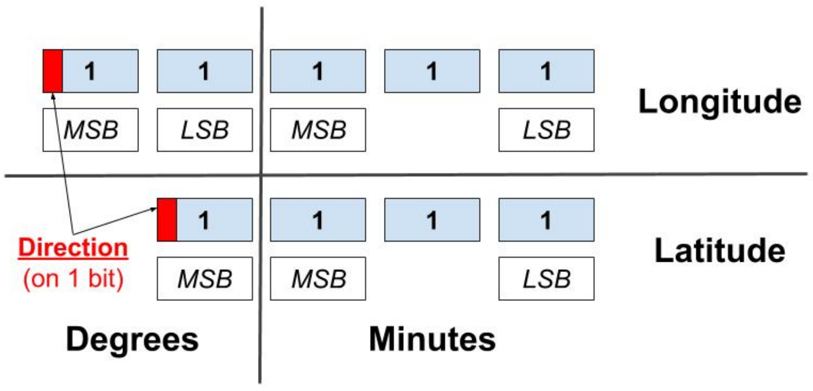


Figure 4.4: Latitude and Longitude of the GPS position
All the numbers correspond to the size of the part in byte.

4.2.3 Automation of the Arduino program

As an Internet of Things (IoT) application, the management of battery life is one of our main concerns. We need to modify the architecture of the program in order to be the more efficient in terms of energy consumption. To do this, we drew on the works Power-Down Sleep[3], and we used the libraries *<avr/sleep.h>*, *<avr/power.h>* and *<avr/wdt.h>*.

We know that the Arduino consumes power, even if we do not take any measurements, so we need to use different sleep modes in order to prevent any needless current consumption. This allows us to shutdown internal components such as internal clocks, ADC and oscillators and the Arduino can be woken up for example by an external interrupt or a watchdog timer. In fact, we use a watchdog in our program to wake the Arduino from power-down sleep.

We first need to configure the watchdog as a timer, the maximum period we can set is 8 seconds. Thus means if we want to shutdown the Arduino for a longer period of time, we need to use a counter to extend the sleeping time from 8 seconds to x*8 seconds. For example, if we want to send sensor values data every hour, we need **450***8 = 3600 seconds = 1 hour.

The Arduino is put into power-down sleep mode while sitting idle between measurements. Between two sleep modes, we need to increment the counter variable, and check if we need to measure the physical parameters.

We made two variables, one to send the measurements and another for the GPS position:

MAX_SLEEP_ITERATIONS and **MAX_POS_ITERATIONS**

4.3 Gateway

For the gateway, we used the one that is on the roof of the Electrical and Computer Sciences department and that is linked to The Things Network cloud. In order to decode the payload, we developed a javascript decoder that can be found in Appendix B, it basically decode the bytes of the LoRa payload and return a json object that can be used by the dashboard. Those json files are , then, stored in TTN cloud and can be accessed by a REST API.

Chapter 5

Dashboard

5.1 Overview

Let's start this section with a screenshot of the dashboard we produced for this project:

