Proof of Concept (PoC)

Greenhouse Project

4 Janvier 2020

Teacher: Frédéric Fauberteau

TABLE OF CONTENTS:

INITIAL PROJECT: AUTOMATED GREENHOUSE

Our idea was to help farmers to grown healthy food, as fruits and vegetables.

So, it was to create a specific atmosphere condition and control our products. It was the creation of a greenhouse.

If we would like to get good plants, we need to consider the photosynthesis (Photosynthesis is the way which plants are going to absorb solar energy to grown).

So, if we want to reach good yield, we need to think about 3 parameters:

- Air humidity: to check hydric stress of plants
- Light: to get a lot of solar energy
- Temperature: more temperature is higher more plants can absorb water and nutrients

Moreover, we decided to create an automatic system in order to control our POC all day.

Our Automated Green House would allow to:

- Check all features
- Adjust them
- Send alerts when the system is out of limits
- Send a weekly report

HOW SHOULD IT WORK?

Step 1: Collect values from sensors

To control our Automated Green House, we choose 4 features:

- DHT11:
    - Temperature sensor: it detects temperature (°C)
    - Air humidity sensor: it detects humidity (%)
- LDR (Light sensor):it detects lightning of the greenhouse
- Web server: it allows to retrieve all the data from the sensors above, viewable live data through internet.

Step 2: Arduino Yun

There are links from sensors to Arduino:

- ADC converts value from DHT11 sensor to analog value
- PWM allows to have a variable value from Light sensor to analog value.

The sensors are connected to the Arduino thanks to a board and wires.

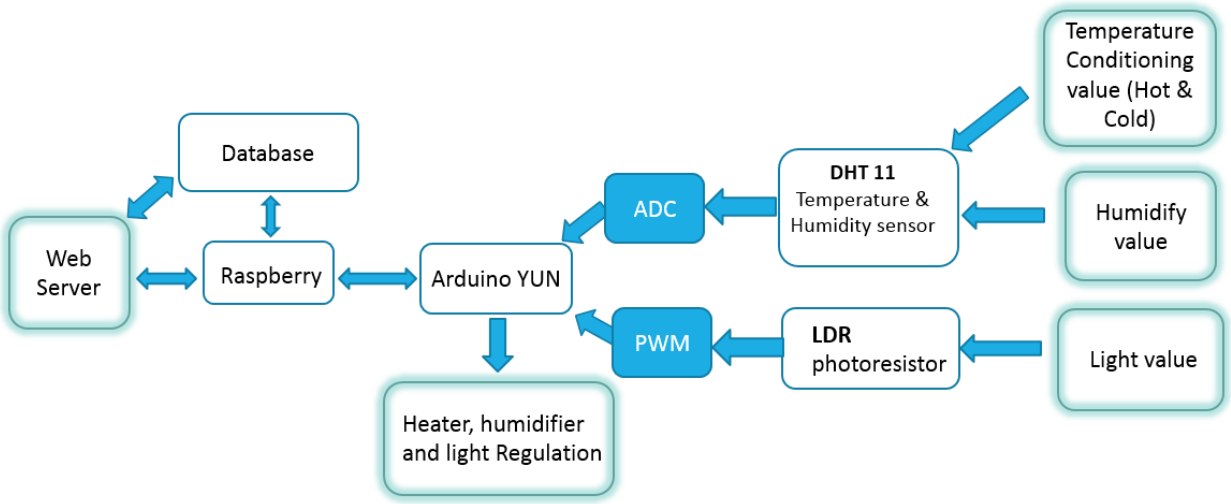
The Arduino will act on our POC project by 3 different ways:

- Collect values
- Display values
- Regulate the system retroactively (it can adjust T°C, air humidity and Light)

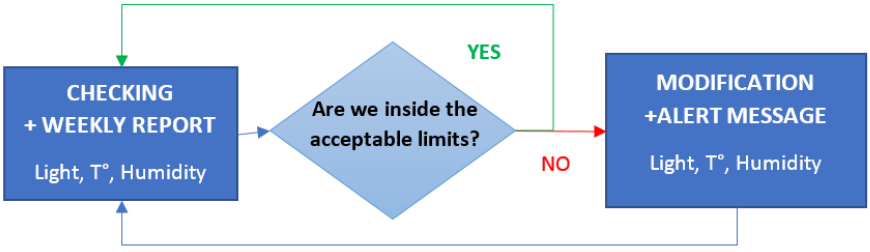
Step 3: Raspberry

The Arduino and Raspberry boards would be connected by serial communication.

