SOFTWARE REQUIREMENTS SPECIFICATION

for

Meteorological Station with LoRa-WAN

Version 1.0

Prepared by: Vitor Engers (16105314)

Submitted to: Giovani Gracioli

September 20, 2021

Contents

1 Introduction **3**
- 1.1 Purpose . . . 3
- 1.2 Intended Audience . . . 3
- 1.3 Intended Use . . . 3
- 1.4 Project Scope . . . 3

2 Overall Description **6**
- 2.1 User Needs . . . 6
- 2.2 Assumptions and Dependencies . . . 6

3 System Features and Requirements **7**
- 3.1 Function and Requirements . . . 7
  - 3.1.1 Software Dependencies . . . 7
  - 3.1.2 Hardware Dependencies . . . 7
- 3.2 External Interface Requirements . . . 7
- 3.3 System Features . . . 7
- 3.4 Nonfunctional Requirements . . . 7

4 Diagrams **8**
- 4.1 User Class Diagram . . . 8
- 4.2 State Machine Diagram . . . 10

5 Development and results **11**

6 Conclusion **18**

7 Future Development **19**

1 Introduction

1.1 Purpose

In developed countries, the production of commodities such as soybeans and corn, it is very common to use more modern technologies for planting, monitoring and crop care. Despite constant evolution in the precision agriculture sector around the world, in Brazil most of this branch is done in an archaic way. Monitoring weather conditions and understanding how these factors can impact plant growth, disease development and plant health is essential to increase crop productivity.

Most of the diseases that can devastate a crop and reduce its productivity by up to 60% are caused by pathogens that need a specific environment to proliferate, such as a specific amount of air humidity and heat. However, these factors can vary from one plot of land to another and this distance makes monitoring costly and difficult to implement.

To circumvent the distance between the land plots and the power supply, Sub-GHz radio frequency communication with the LoRa-WAN protocol will be used.

As a result, this project intends to acquire data from a climate monitoring station and transmit the data through a Sub-GHz radio frequency network using the LoRa-WAN protocol. All reports will be saved on a server for possible technical analysis and exposed in an intuitive way.

1.2 Intended Audience

This Software Requirements Specification (SRS) is mainly for the scope of the Integrator Project I discipline, but it can be used by anyone interested in the field to study and develop low-cost and accessible technologies to be implemented in Brazil.

1.3 Intended Use

This document is intended to be used for the evaluation of the Integrator II Project and also for the replication of the project by third parties as long as it is authorized by the author.

1.4 Project Scope

The software is basically composed of two separate modules that communicate over radio waves using the LoRa-WAN protocol. The first part to be described concerns the

weather station that will capture the data from the sensors and sent it to the gateway. The class diagram can be seen in Figure 1.1.

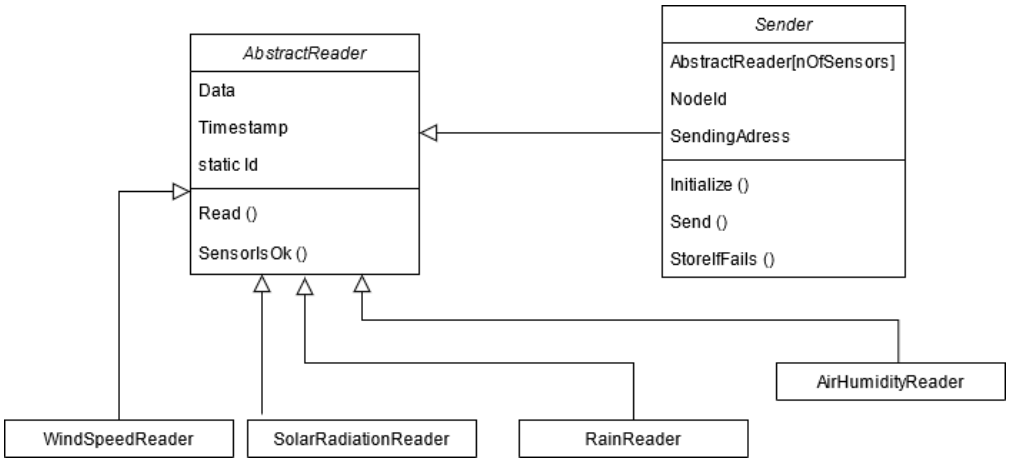

Figure 1.1: Metereological station user classes diagram.