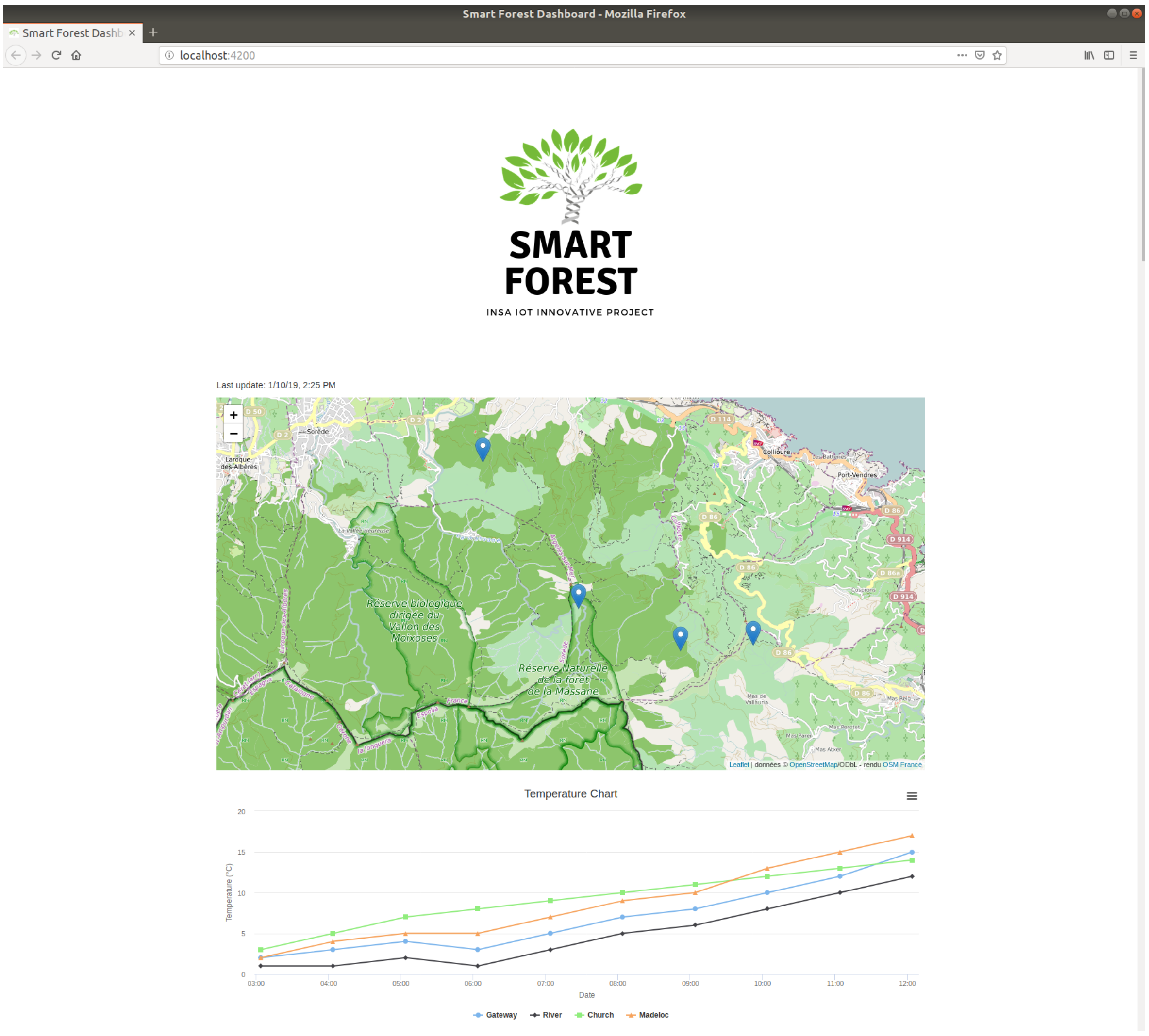


Figure 5.1: Dashboard of the Smart Forest project

This screenshot shows only a part of the dashboard, but a complete view of the dashboard we produced for the project can be found in the appendices.

On this dahsboard, we display a map centered on the La Massane forest with some pins. Each pins corresponds to a weather station. In the view above, there are four registered weather stations.

Below the map, we can find charts (one per monitored data). When the user clicks on a pin, the dashboard displays, on each chart, only the data linked to the weather station selected.

We are now going to see how this dashboard was developed.

5.2 Technical aspect

We developed the dashboard using Angular 7.

Several reasons led us to choose this programming language.
First, we had never worked with this language, and we thought that it was a great opportunity to increase our knowledge.
Next, Angular is a client-oriented programming language which allows us to develop an application rapidly, and this application can be easily deployed on a server. Consequently, we didn't have a lot of difficulties installing it on the research center server.

The Angular architecture includes a main application which contains several components. All these components can use functions defined in libraries and in services. A component implements an HTML template and a .component.ts file which contains the algorithms defined by the developer.

Our dashboard uses two components: one for the map and another for the charts. But both of these components display data relative to our weather stations. Consequently, we centralized the algorithms and the knowledge about the data inside a service which is called data.service.

Here is an overview of the architecture of our dashboard:

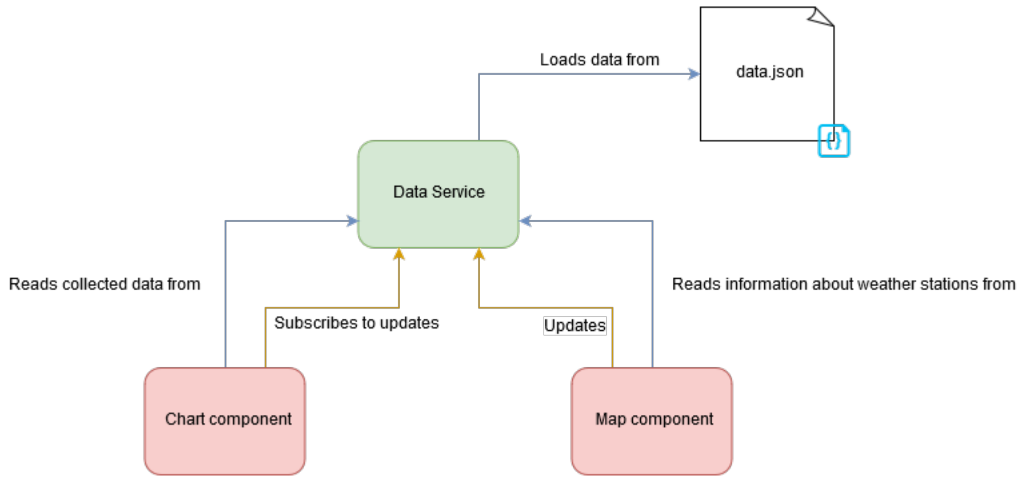


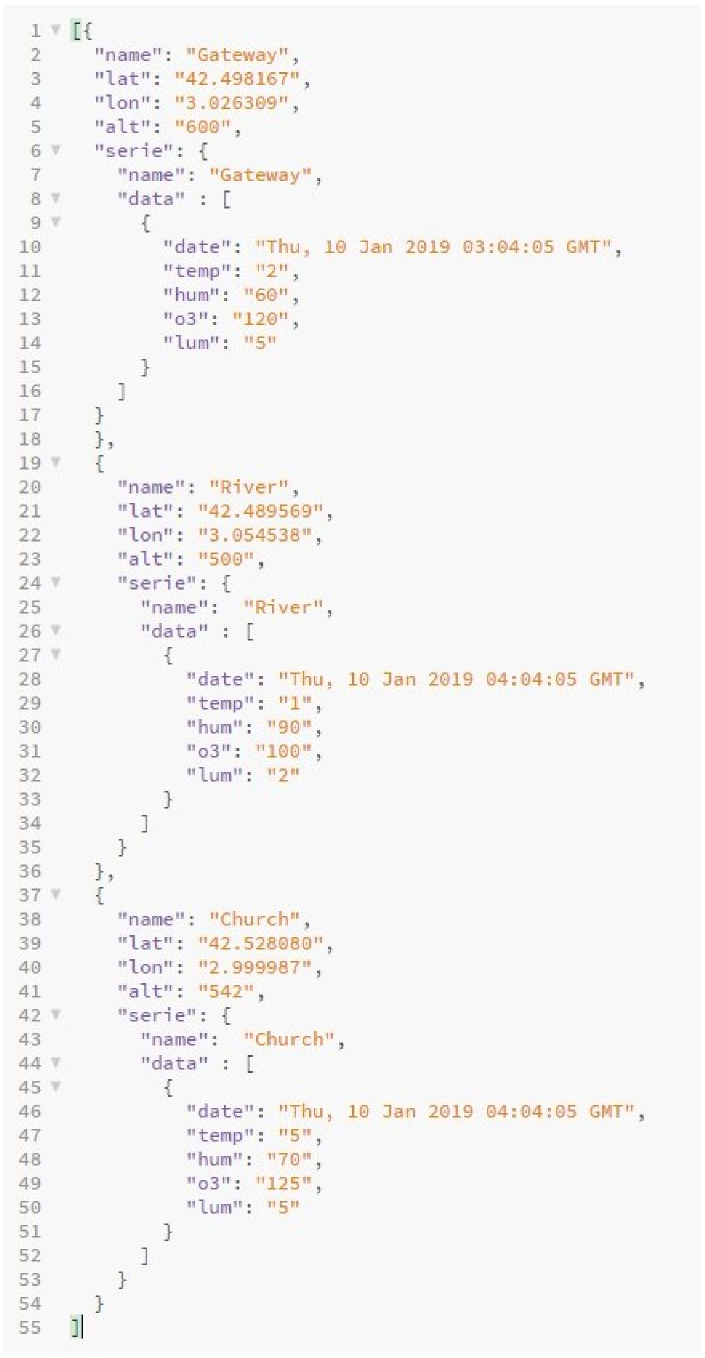
Figure 5.2: Architecture of the Angular components and services

At initialization, the data service loads the data from the data.json file. Then, the two components read the data collected inside the data service, and use the relevant data.

When an update comes from the map component (if the user clicks on the map for example), the data service variables are also updated, and so the chart component detects a change and updates too.

Concerning the data, it is stored in a data.json file, in the src/assets folder, and it contains all the collected data from each weather station.

Here is an example of the data.json file with three weather stations called "Gateway", "River" and "Church", and with a data for each measured factor at each weather station:



```
[{
  "name": "Gateway",
  "lat": "42.498167",
  "lon": "3.026309",
  "alt": "600",
  "serie": {
    "name": "Gateway",
    "data" : [
      {
        "date": "Thu, 10 Jan 2019 03:04:05 GMT",
        "temp": "2",
        "hum": "60",
        "o3": "120",
        "lum": "5"
      }
    ]
  }
  },
  {
    "name": "River",
    "lat": "42.489569",
    "lon": "3.054538",
    "alt": "500",
    "serie": {
      "name":  "River",
      "data" : [
        {
          "date": "Thu, 10 Jan 2019 04:04:05 GMT",
          "temp": "1",
          "hum": "90",
          "o3": "100",
          "lum": "2"
        }
      ]
    }
  },
  {
    "name": "Church",
    "lat": "42.528080",
    "lon": "2.999987",
    "alt": "542",
    "serie": {
      "name":  "Church",
      "data" : [
        {
          "date": "Thu, 10 Jan 2019 04:04:05 GMT",
          "temp": "5",
          "hum": "70",
          "o3": "125",
          "lum": "5"
        }
      ]
    }
  }
]
```

Figure 5.3: An example of the file data.json

In the file above, we can see that the weather station called "Gateway" is located at GPS coordinates (42.498167, 3.026309), and a measurement of data was done on Thursday, 10 January 2019 at 03:04:05. The temperature was 2°C, the humidity rate was 60%, the quantity of O3 was 120 ppb (parts per billion), and the luminosity was 5%.

To update the data.json file, we decided to implement a small node.js server as a back-end server. This decision was made because Angular is client oriented and not made to receive up-link requests to update a database.