Raspberry will receive information from Arduino and add them on a database and a web server



Picture 1: *General Diagram*



Picture 2: *General sensor control*

FEATURES

I. DHT 11: Humidity and Temperature Sensor

PURPOSE :

For the DHT11, our goal was to obtain information regarding air humidity and room temperature and affect it. Therefore, we searched for measurable values to categorize the state of the room. We discovered that the general optimal room temperature for plants is between 16-22°C. For humidity the value should be between 41-80% relative humidity. Our devices include an air humidifier and a heating device.

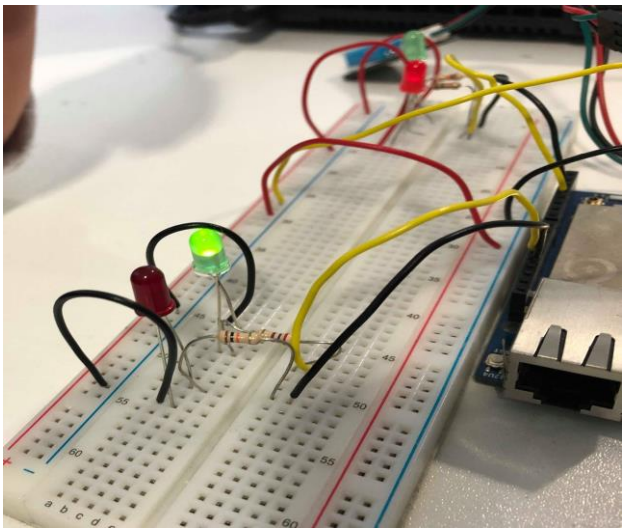
OPERATION

When the values for temperature and humidity falls below the targeted values, these devices should be turned on respectively. If the measurements are higher than the targets the devices should be turned off.

COMPOSITION

The system is composed of a humidity and temperature sensor: DHT11, two small LED system with 4 LEDs, 4 resistors 270 Ohms, a breadboard and an Arduino Yun.

The 2 small LED system is composed of one red LED and one green LED for each. One system is used for the heating and the other for the humidity.



Picture 3: *DHT 11 circuit*

CODE

The temperature and humidity are considered floating objects and are defined as zero to begin with. This value is upgraded for each time the DHT11 sensor is sampling. The code is started by including a DHT library in the form of a zip. The DHT is chosen as the analogue pin A0. The pins connected to the LEDs are number 2 for the heater, and 12 for the humidifier. These LEDs are green on our board. The red LEDs represent the absence of voltage delivered to the devices. All the pins are defined as outputs, except for the analogue pin connected to the DHT11. This one is defined as input. The values are read and printed.

When the sensor's detected values are higher than the limits stated, the green LEDs turn off- and the voltage for the red LEDs goes high. Therefore, we can see when the room is warm or humid enough. In this way we illustrate when the heater or humidifier must be turned off to stay within the range of values. When the values are too low the if test manages the voltage to be high on the devices and following the heater or humidifier will be turned on. The schematics are illustrated in the following picture.

```
#include <dht.h>

int DHT11pin=A0;
int Heating=2;
int Cooling=3;
int MoreMoist=12;
int MoreDry=13;
float DHTTemp=0;
float DHTHum=0;

dht DHT ;

void setup() {
  Serial.begin(9600);
  pinMode(Heating, OUTPUT);
  pinMode(Cooling, OUTPUT);
  pinMode(MoreMoist, OUTPUT);
  pinMode(MoreDry, OUTPUT);
  pinMode(DHT11pin, INPUT);
  delay(1000);
}

void loop() {
   DHT.read11(DHT11pin);
   DHTTemp=(DHT.temperature);
   DHTHum=(DHT.humidity);
   Serial.print("DHT11 Temperature measured as:");
   Serial.println(DHT.temperature);
   Serial.print("DHT11 Humidity measured as:");
   Serial.println(DHT.humidity);

   if (DHTTemp < 16) {
    digitalWrite(Heating, HIGH);
   // delay (1000);
    digitalWrite(Cooling, LOW);
    //delay (1000);
   }
```

```
   if (DHTHum < 41) {
    digitalWrite(MoreMoist, HIGH);
    //delay (1000);
    digitalWrite(MoreDry, LOW);
    //delay (1000);
   }

   if (DHTTemp > 80) {
    digitalWrite(MoreMoist, LOW);
    //delay (1000);
    digitalWrite(MoreDry, HIGH);
    //delay (1000);
   }

   delay (500);
}
```

Pictures 4 & 5: *Codes of Temperature (left) and Air Humidity (right)*

II. Light sensor

PURPOSE

The purpose of the lighting system is to control and regulate the luminosity of the greenhouse.