The second part basically receives the information, filters it and sends it to a server. The gateway class diagram can be seen in Figure 1.2

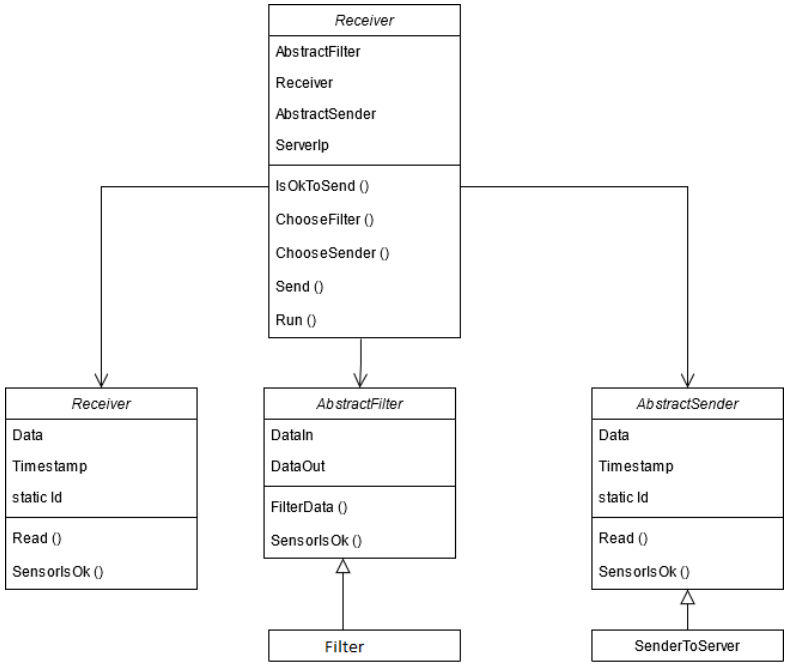

Figure 1.2: LoRa-WAN gateway user classes diagram.

The server that will receive the information via ethernet will analyze the data, check

favorable conditions for the diseases included and issue an alert if there is danger. The User Class diagram of the server is shown in the Figure 4.3. 1.3.

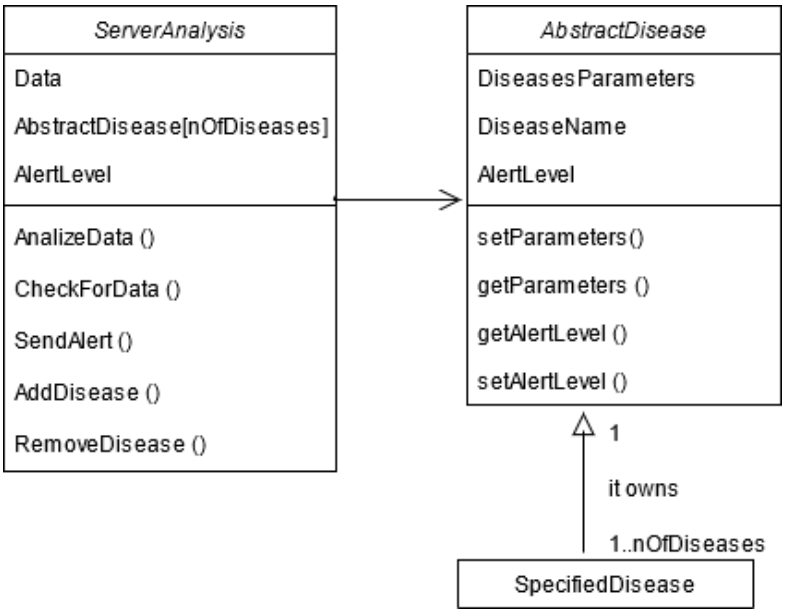

Figure 1.3: LoRa-WAN Server user classes diagram.

2 Overall Description

2.1 User Needs

The product is intended for small and medium-sized corn farms, which the owners cannot afford with more expensive solutions (such as drones). Possible users range from the farm owner, its employees and potential agronomists.

This satisfies the need for data on growing corn in an intuitive way that even people without technological knowledge can understand.

2.2 Assumptions and Dependencies

The only critical need that can affect the functioning of the system is the lack of a stable internet connection at the base of the gateway, something quite common in more isolated farms. Despite this, the other needs can be easily satisfied.

3 System Features and Requirements

3.1 Function and Requirements

3.1.1 Software Dependencies

- LoRa Gateway OS (latest version)

3.1.2 Hardware Dependencies

- LoRa-WAN Module RHF76-052 for nodes
- Tiva C Series TM4C for nodes
- LoRa-WAN Module RHF0M301 for gateway
- Raspberry Pi 2B for gateway
- 12V Batery
- Solar Radiation Sensor (model tbd)
- Wind Speed Sensor (model tbd)
- Pluviometer (model tbd)
- Air Humidity Sensor (model tbd)

3.2 External Interface Requirements

A server is required to send the data to be exposed to the user. LoRa Server will be used, which is an open source server supported by LoRa Gateway OS.