After setting up the server, we set up a subscription to the The Things Network database where our gateway send its data. When a new data is send to the database, we could receive it on our back-end server and update the data.json file. Therefore, the data displayed on the interface is automatically displayed.

5.3 Possible improvements

The data storage has to be improved for the next version of the system. In fact, currently the data storage is entirely done in the data.json file. But in the future, when we will monitor a great amount of data, it would be better to store it in a database. It will be the case if we monitor data over few years.

The administration of the dashboard could be improved by adding an admin page where we could add a new weather station, or modify data when a problem occurred, and some other useful functionalities.

Chapter 6

Packaging

After finishing the hardware part, the challenge was to design a package for the weather station. In fact, the package should be solid and have a nice design from an aesthetic point of view.
We chose to produce our package with a 3D printer based on PLA material. The design was made with SolidWorks software, from which we generated an STL file compatible with the 3D printer.

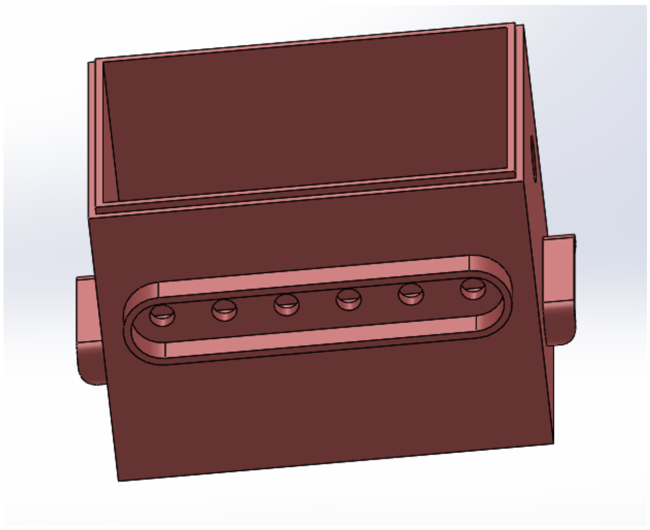


Figure 6.1: Opened package of a weather station

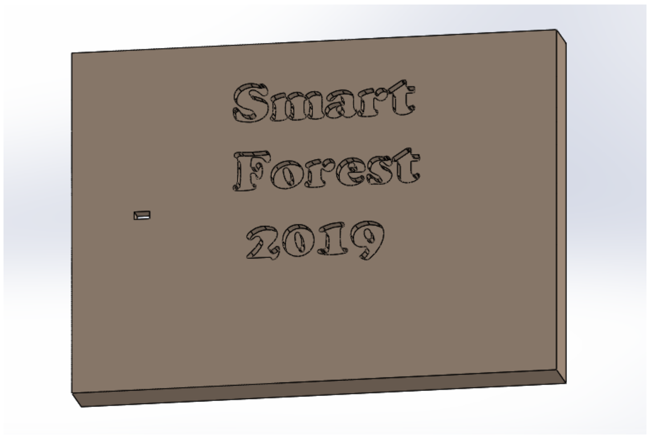


Figure 6.2: Package cover of a weather station

Chapter 7

Planning and calendar

In this section, you may find information about our organization for all the management of the Smart Forest project.

7.1 Provisional Gantt

Our provisional Gantt can be seen in the appendices (appendix A).

First, we planned to start with the development of the weather stations. We wanted to obtain working hardware devices before starting to implement other functionalities. According to us, the weather stations are the smart devices around which all the project has to adapt.

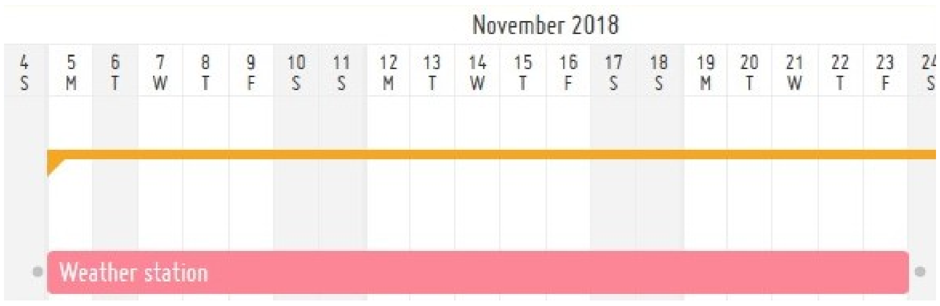


Figure 7.1: Gantt diagram for the weather stations

We also planned to develop the gateway and the weather stations at the same time because the gateway embeds a weather station. However, the end of the development for the gateway was later than the one of the weather stations, because we also had to develop all the services for data collection and data transmission.