OPERATION

The system is automatic. When it detects a low value of luminosity in the greenhouse, it will automatically turn on the light inside the greenhouse to give enough energy to the vegetation.

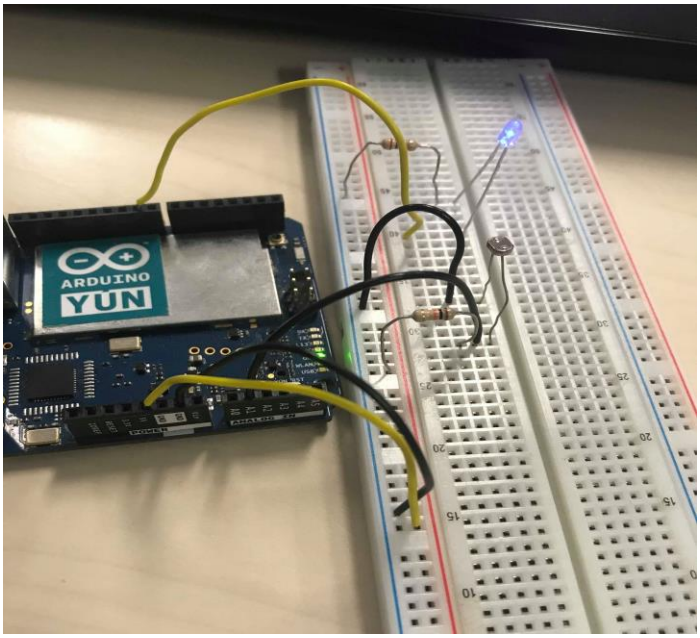
We only consider the sunlight. Because when the sun is hidden behind the clouds, there is light but not direct sunlight. Our system only compensates the lack of sunlight.

Here some examples of encountered situations:

- Situation 1: there is a really bad weather, the vegetation doesn't have enough sunlight. So, the system will turn on the light in the greenhouse, but just the right amount of light needed.
- Situation 2: there is a nice weather with a lot of sunlight. The greenhouse has enough light for the vegetation. So, our system will shut down the light in the greenhouse.

COMPOSITION

The system is composed of a light sensor: LCD, a resistor 10k Ohms, a LED, a resistor 270 Ohms, a breadboard and an Arduino Yun.



Picture 6: *Light sensor*

CODE

The code is started by choosing the analogue pin A2 for the LDR. The pin connected to the LED is number 9 as digital PWM for analogic value instead of digital value. The LED pin is defined as output, and the LDR pin is defined as input. The values are read and printed.

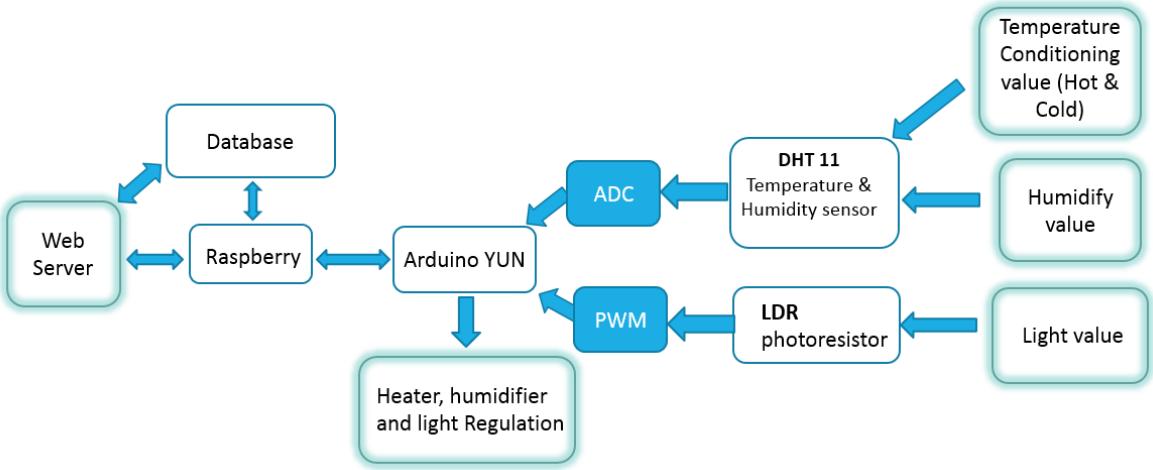
The LDR value is between 0 and 1023 depending on how luminated the room is. When there is a lot of light, we will have a high number closer to 1000. On the contrary, when there is no light, the value will be close to 0. We established that the average value is close to 400. So, when the value is under this value, the LED will shine. And when it is above, the LED will shut down.