3.3 System Features

The gateway needs to have access to an internet network with a stable connection, not needing to be broadband.

3.4 Nonfunctional Requirements

It is desirable for the system to be able to run on battery power for a few years, as the LoRa-WAN protocol has a low energy cost.

4 Diagrams

4.1 User Class Diagram

The software is basically composed of two separate modules that communicate over radio waves using the LoRa-WAN protocol and a third part that receives the information via ethernet. The first part to be described concerns the metereologica station that will capture the data from the sensors and sent it to the gateway. The class diagram can be seen in Figure 4.1

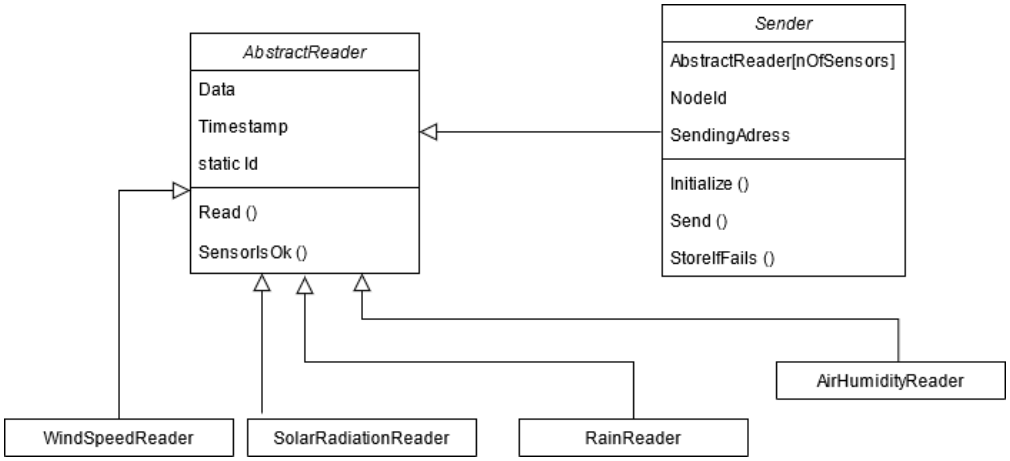

Figure 4.1: Metereological station user classes diagram.

The second part basically receives the information, filters it and sends it to a server. The gateway class diagram can be seen in Figure 4.2

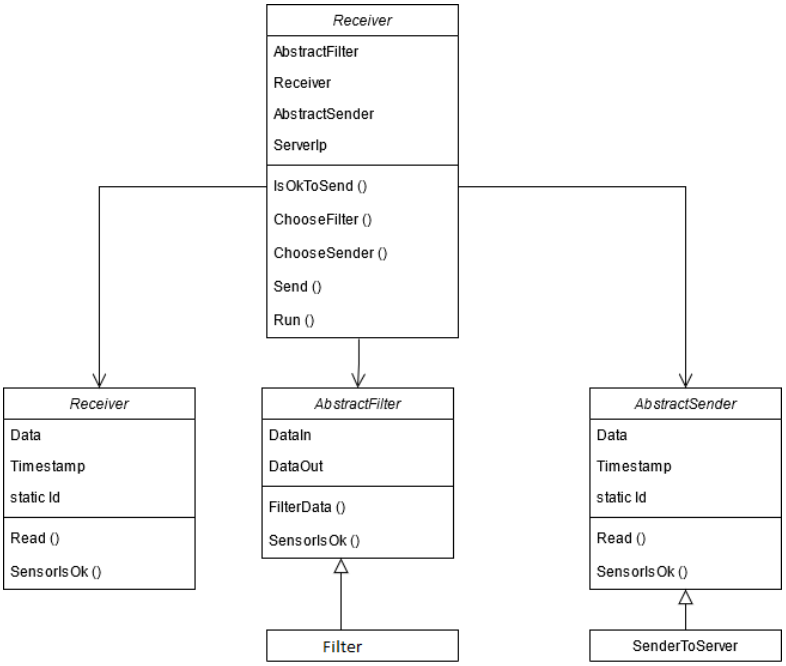

Figure 4.2: LoRa-WAN gateway user classes diagram.

The server that will receive the information via ethernet will analyze the data, check favorable conditions for the diseases included and issue an alert if there is danger. The User Class diagram of the server is shown in the Figure 4.3.

