# Rainforest IoT tracking

Nonali Andrea

Innovative Telecommunication Systems - 8/09/2022

# 1 Introduction

The following paper presents an IoT simulated system that aims to preserve the environment and the bird species of an indoor artificial rainforest, which consists of various species of plants and birds. This ecosystem will be equipped with various sensors, both to enable the customers to have an experience similar to reality and to enable workers to understand better the live condition of the rainforest. Given that the birds live closely with humans in an artificial and automated environment, special monitoring of their health conditions and location is needed. For simplicity sake, our project will use sensors that targets only three species of birds, but adding any other animal is possible with the same procedures described. The technologies chosen for this project, aside from Node-RED, are the Python programming language, which has several libraries to create a TCP server used to simulate the sensors and a famous library called Tkinter, suitable to create desktop applications.

# 2 Devices and Data

## 2.1 Environment

The environment consists of various sensors that cooperate to maintain ideal conditions. The humidity and soil moisture sensors send the respective value, while the user presence one sends whether a user is present in the area where is located. The water vaporizer and the irrigator are automatically set on or off based on various conditions described in the next chapters, with a settable power.

| Name | Id | Data Type | Description |
|---|---|---|---|
| User Presence | 1 | Boolean | Detects the presence of the users in certain points |
| Water Vaporizer | 2 | Boolean | Sprinkles vaporized water |
| Vaporizer Power | 3 | Float | Sets the power of water vaporizer |
| Irrigator | 4 | Boolean | Irrigate the terrain |
| Irrigator Power | 5 | Float | Sets the power of irrigators |
| Humidity | 6 | Float | Returns the humidity value of the environment |
| Soil Moisture | 7 | Float | Returns the soil moisture of the terrain |

## 2.2 Birds

Each bird has a health sensor and a GPS sensor. The health sensor sends the body temperature and the heart rate of the bird. The GPS sensor sends the latitude and longitude of it.

| Name | Id | Data Type | Description |
|---|---|---|---|
| Bird1 Health | 8 | JSON | returns the heart rate and body temperature of Bird1 |
| Bird2 Health | 9 | JSON | returns the heart rate and body temperature of Bird2 |
| Bird3 Health | 10 | JSON | returns the heart rate and body temperature of Bird3 |
| Bird1 Coods | 11 | JSON | returns the lat and long of Bird1 |
| Bird2 Coords | 12 | JSON | returns the lat and long of Bird2 |
| Bird3 Coords | 13 | JSON | returns the lat and long of Bird3 |

### 2.3 Alarms

The alarms are special sensors that are triggered when the environment or the birds sensors detect non-ideal conditions. Note that each bird will have his corresponding alarm for health and gps coordinates.

| Name | Id | Data Type | Description |
|---|---|---|---|
| Humidity | 14 | Boolean | Signals anomalous humidity |
| Soil Moisture | 15 | Boolean | Signals anomalous soil moisture |
| HeartBeat | 16-18 | Boolean | Signals anomalous heartbeats |
| Body Temperature | 18-21 | Boolean | Signals anomalous body temperature |
| Boundaries | 21-24 | Boolean | Signals anomalous position of birds |

#### 2.3.1 Alarms trigger conditions

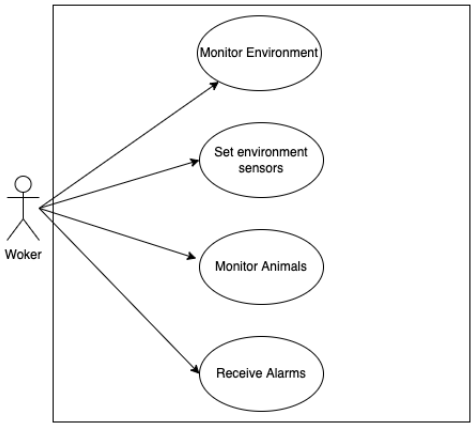
Below the table that describes the non-ideal conditions for which the alarm will be set on.

| Name | Range |
|---|---|
| Humidity | hum <77 or hum >88 |
| Soil Moisture | sm <77 or sm >88 |
| Heart Beat | hb <285 or hb >305 |
| Body Temperature | bt <39 or bt >43 |
| Boundaries | lat/lon <40 or lat/lon >50 |

## 3 System design

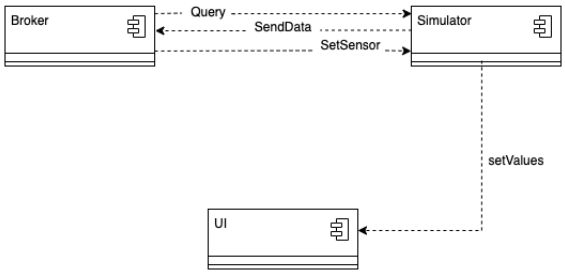
### 3.1 Use Case

The system must enable the workers to monitor all the sensors and to control some of them. Workers must also be able to receive the alarms whenever the values of the sensors are not in the ideal range.