To vary the light intensity of the LED, we can't directly use the LDR output because this number will be high and would therefore also lead to a high voltage and a more illuminated light. When the room is dark, we will have a low number and should therefore use a high number to illuminate the led. In order to do that, we subtract the LDR value to itself and we will get this connection.

```
int LED = 9; // digital PWM like analog
int LDR = A2;

void setup() {
  Serial.begin(9600);
  pinMode(LED, OUTPUT);
  pinMode(LDR, INPUT);
}

void loop() {

  int LDRvalue = analogRead(LDR);

  if(LDRvalue <= 400)
  {
    //digitalWrite(LED, HIGH);
    analogWrite(LED, (1023 - LDRvalue));
    Serial.print(" it's dark here!! we need some light \n");
    Serial.println(LDRvalue);
    }
    else
    {
      //digitalWrite(LED, LOW);
      analogWrite(LED, (1023 - LDRvalue));
      Serial.print(" it's bright here!! we need some dark \n");
      Serial.println(LDRvalue);
      }

  delay(1000);
}
```

Picture 7: *Light sensor code*

III. Web server



Picture 1: *General Diagram*

PURPOSE

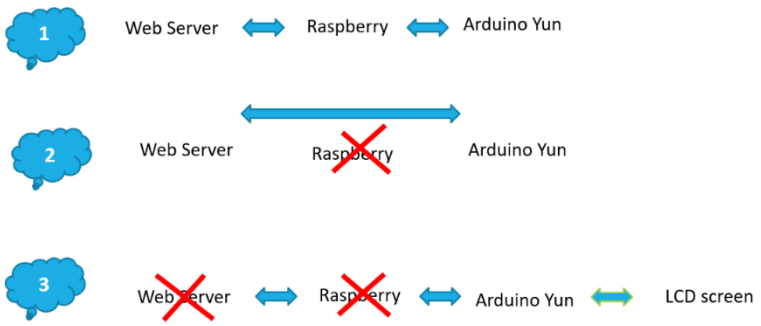
The webserver allows to take all the information of the sensors and show them in a webpage on Internet. That would be interesting to see the average values in the greenhouse, if they are in a correct range or not, that would signify that we can face some problem about our electronic components.
This knowledge can be taken at distance; what if the owner is busy and cannot be next to its greenhouse? So that it is interesting to get to know of its situation wherever the person is.

The greenhouse in that way can be monitored to see if all the plants get the right conditions to grow.

OPERATION

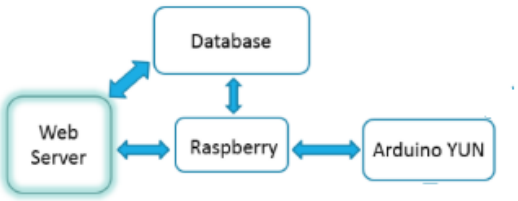
As our initial plan about the webserver could be difficult to lead, we decided to make several situations to make our webserver differently, or in worse case, use some LCD screen to display our data.
So here are 3 situations. Each situation represents some theoretical steps we wanted to follow.

The two situations would retrieve the data of each sensors (Humidity, temperature, lighting) and the last one retrieve only humidity and temperature values.



Picture 8.1: Presented Situations

a) Situation 1



***Picture 8.2**: Webserver on raspberry*

The raspberry must maintain the webserver through PHP page. We must make an account so that a page would be dedicated to us. The same with MySQL Database; a page with database is allocated to the account created.

Arduino Yun board pushes the data on Raspberry thanks to serial communication; then the data must be stocked on MySQL Database.

In the purpose of retrieving the data from Arduino, we need on Raspberry to:

-get access to the Arduino data

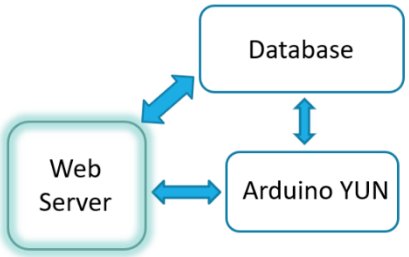
-connect on MySQL database space editing the right credentials

-create the variables on MySQL that can stock all the data coming from the Arduino, then create a table on MySQL

-insert in this table the Arduino data, through SQL code

--> Those are the steps to print the data on Raspberry screen

b) Situation 2



***Picture 8.3:** Webserver on Arduino*

The webserver can also be maintained by the Arduino, so that means that the Arduino is not only getting data from the sensor, it can also push the data into a database and make a webserver.