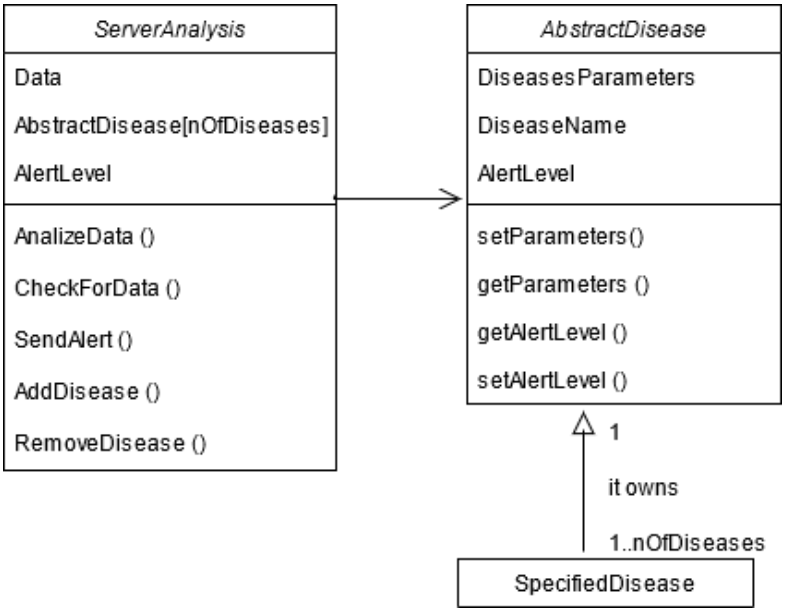

Figure 4.3: LoRa-WAN Server user classes diagram.

4.2 State Machine Diagram

The state machine diagram for the LoRa-WAN weather station has three layers, similar to the user class diagram. These being the weather station, the gateway and the server. The complete diagram can be seen in Figure 4.4.

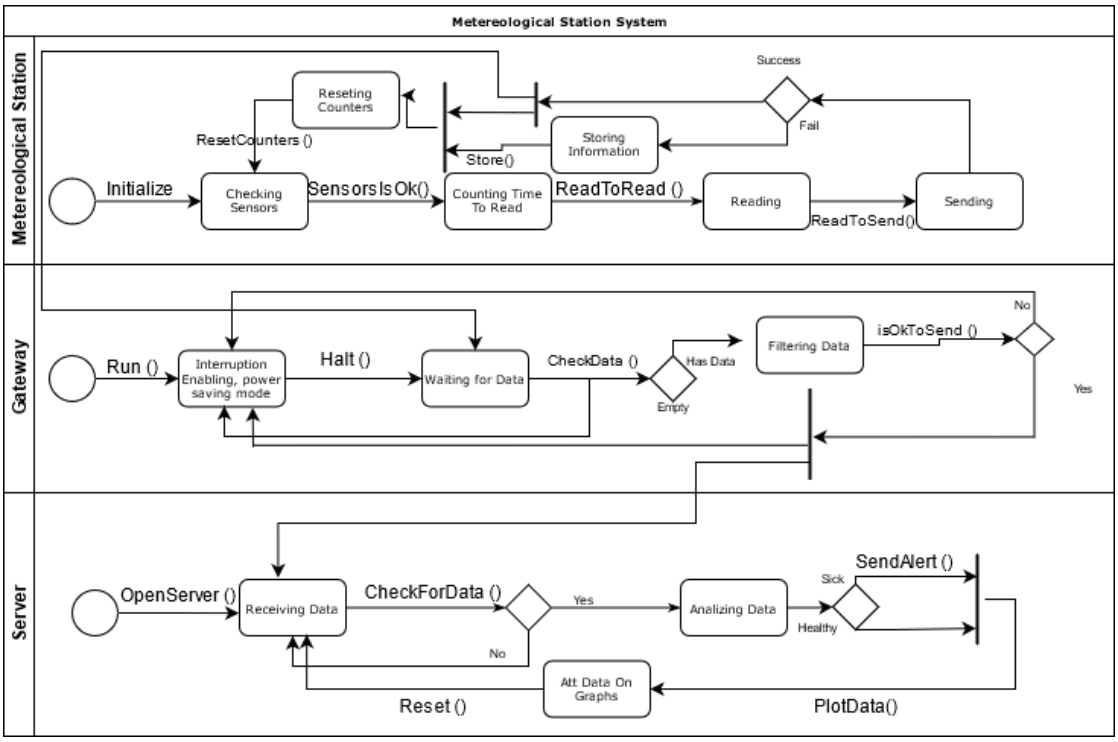

Figure 4.4: Machine State Diagram of the LoRa-WAN Metereological Station.