## 3.2 System Entities

The main entities of the system are the sensor simulator, the graphical interface and the Node-RED broker. Their high-level communication is described in this diagram.




## 3.3 Communication Protocols

### 3.3.1 TCP

As the previous diagram illustrates, the broker entity is responsible to gather the information regarding the sensors, or to set the sensors to specific values. To do so, it contacts the simulator entity using the TCP protocol with the following request schema:

- *GET [device_id]*
- *SET [device_id] [value]*

When the server receives a message, it uses the *Command* class to parse the command and validate it, as described by the next diagram.


If the command is not valid an error message will be returned with one or more specification messages regarding the errors that occurred; otherwise the sensor will execute the operation indicated by the command class. The code of the command class is illustrated below.

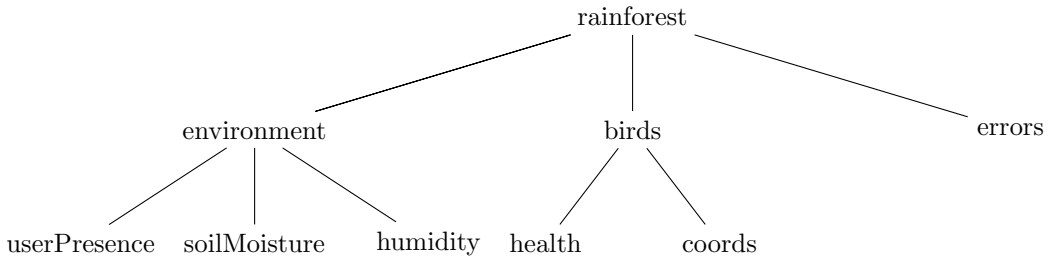
```
class Command:
    def __init__(self, data: str):
        self.error_message = "ERRORS: "
        self._parse(data)
        self._validate()

    def _parse(self, data: str):
        data = data.split(" ")
        self.action = data[0]
        self.sensor_id = data[1]
        try:
            self.value = self._parse_value(data[2])
        except IndexError:
            self.value = None

    def _validate(self):
        self.valid = True
        if not self.action in AVAILABLE_ACTIONS:
            self.error_message += f"- invalid action {self.action}"
            self.valid = False
        if not exists(self.sensor_id):
            self.error_message += f"- invalid sensor id {self.
    sensor_id}"
            self.valid = False
```

### 3.3.2 MQTT

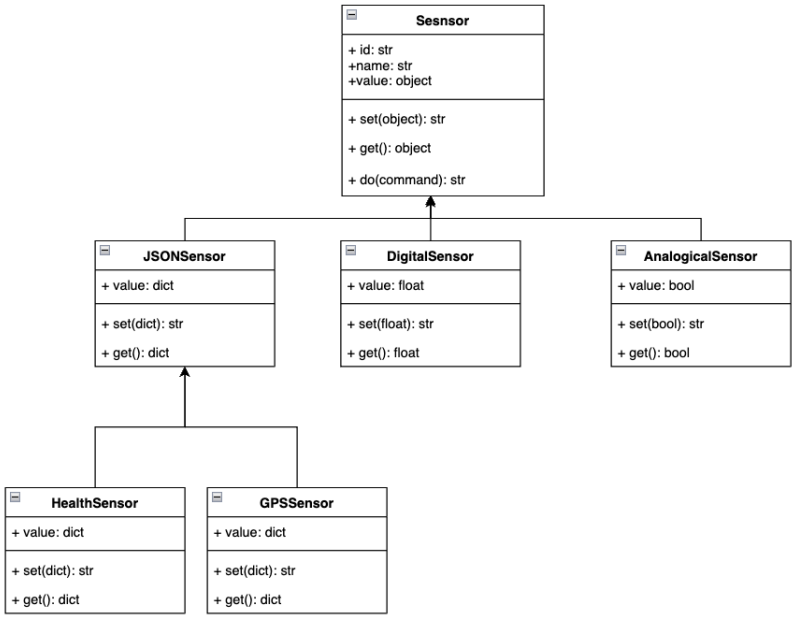
The MQTT protocol is used by the broker to notify about the data collected from the simulator. In this way an entity can receive the data on a topic of interest to perform its specific tasks, such as saving data to the database, logging errors to a file or creating live charts. The MQTT protocol designed has the following structure.