It has the same idea of situation 1, this is to say push data on MySQL database, then create some code on php to make it presentable on a webpage.

c) Our trials about Situation 1

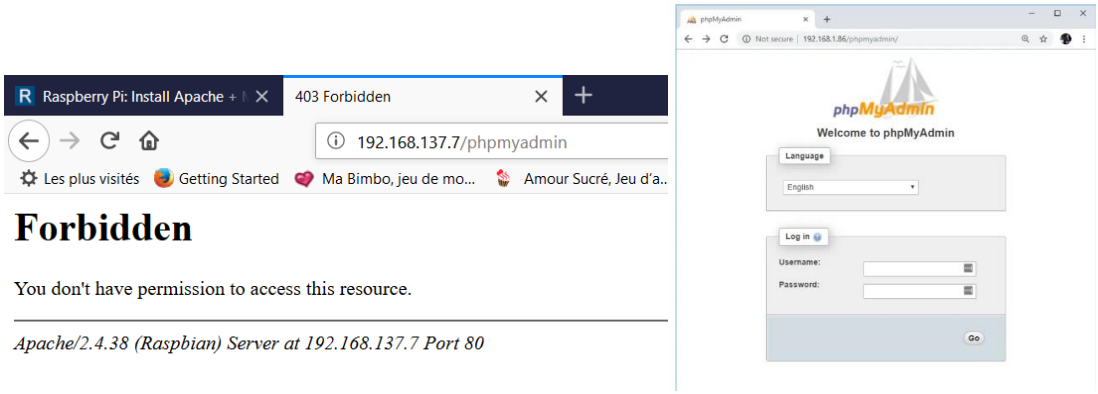
The webserver was initially to be implemented with the Raspberry. We tried hard to install the LAMP server (PHP, MariaDB, Apache, PhpMyAdmin).
So we managed to install everything except PhpMyAdmin, which handles the MySQL database in a web interface written in PHP.

The problem was due to a right access to the files, so we managed to follow different tutorials:
- Once was to configure Apache, writing in its configuration file, without success.

- Another was to move the phpMyAdmin file into */var/www/html*
- Another one was to grant all privileges to php user.

Even we tried to do so, nothing expected worked.

Finally, we got no pages shown while entering our raspberry address following phpMyAdmin on the toolbar:



Picture 9: *Blank page got instead (left) of the credentials phpMyAdmin page (right)*

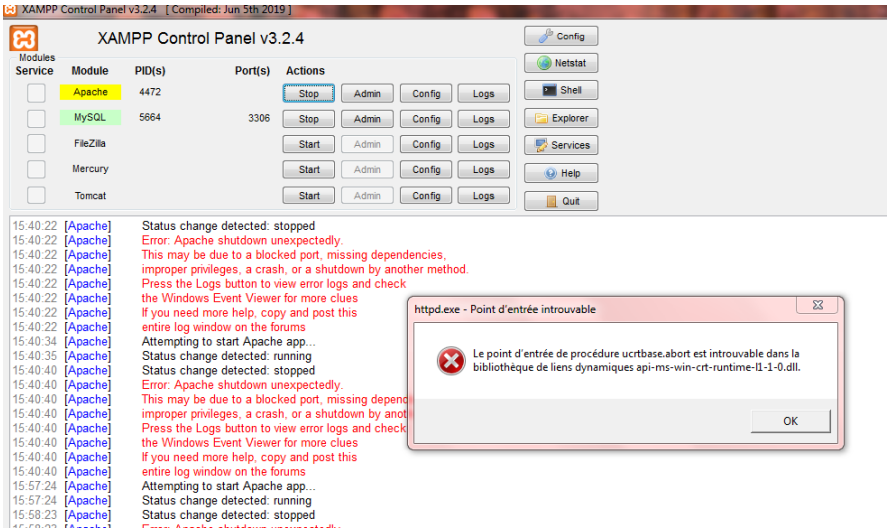
Seeing that, we tried **situation 2**.

d) Our trials about Situation 2

So, we went through the XAMPP which is an alternative of the Lamp server.
It has multiple packages downloaded once.