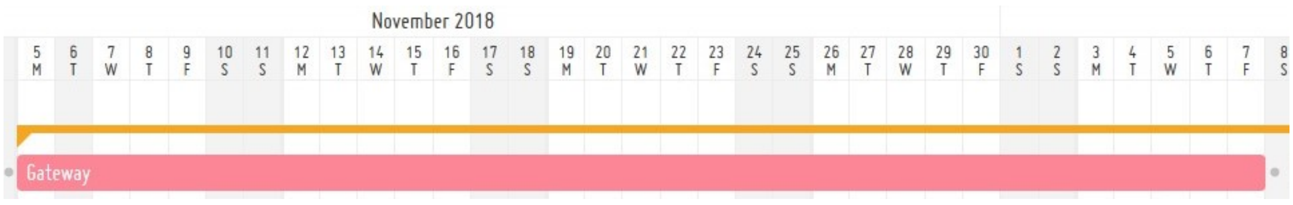


Figure 7.2: Gantt diagram for the gateway

After that, we wanted to add the batteries and the solar panels to the weather stations, to be able to provide self-supplied devices.

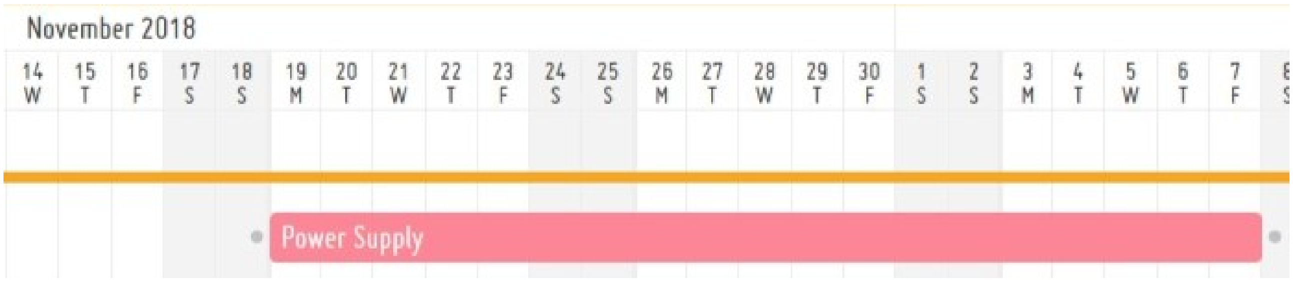


Figure 7.3: Gantt diagram for the power supply

We finally planned to implement the data collection on research center side, and to develop the dashboard to see all the collected data.

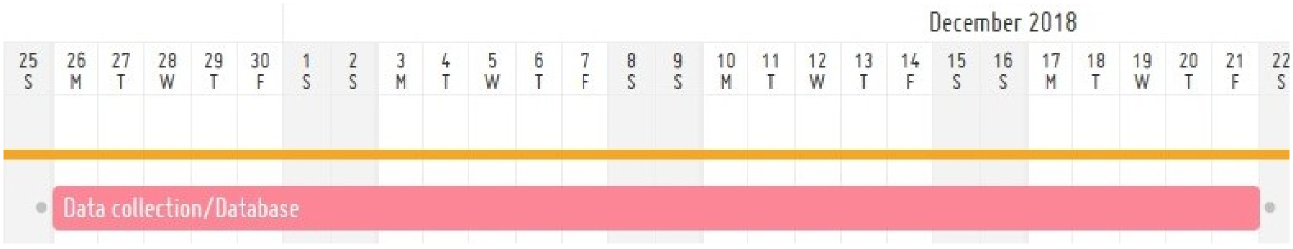


Figure 7.4: Gantt diagram for the data collection

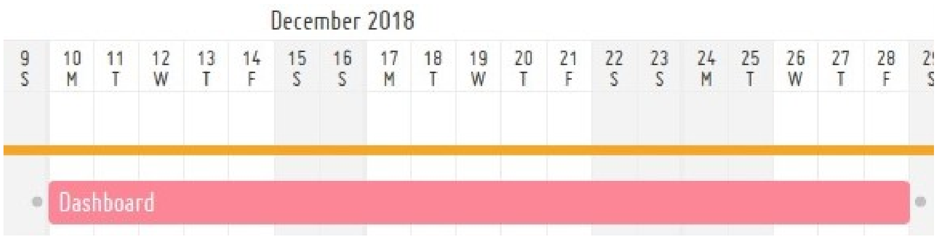


Figure 7.5: Gantt diagram for the dashboard

In this Gantt, we planned to finish the development of the project at the end of December 2018. But the reality is completely different.

7.2 Actual planning

First, we rapidly faced issues with the weather stations, and their development took so much more than the 3 weeks expected. Actually, the development of the weather stations finished on January 2019.

As a consequence, we anticipated the development of the dashboard. Instead of waiting the beginning of December 2018, we started to develop the dashboard on November 2018, only one week after the start of the development of the weather stations. This was a really good decision because we finished to develop the dashboard on January 2019.

Then, we started to develop the gateway later than the expected date. Actually, we started to develop it on December 2018. But the development was definitely faster than we expected, and took only one week. In fact, we made some choices, like the use of The Things Network, which helped us in this development.

Talking about the power supply, we faced some communication and exchanges issues with our tutors, which did not allow us to retrieve the required equipment from Banyuls. As a consequence, we sized the batteries, and we prepared the theory for the use of solar panels to supply our devices, but we did not have the chance to test.

Finally, we started to work on the data collection on January 2019. We succeeded producing a NodeJS server able to retrieve automatically data from The Things Network after two weeks of work.