The *errors* topic has a subtopic for every topic of the environment and birds ones, that for simplicity of the tree graph was omitted.

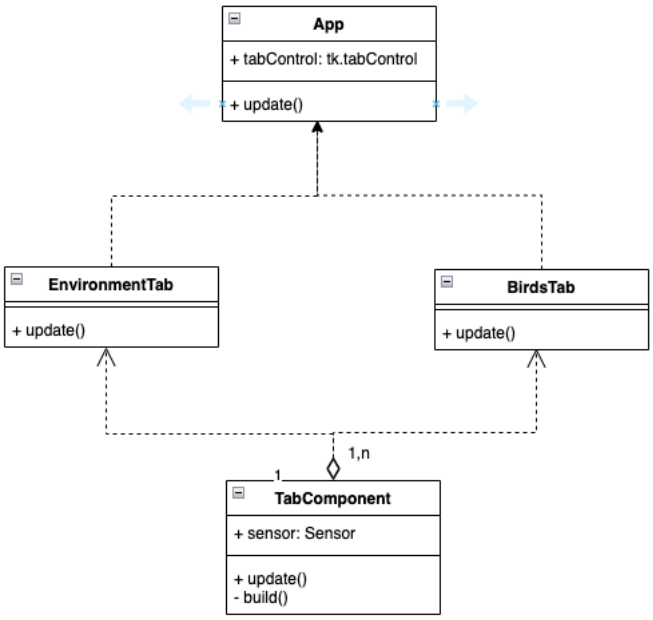
## 3.4 Simulator sensors

The simulator is able to mock three types of sensors: analogical, digital and JSON sensors, which are special sensors that maps a digital values to a string of characters.



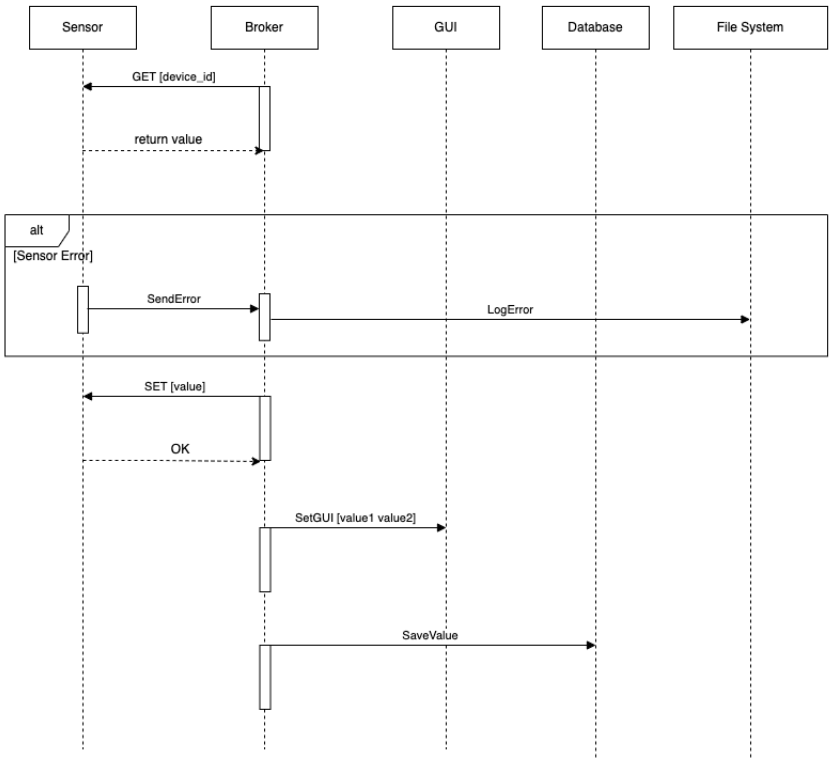

## 3.5 UI Composition

The UI has two main tabs, the environment tab and the birds tab, each with the information gathered from the sensors. Each tab will have a component that renders the data based on the type of sensor it represents. Moreover all the components have an *update()* function that is triggered every second to update its content.