Picture 10: *Installation on XAMPP*

The installation went well except for Apache, it blocked apparently because of a missing file.



Picture 11: *Apache not running*

The computer running the XAMPP server is a Windows 7; I tried multiple ways to solve it

- Make the updates for it as recommended for *api-ms-win-crt-runtime* file
- Downloading package on Microsoft reference page especially for Windows 7
- Taking the file from another computer with Windows 10 and put it in Windows/System32
- Trying to install Visual Studio 2015

All these steps couldn't make the Apache port work.
So we finally decided to go on **Situation 3**.

e) Our trials about Situation 3

Picture 11.2: *Situation 3*

The idea is to link an LCD screen to DHT11 sensor, so that we can show the values on the screen.
In order to do that, we change the code of the DHT11 and the breadboard system.
For the code, the DHT11 part is still the same. Here the new lines add to it:

CODE

```
#include<LiquidCrystal.h>
#include <DHT.h>

int DHT11pin=A0;
//Initialisation de l'écran LCD avec les numéros des pins utilisés
LiquidCrystal LCD(12, 11, 4, 3, 2, 1);

int Heating=5;
int Cooling=6;
int MoreMoist=7;
int MoreDry=8;
float DHTTemp=0; // temp mesurée
float DHTHum=0; // pourcentage humidité

dht DHT ; // initialisation du capteur

void setup() {
  Serial.begin(9600);
  LCD.begin(16,2); // nb de colonnes et lignes de l'ecran
  LCD.clear();

  pinMode(Heating, OUTPUT);
  pinMode(Cooling, OUTPUT);
  pinMode(MoreMoist, OUTPUT);
  pinMode(MoreDry, OUTPUT);
  pinMode(DHT11pin, INPUT);
  delay(1000);
}

void loop() {
  int LDRvalue = analogRead(LDR);

   DHT.read11(DHT11pin);
   DHTTemp=(DHT.temperature);
   DHTHum=(DHT.humidity);

  // PARTIE LCD
  if(isnan(DHTTemp) || isnan(DHTHum)) // si les valeurs retournées ne sont pas des nb
  {
    LCD.setCursor(0, 1); //Positionnement du curseur
    LCD.print(DHTTYPE); //On affiche le type de capteur
    LCD.setCursor(5, 1);
    LCD.print(" illisible"); //On affiche l'erreur
  }
  else
  {
    //mise en forme et affichage des informations sur l'écran LCD
    LCD.setCursor(0, 0); //Positionnement du curseur
    LCD.print("Degres : ");
    LCD.setCursor(9, 0);
    LCD.print(DHTTemp); //Affichage de la température
    LCD.setCursor(13, 0);
    LCD.print((char)223); //Affiche le caractère ° (degrés)
    LCD.setCursor(14, 0);
    LCD.print("C"); //En degrés Celsuis
    LCD.setCursor(0, 1);
    LCD.print("Humidite : ");
    LCD.setCursor(11, 1);
    LCD.print(DHTHum); //Affichage de l'humidité
    LCD.setCursor(15, 1);
    LCD.print("%");
  }
  delay(5000);
```

Picture 12: *LCD code with Arduino*

The code is started by including an LCD library add to the DHT library. The initialization of the LCD is chosen as digital pins. All the pins are defined as outputs, except for the analogue pin connected to the DHT11. This one is defined as input. The values are read and printed. Also, we define the number of row and columns of the LCD we use. Here it's 2 rows and 16 columns.