To conclude, the development of the project took ten weeks, which is two weeks more than expected, but the final deadline was respected.

Conclusion

During this project, the oceanographic research center of Banyuls-sur-mer asked us to deploy a network of smart sensors to monitor the weather parameters across the La Massane forest in Banyuls-sur-mer.

We faced a variety of challenges from hardware related decisions to interface configurations.
Hardware wise, we had the opportunity to make ourselves the design choices related to the architecture, the energy consumption and the casing to answers the research center's needs.
As an Open-Source technology, we chose LoRa for the communication part of the network. We were able to design how the sensors communicate with the gateway and configure the gateway to talk with our web interface.
We learned a lot about the LoRa protocol and how to design a device for this technology.

This project also brought us a lot of knowledge on power supplying devices. We designed the consumption of our devices to be able to choose the relevant batteries and solar panels for our smart weather stations.

We deployed an user interface using Angular to display the data collected from the weather stations. This data is stored in a JSON file. The dashboard is perfectly working, and can be used by the researchers. This project allowed us to discover and to work with Angular 7, which was totally new and enriching for us.

In the future, some improvements can be bring to the project to improve it.
First, some sensors, like a rain gauge, a dendrometer, or a water level sensor, can be added to the weather stations to increase the quantity of parameters monitored in the forest.
Then, the possibility to add new weather stations to the network was took into consideration. In fact, we designed our identifiers to work with 128 different weather stations. Also, our dashboard can display all the weather stations it reads inside the data file, so we do not have any restriction on the number of weather stations on dashboard side.

Finally, the next step, and the final one for our project, is to deploy our network directly in the La Massane forest and in the research center of Banyuls. This could be done quickly.

List of Figures

1.1 System overview 2

4.1 KiCAD visualization of the Temperature/Humidity sensor and the Light Sensor shields 8
4.2 Temperature/Humidity sensor and the Light Sensor shields 9
4.3 Sensor values and weather station position payloads 10
4.4 Latitude and Longitude of the GPS position 11

5.1 Dashboard of the Smart Forest project 12
5.2 Architecture of the Angular components and services 13
5.3 An example of the file data.json 14

6.1 Opened package of a weather station 16
6.2 Package cover of a weather station 16

7.1 Gantt diagram for the weather stations 17
7.2 Gantt diagram for the gateway 17
7.3 Gantt diagram for the power supply 18
7.4 Gantt diagram for the data collection 18
7.5 Gantt diagram for the dashboard 18

List of Tables

2.1 Bill of materials of the gateway 3
2.2 Bill of materials of a weather station 3

3.1 Consumption per component 4
3.2 Measuring and transmitting duration of each sensor 5

Bibliography

[1] Alibaba.com. High capacity 21700 5000mah 3.7v li ion battery cheapest 20700 21700 lithium battery on alibaba.com. https://www.alibaba.com/product-detail/High-Capacity-21700-5000MAH-3-7V_60747061006.html?spm=a2700.md_fr_FR.debelsubf.1.68d57a7c8T7OKr.

[2] AliExpress. Aliexpress search 3500mah 7.4v battery. https://www.aliexpress.com/w/wholesale-3500mah-7.4v-battery.html.

[3] Tony DiCola. Example 2: Power-down sleep. https://learn.adafruit.com/low-power-wifi-datalogging/power-down-sleep.

[4] Solar Technology International. Calculating your solar panel requirements. https://www.solartechnology.co.uk/support-centre/calculating-your-solar-requirments.

[5] Annuaire Mairie. Ensoleillement et température à banyuls-sur-mer. https://www.annuaire-mairie.fr/ensoleillement-banyuls-sur-mer.html, December 2018.

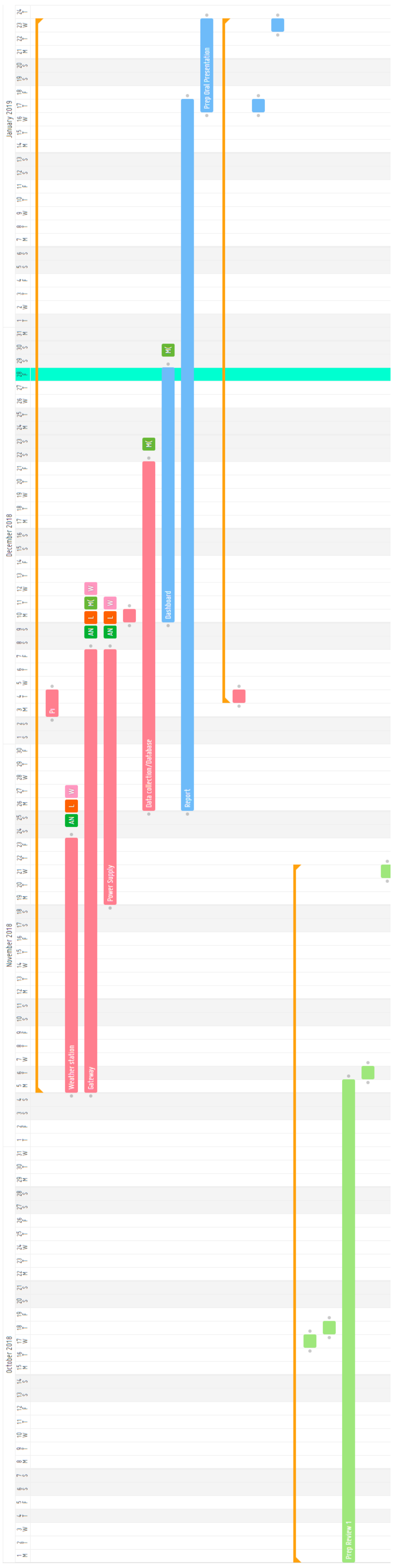
[6] StackExchange. Calculate battery life. https://electronics.stackexchange.com/questions/340741/calculate-battery-life.