## 3.6 Broker Workflow

The Node-RED broker is in charge of querying the sensor and perform the logic to activate or deactivate some sensors and the alarms. Moreover, the broker is also responsible to handle and log errors received from the sensors and to save the queried values to the database. This sequence diagram describes the general workflow of all the flows that are used by the broker.




# 4 Flows Design

## 4.1 Common Patterns

Every flow described in this chapter has a common pattern to handle common events.

### 4.1.1 Error Handling

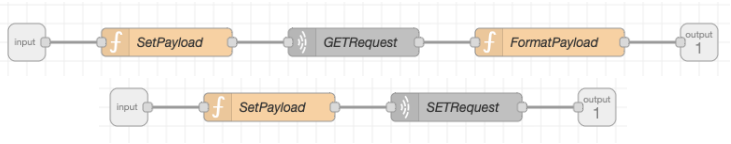
Every flow handles the errors returned from the server with the *Switch* command of Nodered. In fact, every error (or group of errors) that the server returns is preceded by the *ERRORS* string, and the switch can simply check if the response returned by the server contains this string. If an error is contained, then it will be sent to the corresponding MQTT topic and logged in a dedicated file.

### 4.1.2 MQTT and gathered values

Every flow sends the values retrieved from the sensors to the corresponding MQTT topic, encrypted with the AES algorithm. This allows for every subscriber that has the encryption key to handle the values for various purposes, such as logging or storing in the db the values.

## 4.2 Subflows

Two subflows, *TCPQuery* and *TCPSetSensor* are used to respectively query the sensor simulator and set the sensor of the simulator.




## 4.3 Water Vaporizer

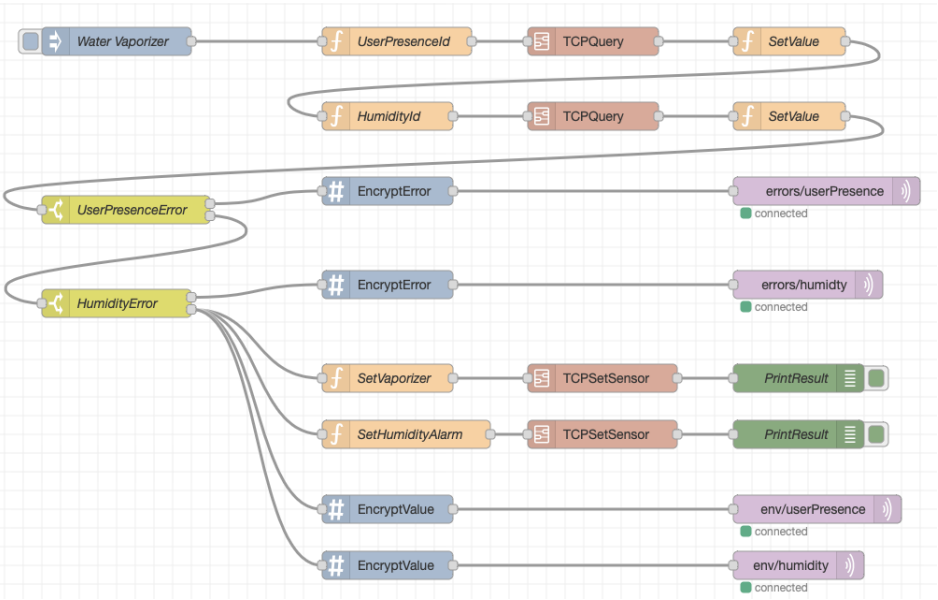
This flow is responsible to gather the data from both the user presence sensor and the humidity sensor, and to enable the water vaporizer under certain conditions, described by this table:

| User Present | Humidity <77 | Sensor Values |
|---|---|---|
| False | False | Off |
| True | False | Humidity <88 ? On: Off |
| False | True | On |
| True | True | On |

If one or more sensors will present an error, then water vaporizer will be enabled only under the conditions described below:

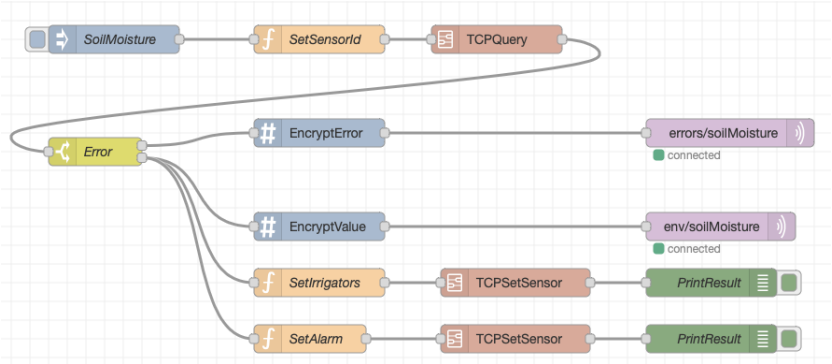
| User Present Error | Humidity Error | Sensor Values |
|---|---|---|
| True | False | Humidity <77 ? On: Off |
| False | True | Off |
| True | True | Off |

If the humidity is out of range, an alarm will be also activated, otherwise deactivated.



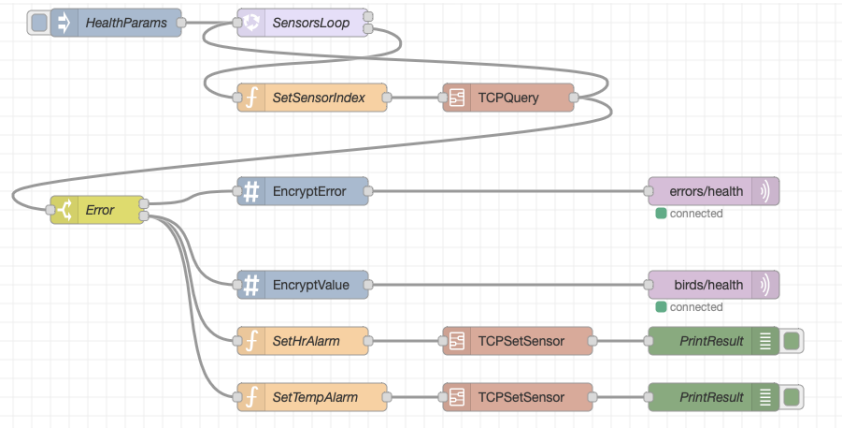

## 4.4 Irrigation

This flow checks the soil moisture value. If this value is below the recommended value of 77%, the irrigators will be activated. If the soil moisture is below this value an alarm will be also activated, otherwise deactivated.



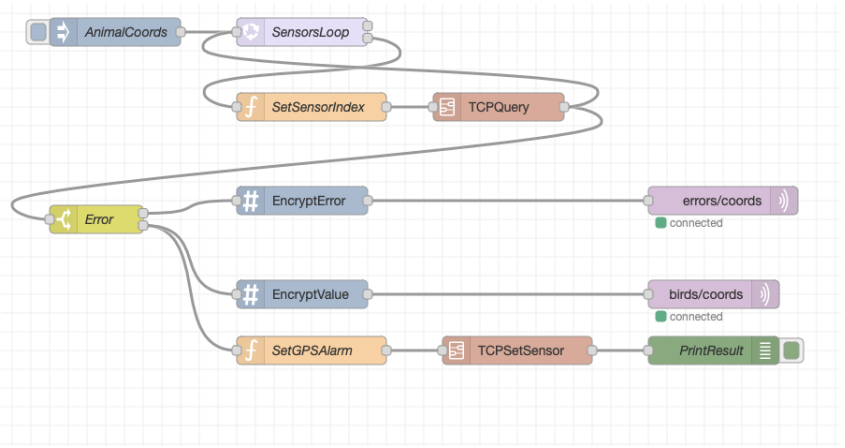

## 4.5 Animals Health

This flow runs in a loop that sets the correct sensor id to collect the information about the birds health. If the heart rate or the body temperature values are not in the correct range, an alarm will be also activated, otherwise deactivated.