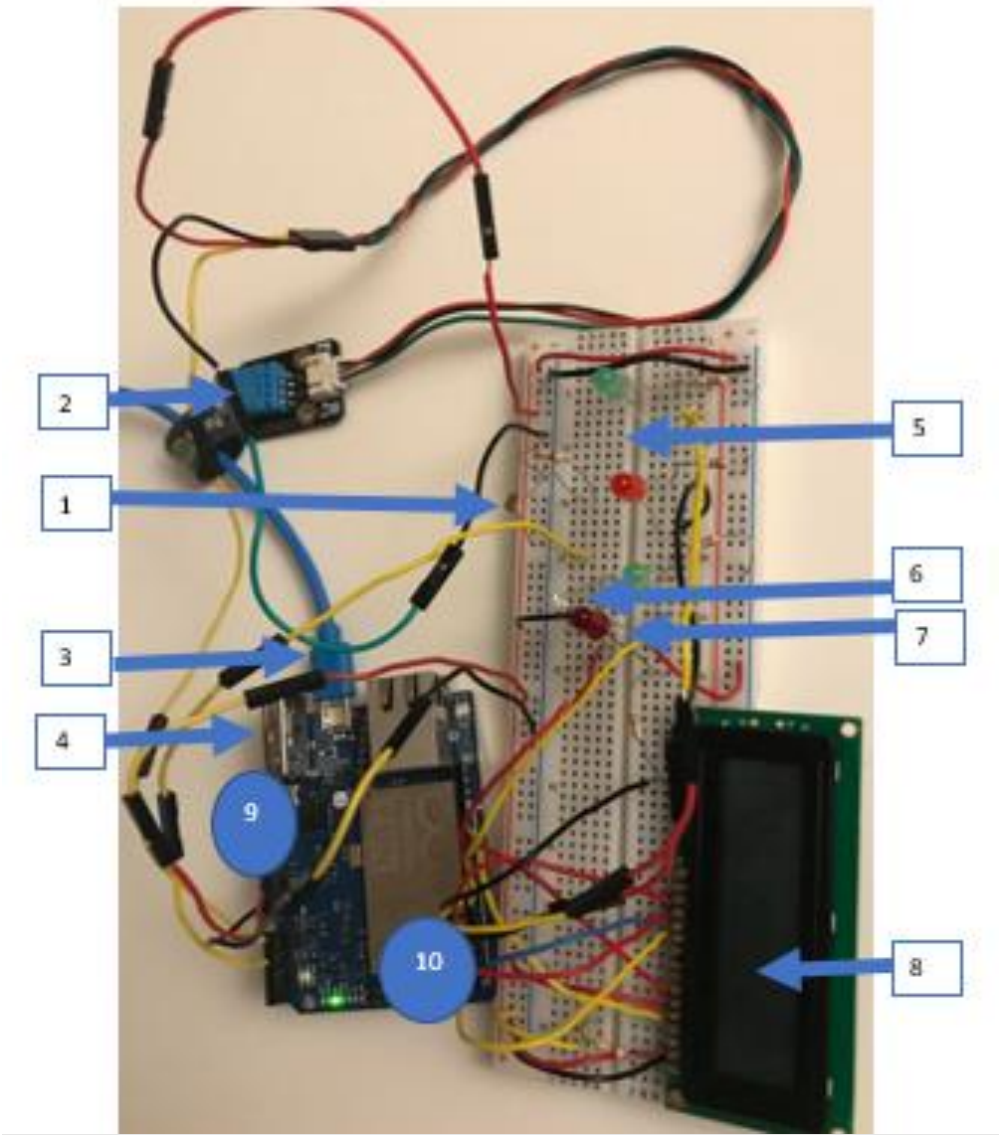
In the loop function, there 2 part. The part with the DHT11 and the LEDs system, and the part with the LCD. Here we only put the part with the LCD because the code with the DHT11 is in the previous page.

We look for the value take from the DHT11, and if the return value aren't numbers, we put an error on the LCD screen, if not, we format and display information on the LCD screen.

At the end, the screen will change every 5 seconds.

In conclusion, we run this situation with success.

IV. General operating system



Picture 13: *Global system*

Legend: 1st part:

1	LDR: Light sensor 3 ports: Ground, 5V, and analog A2	5	LEDs for Temperature
2	DHT11: Air Humidity & Temperature sensor 3.ports: Ground, 5V, and: →Digital pin 12 (+ Humidity) / Digital port 13 (+ dry) →Digital pin 3 (+ cool) / Digital port 2 (+ warm)	6	LED of LDR
3	Serial connection between raspberry and Arduino	7	LEDs for Air humidity
4	Arduino board	8	LCD