[7] Northern Arizona Wind & Sun. Solar charge controller basics. https://www.climatestotravel.com/climate/france.

[8] Climates To Travel. France climate: average weather, temperature, precipitation, best time. https://www.climatestotravel.com/climate/france.

Annexes

A. Gantt diagram of the project

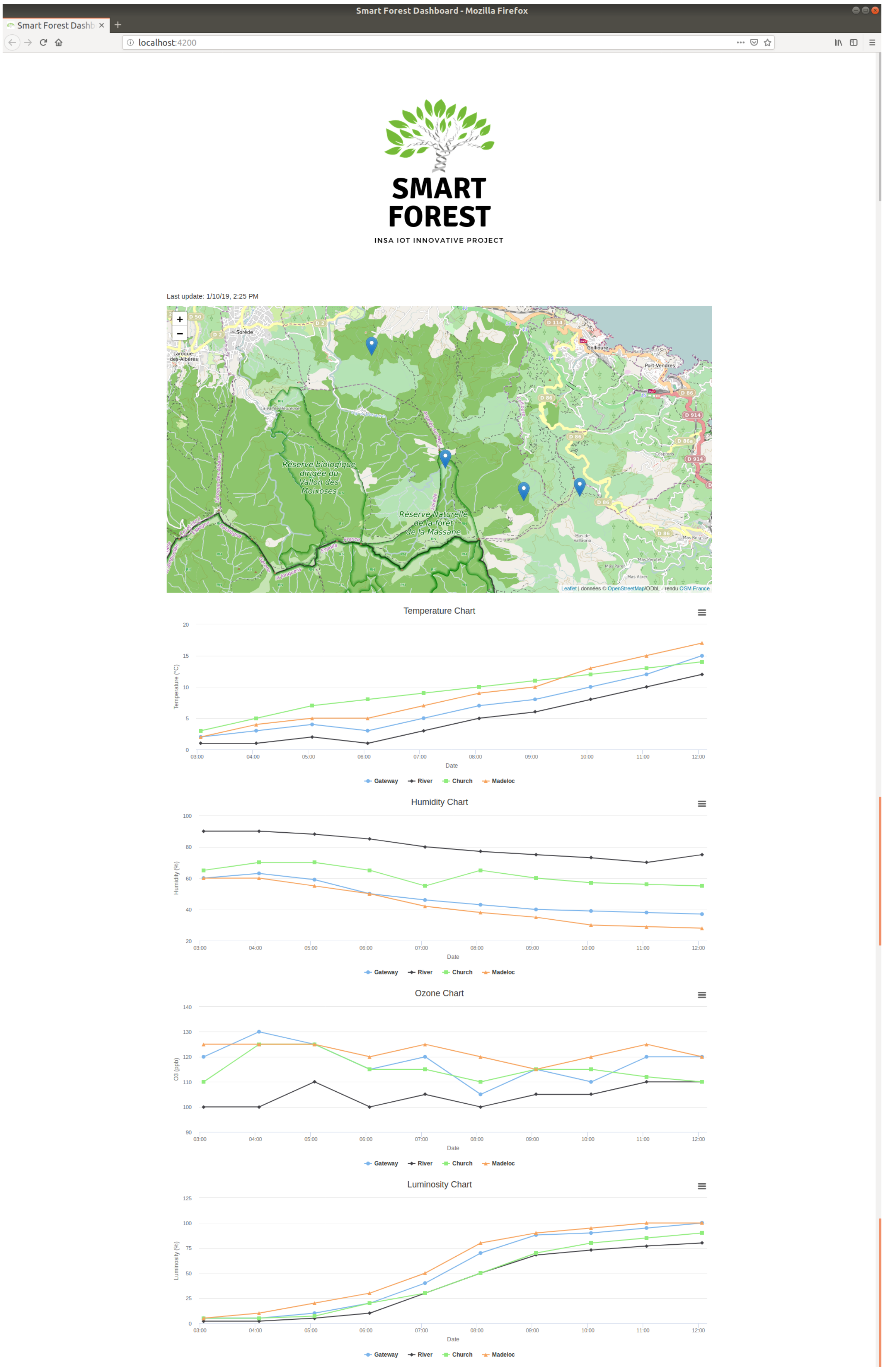


B. Gateway js code

```
function Decoder(bytes, port) {
  // Decode an uplink message from a buffer
  // (array) of bytes to an object of fields .
  var response = {};

  // if (port === 1) decoded.led = bytes [0];

  response.type_id = bytes [0] & 0x80 ;
  response.ws_id = bytes [0] & 0x7F;

  if (response.type_id == 0x00){
    response.type = "GPS Payload"
  }
  else if (response.type_id == 0x80){
    response.type = "Sensors Payload"
  }

  switch (response.ws_id) {
    case 1: response.ws_id == 0x01;
        response.ws_name = "Church";
        break;
    case 2: response.ws_id == 0x02;
        response.ws_name = "Gateway";
        break;
    case 3: response.ws_id == 0x03;
        response.ws_name = "River";
        break;
    case 4: response.ws_id == 0x04;
        response.ws_name = "Madeloc";
        break;
    default :
        response.ws_name = "Unknown";
        break;
  }


  if(response.type_id == 0x80) { // If it is a sensors data payload
    response.light_raw = (bytes [1] << 8 | bytes [2]) ;
    response. light = response. light_raw *(5.0/1023.) /(10E3*5E−7)
    response.HR_raw = (bytes[3] << 8 | bytes [4]) ;
    response.temp_raw = (bytes [5] << 8 | bytes [6]) >> 2;
    response.HR = (response.HR_raw) / (16384 − 1) * 100;
    response.temp = ((response.temp_raw)/(16384−1))*165 − 40;
    response.O3 = (bytes [7] << 8 | bytes [8]) ;
  }

  if(response.type_id == 0x00) { // If it is a GPS data payload

     // Longitude
    if ((bytes [1] & 0x80) == 0x80){
```

```
    response.long_o = 'E'
  }
  else if ((bytes[1] & 0x80) == 0x00) {
    response.long_o = 'W'
  }
  response.long_deg = (bytes[1] & 0x7F) << 8 | bytes[2];
  response.long_min = (bytes[3] << 16 | bytes[4] << 8 | bytes[5])



  /*response.long_min = (String)(response.long_min);
  response.long_min = response.long_min.split ('') ;
  response.long_min.splice(response.long_min.length -4,0,'.') ;
  response.long_min= response.long_min.join ('') ;
  response.long_min=parseFloat(response.long_min);*/

  // Latitude
  if ((bytes[6] & 0x80) == 0x80){
    response.lat_o = 'N';
  }
  else if ((bytes[6] & 0x00) == 0x00) {