Basically, the part of the weather station processes the data and, if valid, sends a signal with the information to the gateway and at the same time returns to the counting state.

The Gateway, in turn, waits for an interruption in receiving data, checks the data, filters the data and sends a signal with the information to the server.

When the server receives a signal that there is new information in the database, it analyzes the information and issues a danger alert if there is.

5 Development and results

In the Figure 5.1, it is possible to visualize the proposed system in a more practical way and in its application area.

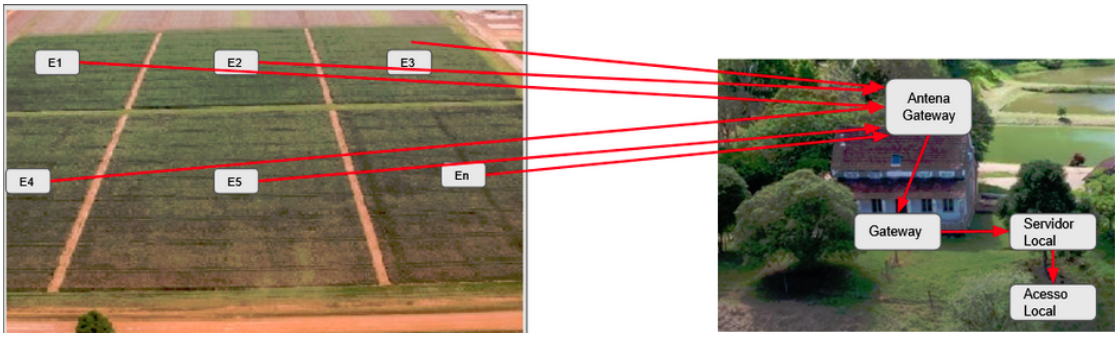

Figure 5.1: System caricature.

There will be a possibility that several nodes will send information to one gateway and the gateway will store the information on a local database. All the infos will be shown in intuitive graphs.

A block diagram for the statition node can be seen on the Figure 5.2.

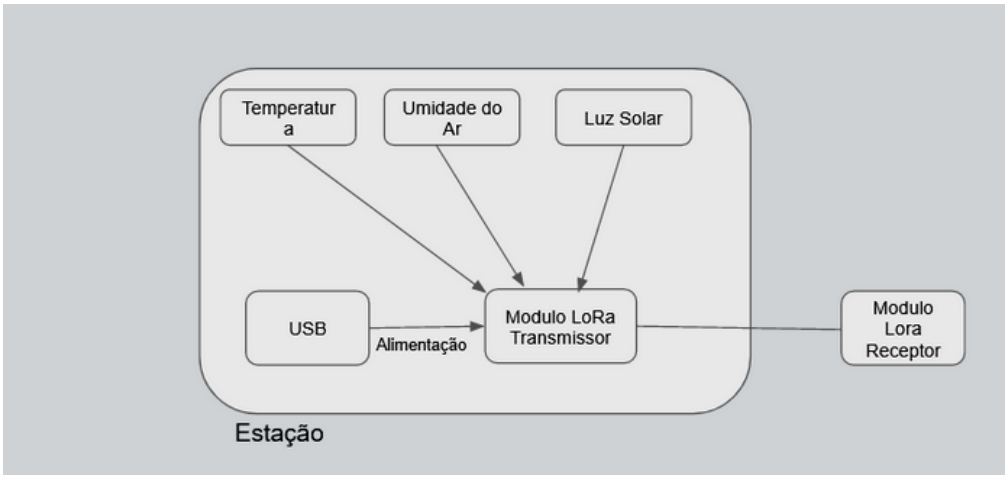

Figure 5.2: Block diagram for the transmitter node.

This block represents the part of the system that will read the sensor information and transmit via LoRaWaN to the gateway. The node will be USB powered for development purposes only.

The part of the system that will receive the transmission is shown in Figure 5.3.