Legend 2nd part: Arduino

9	Digital part of ARDUINO Yun Link to DHT11 & LCD	10	Analog in part of ARDUINO Yun Link to LDR

GLOBAL CODE :

```
#include<LiquidCrystal.h>
#include <DHT.h>

int LED = 9;
int LDR = A2;

int DHT11pin=A0;
LiquidCrystal LCD(12, 11, 4, 3, 2, 1); //Initialisation de l'écran LCD avec les numéros des pins utilisés

int Heating=5;
int Cooling=6;
int MoreMoist=7;
int MoreDry=8;

float DHTTemp=0; // temp mesurée
float DHTHum=0; // pourcentage humidité

dht DHT ; // initialisation du capteur

void setup() {

  Serial.begin(9600);
  LCD.begin(16,2); // nb de colonnes et lignes de l'ecran
  LCD.clear();

  pinMode(Heating, OUTPUT);
  pinMode(Cooling, OUTPUT);
  pinMode(MoreMoist, OUTPUT);
  pinMode(MoreDry, OUTPUT);
  pinMode(DHT11pin, INPUT);

  pinMode(LED, OUTPUT);
  pinMode(LDR, INPUT);
  delay(1000);
}
```

```
void loop() {

  int LDRvalue = analogRead(LDR);

   DHT.read11(DHT11pin);
   DHTTemp=(DHT.temperature);
   DHTHum=(DHT.humidity);
   Serial.print("DHT11 Temperature measured as:");
   Serial.println(DHT.temperature);
   Serial.print("DHT11 Humidity measured as:");
   Serial.println(DHT.humidity);

  // PARTIE LCD
  if(isnan(DHTTemp) || isnan(DHTHum)) // si les valeurs retournées ne sont pas des nb
  {
    LCD.setCursor(0, 1); //Positionnement du curseur
    LCD.print(DHTTYPE); //On affiche le type de capteur
    LCD.setCursor(5, 1);
    LCD.print(" illisible"); //On affiche l'erreur
  }
  else
  {
    //mise en forme et affichage des informations sur l'écran LCD
    LCD.setCursor(0, 0); //Positionnement du curseur
    LCD.print("Degres : ");
    LCD.setCursor(9, 0);
    LCD.print(DHTTemp); //Affichage de la température
    LCD.setCursor(13, 0);
    LCD.print((char)223); //Affiche le caractère ° (degrés)
    LCD.setCursor(14, 0);
    LCD.print("C"); //En degrés Celsuis
    LCD.setCursor(0, 1);
    LCD.print("Humidite : ");
    LCD.setCursor(11, 1);
    LCD.print(DHTHum); //Affichage de l'humidité
```

```
    LCD.setCursor(11, 1);
    LCD.print(DHTHum); //Affichage de l'humidité
    LCD.setCursor(15, 1);
    LCD.print("%");
  }
  delay(5000);

  // PARTIE LDR
  if(LDRvalue <= 400)
  {
    //analogWrite(LED, HIGH);
    analogWrite(LED, (1023-LDRvalue));
    Serial.print(" it's dark here!! we need some light \n");
    Serial.println(LDRvalue);
  }
  else
    {
      //analogWrite(LED, LOW);
      analogWrite(LED, (1023-LDRvalue));
      Serial.print(" it's bright here!! we need some dark \n");
      Serial.println(LDRvalue);
   }
```

```
// PARTIE DHT11
if (DHTTemp < 16) {
 digitalWrite(Heating, HIGH);
 delay (1000);
 digitalWrite(Cooling, LOW);
 delay (1000);
}

if (DHTTemp > 22) {
 digitalWrite(Heating, LOW);
 delay (1000);
 digitalWrite(Cooling, HIGH);
 delay (1000);
}

if (DHTHum < 41) {
 digitalWrite(MoreMoist, HIGH);
 delay (1000);
 digitalWrite(MoreDry, LOW);
 delay (1000);
}

if (DHTTemp > 80) {
 digitalWrite(MoreMoist, LOW);
 delay (1000);
 digitalWrite(MoreDry, HIGH);
 delay (1000);
}


 delay (500);
}
```

Picture 14: *Global code*

IMPLEMENTATION OF DHT11, LDR AND LCD:

By combining the codes for sampling and actions we can take temperature, humidity and lighting into account using one code and one board. In this way we make a prototype for a connected system that can function in a greenhouse. The code is simply made by merging the two codes described earlier in this report together into one.

CONCLUSION

I. System's limits

- Web server part is not finished

II. Perspectives of improvement

We have some perspectives of improvements:

- Sending mails with the occurrences of triggers to change temperature, humidity and LDR+ average values of the week.
- Replace the LEDs with real equipment's (water pump...)

III. Conclusion

We were more comfortable with the electronic part, however, the issues with webserver penalized us a lot. The fact is that once you blocked on the installation process, it can be tricky to continue. The LEDs allowed us to model our greenhouse very easily, it would be nice to go further in this kind of project and making it into a long term one to answer to the POC in a more definitive and completed way.

IV. Sources

WEBSITE :

Webserver: Arduino

https://www.youtube.com/watch?v=wSSTSGYk-m0&t=162s

Webserver: Raspberry

https://randomnerdtutorials.com/raspberry-pi-apache-mysql-php-lamp-server/

OTHERS:

TP1-Arduino-Introduction

TP2-Raspberry-Introduction