    response.lat_o = 'S';
  }
  response.lat_deg = (bytes[6] & 0x7F);

  response.lat_min = (bytes[7]<<16|bytes[8]<<8|bytes[9]) ;
  /*response.lat_min = (String)(response.lat_min);
  response.lat_min = response.lat_min.split ('') ;
  response.lat_min.splice(response.lat_min.length -4,0,'.') ;
  response.lat_min= response.lat_min.join ('') ;
  response.lat_min=parseFloat(response.lat_min);*/

  // Altitude
  response.altitude = (bytes[10]<<8 | bytes[11]);
  }

  var utc = new Date().toJSON();
  utc= utc.substring (0, utc.length-1);

  response.date = utc;

  var formated = {};

  formated.name = response.ws_name;

    response.lat_min_real = ((response.lat_min /10000./60.) *10000);
    response.lat_min_real = Math.round(response.lat_min_real);
    response.long_min_real = ((response.long_min /10000./60.) *10000);
    response.long_min_real = Math.round(response.long_min_real);

  if (response.type == "GPS Payload"){
   if(response.lat_o == 'N'){
      formated.lat = '+' + (String)(response.lat_deg) +'.'+ (String)(response.
          lat_min_real);
    }
   else if(response.lat_o == 'S'){
     formated.lat = '—' + (String)(response.lat_deg) +'.'+ (String)(response.lat_min_real
         );
   }
   else {
     formated.lat = "Wrong format";
   }
   if(response.long_o == 'E'){
      formated.long = '+' + (String)(response.long_deg) +'.'+ (String)(response.
          long_min_real);
    }
   else if(response.lat_o == 'W'){
      formated.long = '—' + (String)(response.long_deg) +'.'+ (String)(response.
          long_min_real);
    }
    else {
     formated.long = "Wrong format";
    }
    formated.alt = (String)(response.altitude);
  }
  else if (response.type == "Sensors Payload"){
    formated.data = {};
    formated.data.date = response.date;
    formated.data.temp = (String)(response.temp);
    formated.data.hum = (String)(response.HR);
    formated.data.o3 = (String)(response.O3);
    formated.data.lum = (String)(response.light)
  }

  formated.type = response.type;

  //return response;
  return formated;



}
```

C. Angular dashboard of the project



D. GitHub repositories

Arduino code repository:
`https://github.com/MathieuRaynaud/Smart-Forest-hardware.git`

Dashboard repository:
`https://github.com/MathieuRaynaud/Smart-Forest-software.git`

Server repository:
`https://github.com/MathieuRaynaud/nodejs_server_rest.git`

Packaging and shield repository:
`https://github.com/Jblaffosse/Smart-Forest-packaging.git`