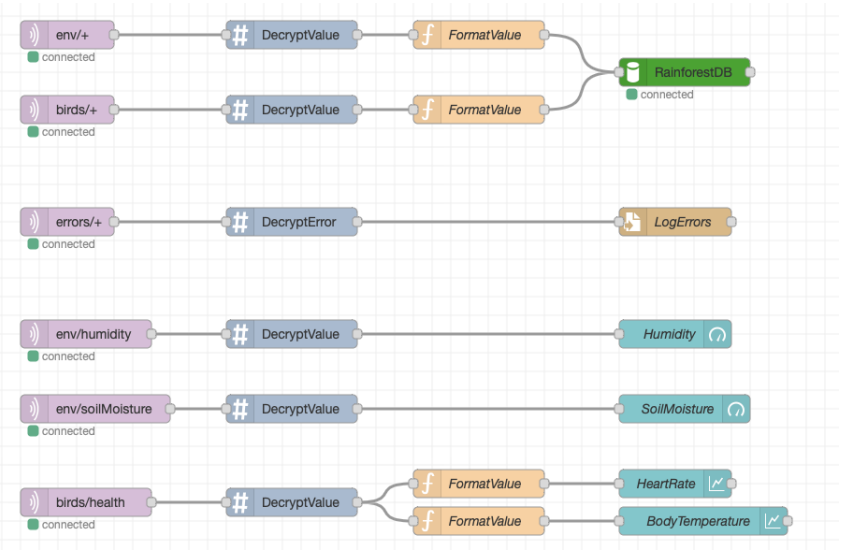

## 4.6 Animals Coordinates

This flow runs in a loop that sets the correct sensor id to collect the information about the birds coordinates. If this values are not in the boundaries of the structure, an alarm will be activated.



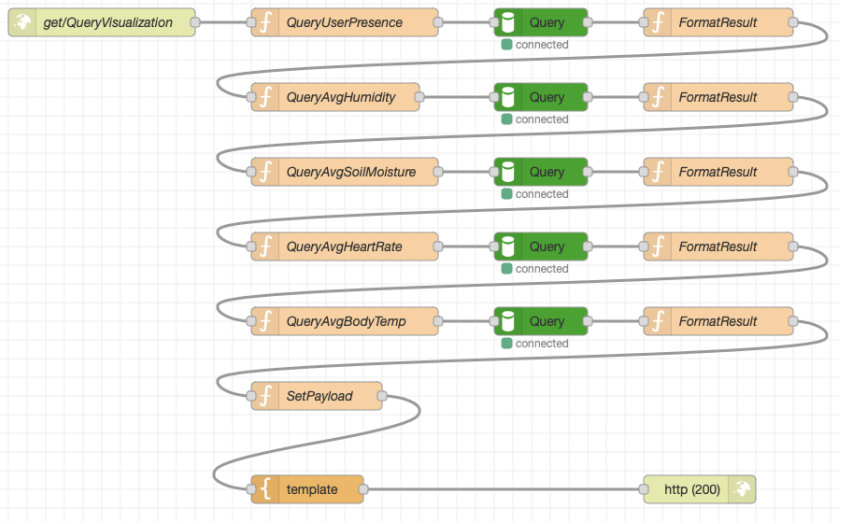

## 4.7 MQTT Subscribers

This flow manages the MQTT messages received from the other flows. Its responsibility is to save the information gathered from the sensors in the database, create live charts and log errors. Note that all the useful informatino to save in the correct collection the values, or to send the values to the correct chart line are retrieved using a function that parses the MQTT msg.topic.




## 4.8 Query Visualization

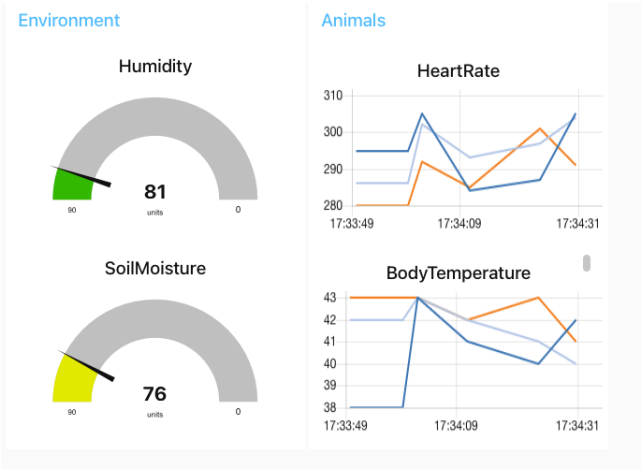
This flow takes advantage of the queries defined in the chapter 6.1 to create a web page located at *127.0.0.1:1880/queries* that visualizes their result. Note that all the queries are performed considering the current date as the date range.




## 4.9 Live charts

Live charts are available at the *127.0.0.1:1880/ui* url. The charts represent the values regarding the environment and the birds. For the environment there are

two gauge charts that represent the range of the soil moisture and the humidity of the environment. For the birds there are two line charts that represent the evolution of the health parameters (heart rate and body temperature) for each animal over time.