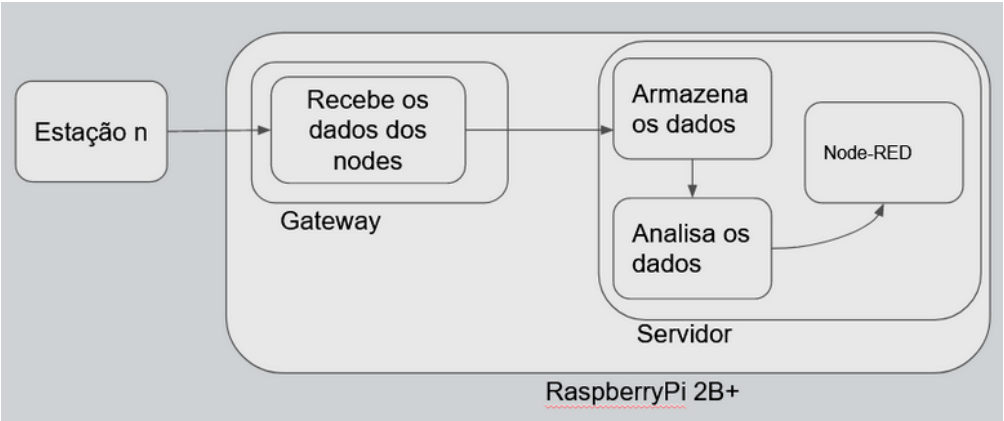

Figure 5.3: Block diagram for the receiver.

A n number of stations can send message to the receiver. The receiver will save the data on a local database, process that data and shown the results on a Node-RED interface.

To construct the project, some items are needed. For the nodes:

- LoRa Module E32-TTL-100
- SMA antenna (430 mhz)
- ESP32
- 5v battery or USB
- Solar Light Sensor (LDR)
- Humidity and temperature sensor (DH11)

And for the gateway:

- LoRa Module E32-TTL-100
- SMA antenna (430 mhz)
- Raspberry Pi 2B
- FTDI or arduino acting like FTDI

All the wiring conections between the ESP32, the LoRa module and the sensor were made following the connections presented on the Table below.

EPS32	LoRa Module	DHT	LDR
3v3	Pin 6	Pin 1	Pin 1
GND	Pins 1, 2, 7	Pin 4	Pin 2
D5	x	Pin 2	x
D6	Pin 3	x	x
D7	Pin 4	x	x
A0	x	x	On the voltage divider

All node hardware has been placed inside a 3D printed case to encapsulate and protect the system. The final result can bem seen on the Figure 5.4.

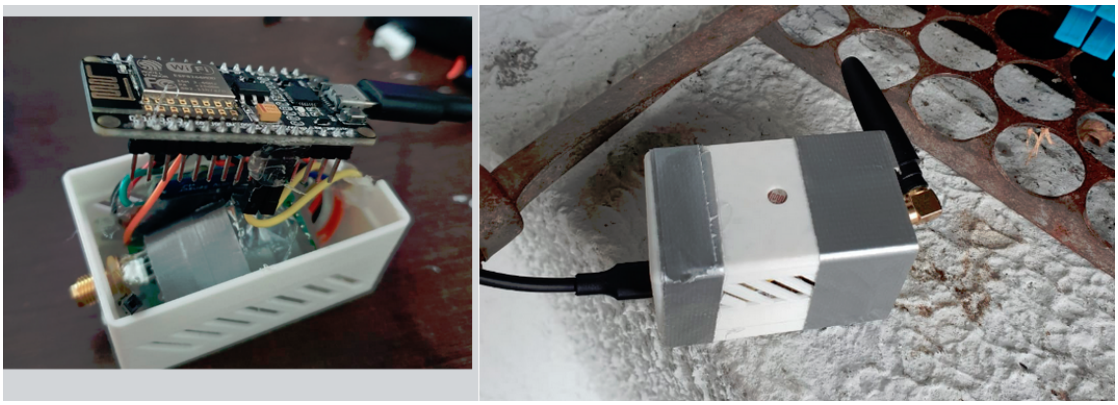

Figure 5.4: Transmitter.

The LDR sensor is facing up in a hole to get the light and the antenna is placed outside the package through a larger hole. Around the case there are slits for circulating air and the DH11 sensor can measure air humidity. Also, it helps the system not to overheat.

The node system acquires every 300 ms and sends an average of the last 10 samples every 5 seconds. The code can be seen on Github.

The receiver was also built and the result can be seen in Figure 5.5.