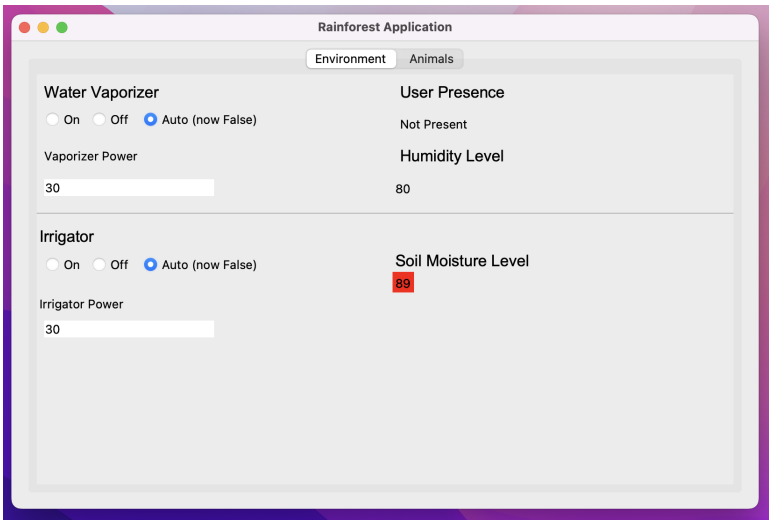



# 5 Graphical Interface Design

The graphical interface is meant to help workers to track the status of the environment and birds and to activate or deactivate certain sensors. It is composed of two tabs, one regarding the environment and the other regarding the animals parameters.

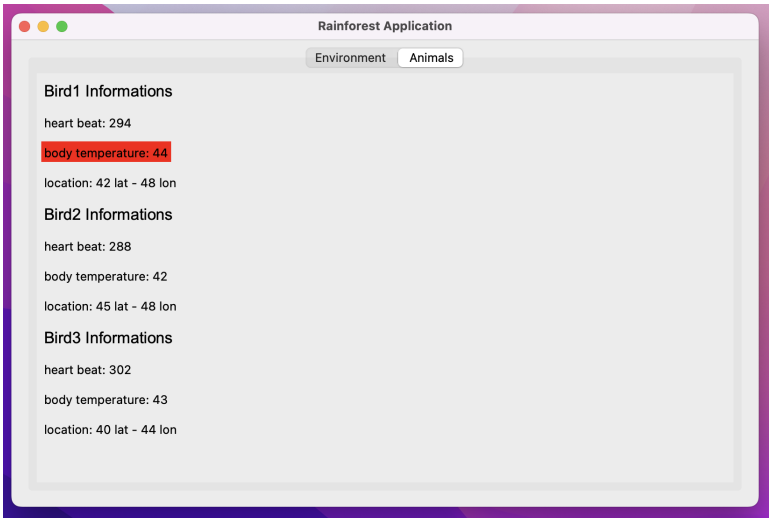
## 5.1 Environment Tab

The environment tab has two sections. The first section is dedicated to the water vaporizer. The water vaporizer can be set on or off by the worker, but this will lock the sensor, that will become not settable anymore by the Nodered broker in automatic, based on the sensor values. If the value of the vaporizer is set to auto, then the broker will decide the value of the sensor depending on the conditions described in the flows chapter. The right part of the first section shows the value of the user presence and the humidity level. If the humidity alarm is on, then the humidity value label will be colored with red. The second section function is similar to the first section, but is referred to the irrigation case, that is controlled by the soil moisture level. Also the label of the soil moisture becomes red if the alarm sensor is on.




## 5.2 Birds Tab

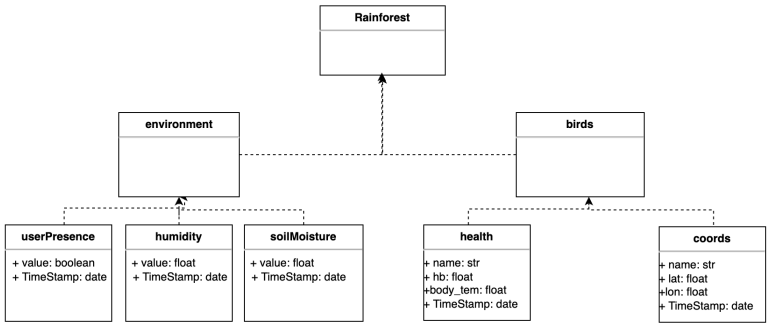
The birds tab contains the information of each bird. If one or more information of each bird is out of the range, this means that the alarm will be set on by the Nodered broker, and the interface will color the values out of range with the red color.