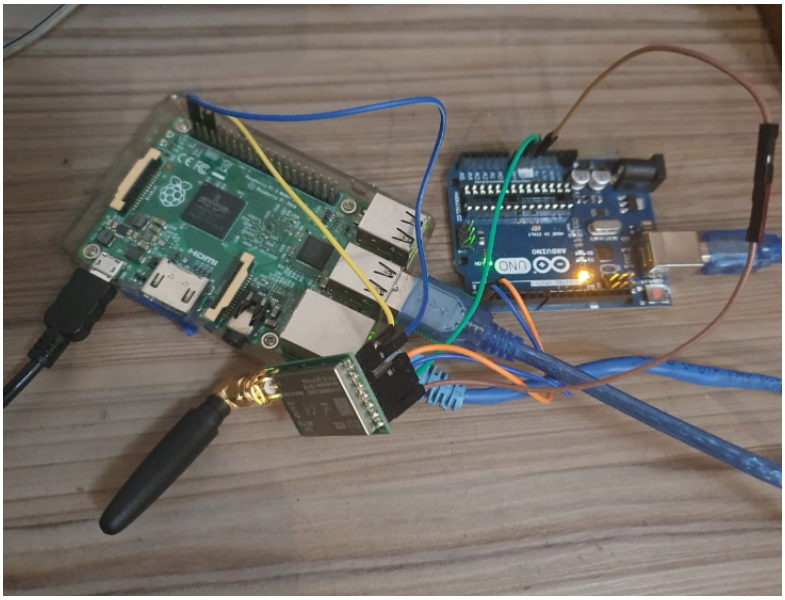

Figure 5.5: Receptor.

The connections beetween the LoRa module, the RPi and the Arduino Uno can be seen on the Table below.

RPi 2B	LoRa Module	Arduino
3v3	Pin 6	Pin 1
GND	Pins 1, 2, 7	GND
x	Pin 3	Rx
x	Pin 4	Tx
USB port	x	USB cable

This parte of the system receive the data from the LoRa modulo through the Serial port, save the logs, process the data and expose the informations in a UI built on Node-RED.

The flow diagram on the Node-RED can be seen on the Figure 5.6 with focus on the blocks parsing the message.

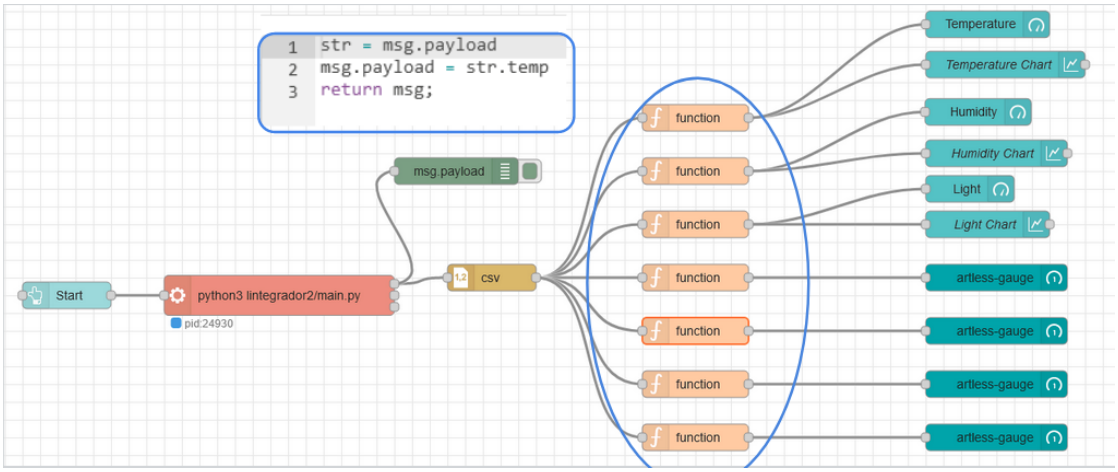

Figure 5.6: Node-RED parsing message.

and the flow with focus on the nodes that built the UI can be seen on the Figure 5.7.

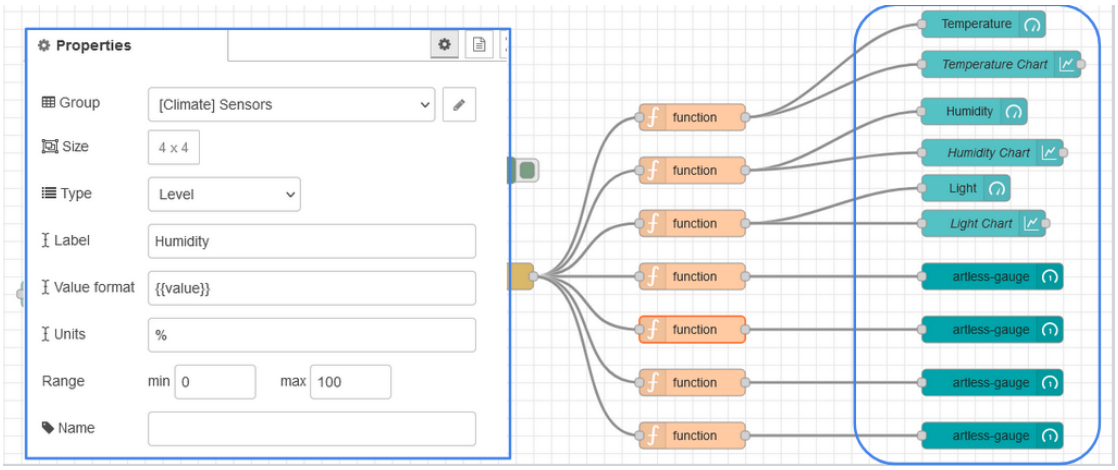

Figure 5.7: Node-RED building UI.

The result of the UI built is shown on the Figure 5.8.

Figure 5.8: Ui built in Node-RED.

Is possible to see a start button to force the system to work if it doesn't happen automatically. On the right, the sensor values are shown in some charts, instant values and over the time. Also, the instant risk of diseases is indicated.

The data is saved in a local database and the result can be seen on the Figure 5.9.

```
timestamp,temperature,humidity,light
2021-08-29 23:42:05.565491,25.40,65.00,64
2021-08-29 23:42:45.974639,25.40,65.00,65
2021-08-29 23:42:50.566221,25.40,65.00,67
2021-08-29 23:42:55.563599,25.40,65.00,67
2021-08-29 23:43:00.564996,25.40,65.00,67
2021-08-29 23:43:05.567536,25.40,65.00,67
2021-08-29 23:43:10.564874,25.40,65.00,67
2021-08-29 23:43:15.566316,25.40,65.00,66
2021-08-29 23:43:20.563673,25.40,65.00,63
2021-08-29 23:43:25.564808,25.40,65.00,63
2021-08-29 23:43:30.566308,25.40,65.00,62
2021-08-29 23:43:35.563597,25.40,65.00,62
2021-08-29 23:43:40.564798,25.40,65.00,63
2021-08-29 23:43:45.566432,25.40,65.00,62
2021-08-29 23:43:50.563734,25.40,65.00,63
2021-08-29 23:43:55.565015,25.40,65.00,62
```

log29-08-2021.csv	8 KB	29/08/2021 23:59:56
log30-08-2021.csv	402 KB	30/08/2021 11:35:41

Figure 5.9: Database.

The data is save in a .csv format, with timestamp and the sensors information and a new .csv file is created when the day changes.

Some unit tests were created for the diseases cases, as is possible to see on the Figure 5.10.

5.9.

```python
import unittest
from diagnose import Diseases

class TestDiseases(unittest.TestCase):

    def test_d1(self):
        d1 = Diseases("Ferrugem Comum", 16, 23, 95, 100, 0, 1023)
        self.assertEqual (d1.checkIfPossible(10, 90, 0), 1)
        self.assertEqual (d1.checkIfPossible(16, 90, 0), 2)
        self.assertEqual (d1.checkIfPossible(20, 90, 0), 2)
        self.assertEqual (d1.checkIfPossible(23, 90, 0), 2)
        self.assertEqual (d1.checkIfPossible(30, 90, 0), 1)

        # variando umi para cada temp. luz 0
        self.assertEqual (d1.checkIfPossible(10, 95, 0), 2)
        self.assertEqual (d1.checkIfPossible(10, 97, 0), 2)
        self.assertEqual (d1.checkIfPossible(10, 100, 0), 2)
```

Figure 5.10: Unit testing.

And the results can be seen on the Figure 5.11. 5.9.

```
pi@raspberrypi:~/Iintegrador2 $ python3 -m unittest test_diagnose.py -v
test_d1 (test_diagnose.TestDiseases) ... ok
test_d2 (test_diagnose.TestDiseases) ... ok
test_d3 (test_diagnose.TestDiseases) ... ok
test_d4 (test_diagnose.TestDiseases) ... ok

----------------------------------------------------------------------
Ran 4 tests in 0.007s

OK
```

Figure 5.11: Unit testing.

Therefore, all tests performed did not result in errors.

6 Conclusion

The project was built during the semester of 2021.1 and suffered some changes compared to its initial planning. LoRa-Wan protocol was replaced by LoRa protocol, wind sensors were not used and battery usage was never implemented. Also, some changes to the code body impacted previously designed diagrams.

However, the project worked as expected and in a satisfactory and stable way, in addition to the addition of the Node-RED platform, which opened up a wide range of features for the system. With some more time invested in developing some future changes, the project will be quite robust and may even evolve as a product.

7 Future Development

For future development, some ideas would add robustness to the system and important new features, such as:

- Implement an MQTT protocol for acknowledgment bit exchanges, ensuring message arrival
- Checksum to ensure message integrity
- Send e-mail if dangerous conditions remain for a long time
- Implementation of a battery and energy expenditure analysis
- Possible exchange of MCU or sensors for others with lower consumption
- Node-RED interface with InfluxDB
- Maybe replace the interface in Node-RED with one in Grafana