# 6 Database

The database used for this project is MongoDB. This database has a NoSQL collection for every MQTT topic aside from the errors that are logged in a file.

The structure of the database is described in the image below.




## 6.1 Queries

The database allows for queries to monitor the state of the system from a higher level than live data. The queries can be performed by the mongodb shell, however, as described in chapter 4.8, users can use the web page to see the results of the queries.

### 6.1.1 Water Vaporizer

This query counts how many people passed under the water vaporizer on a specific day. It can be used along other queries to understand better how to manage the humidity of the environment, since most of the time a user passes in the area of the water vaporizer the water will be sprinkled.

```
db.env.userPresence.find({
    "date" : {
        "$gte": new Date("2022-09-01T00:00:00.000Z"),
        "$lt": new Date("2022-09-02T00:00:00.000Z")
    }
}).count()
```

### 6.1.2 Average humidity or soil moisture

```
db.env.<humidity/soilMoisture>.aggregate([
    {
        $match: {
            "date" : {
                "$gte": new Date("2022-09-01T00:00:00.000Z"),
                "$lt": new Date("2022-09-02T00:00:00.000Z")
            }
        }
    },
    {
        $group: { _id: null, avg: { $avg: "$value" } }
    }
])
```

### 6.1.3 Average heart rate or body temperature

```
db.birds.health.aggregate([
    {
        $match: {
            "date" : {
                "$gte": new Date("2022-09-01T00:00:00.000Z"),
                "$lt": new Date("2022-09-02T00:00:00.000Z")
            }
        }
    },
    {
        $group: { _id: "$name", avg: { $avg: "$hb/body_tem" } }
    }
])
```

### 6.1.4 Birds that had health problems

```
db.birds.health.find({
    "$or": [
        {
            "hb": {
                "$lt": 285,
                "$gt": 305
            }
        },
        {
            "body_tem": {
                "$lt": 39,
                "$gt": 45
            }
        }
    ]
})
```

### 6.1.5 Birds that were out of boundaries

```
db.birds.coords.find({
    "$or": [
        {
            "lat": {
                "$lt": 40,
                "$gt": 50

            }
        },
        {
            "lon": {
                "$lt": 40,
                "$gt": 50
            }
        }
    ]
})
```

# 7 Randomization

The simulator randomizes the sensor values every 10 seconds. The values are adjusted using randomization rules that are applied to each sensor that needs to be randomized. This rules are encoded in a JSON object that has the sensor id, and the range values to use. Each sensor has a *randomize()* function to create a random value using the ranges indicated by the rules. This rules are designed to give the system a little chance to create a value out of the ideal values, so that the alarm will be activated. The code below is the exact representation of the rules.

```
RANDOMIZE_RULES = [
    {
        "index": 0,
        "range": 5
    },

    {
        "index": 5,
        "range": (72, 93)
    },
    {
        "index": 6,
        "range": (72, 93)
    },

    {
        "index": 7,
        "hb": (280, 310),
        "body_tem": (37, 45)
    },
    {
        "index": 8,
        "hb": (280, 310),
        "body_tem": (37, 45)
    },
    {
        "index": 9,
        "hb": (280, 310),
        "body_tem": (37, 45)
    },

    {
        "index": 10,
        "lat": (38, 52),
        "lon": (38, 52)
    },
    {
        "index": 11,
        "lat": (38, 52),
        "lon": (38, 52)
    },
    {
        "index": 12,
        "lat": (38, 52),
        "lon": (38, 52)
```

```
    }
]
```

# 8 Conclusions

The following paper described a working system to monitor the environment and birds conditions of the rainforest to enhance the user experience and the health of the ecosystem. The Python programming language and its libraries made possible to create bot a UI with modular and reusable components and a TCP server that can handle concurrently the requests. The Node-RED broker served as a middelware between the sensor and the UI, with the capability to perform the logic and handle all the situations required. Moreover the capabilities of the broker to work with the MQTT protocol, made it possible to publish the values gathered from the sensor with a suitable observer-subscriber pattern, that can also be used to extend the system with minimal design efforts. The MongoDB database served as a reliable data repository for all the values of the sensors, with queries to derive useful information to monitor the environment from a higher level point of view.