**Farm Monitoring**

**and**

**Control System**

Table of Contents

[Definition, investigation and analysis 3](#_Toc448662692)

[Description of the client: 3](#_Toc448662693)

[Methods currently in use: 3](#_Toc448662694)

[Origin of the data: 3](#_Toc448662695)

[Interviews: 3](#_Toc448662696)

[Interview #1: 3](#_Toc448662697)

[Objectives determined after interview #1: 4](#_Toc448662698)

[Interview #2: 5](#_Toc448662699)

[Objectives determined after interview #2: 5](#_Toc448662700)

[Required Specifications 6](#_Toc448662701)

[Inputs: 6](#_Toc448662702)

[Processing: 6](#_Toc448662703)

[Storage: 7](#_Toc448662704)

[Outputs: 7](#_Toc448662705)

[Alternative Approaches: 7](#_Toc448662706)

[Hardware requirements: 8](#_Toc448662707)

[Software requirements: 9](#_Toc448662708)

[Client Agreement 10](#_Toc448662709)

[Design: 11](#_Toc448662710)

[Objectives: 11](#_Toc448662711)

[Intended benefits: 13](#_Toc448662712)

[Limitations: 13](#_Toc448662713)

[Program flowchart 14](#_Toc448662714)

[Wireframe: 15](#_Toc448662715)

[Database structure: 16](#_Toc448662716)

[I2C transition table 18](#_Toc448662717)

[Testing Plan: 19](#_Toc448662718)

[Client Agreement 22](#_Toc448662719)

[Implementation: 24](#_Toc448662720)

[Testing: 25](#_Toc448662721)

[Implementation Method: 37](#_Toc448662736)

[Client Agreement 38](#_Toc448662737)

[Evaluation: 39](#_Toc448662738)

[Client’s response to the new system: 41](#_Toc448662739)

[Appendix 1 – Arduino Code: 42](#_Toc448662740)

[Appendix 2 Raspberry Pi Code: 47](#_Toc448662741)

[Appendix 2.1 farm\_project.py: 47](#_Toc448662742)

[Appendix 2.2 form.php 52](#_Toc448662743)

[Appendix 2.3 formhandler.php: 65](#_Toc448662744)

[Appendix 2.4 ajax.js: 66](#_Toc448662745)

[Installation: 67](#_Toc448662746)

[Writing the firmware to the Arduino: 67](#_Toc448662747)

[Preparing the Raspberry Pi: 68](#_Toc448662748)

[Circuit: 69](#_Toc448662749)

[I2c configuration: 70](#_Toc448662750)

[Network: 71](#_Toc448662751)

[Operation guide: 72](#_Toc448662752)

[Starting and rebooting the system: 72](#_Toc448662753)

[Inputting the number of eggs: 72](#_Toc448662754)

[Changing the thresholds for the heater: 72](#_Toc448662755)

[Saving a graph: 73](#_Toc448662756)

[Glossary: 73](#_Toc448662757)

# Definition, investigation and analysis

## Description of the client:

The client is 62-year-old lady living in Peris, Ilfov, Romania, named Nuta Cotovanu. She has a medium to large size garden and chicken farm that is for her and her family’s consumption only and the only people that work on it are her husband and her. He has a back problem which makes lifting and carrying the cereals for the animals quite painful.

She currently owns around 100 grown and fledgling chicken which live on 60 square meters of her land.

The fledging birds need to have an incandescent lamp turned on every night to keep them warm.

## Methods currently in use:

Someone has to periodically check and add to the level of food and water in the feeding trays and check the main supply of cereal and order more when it is running low.

They check for eggs every morning and afternoon.

The lamp for the fledging chickens has to be manually plugged into a wall socket every sunset and then pulled out at sunrise.

## Origin of the data:

The number of eggs is monitored in a notebook. A visual check is done every day on the amount of food left in the bag. Temperature, water consumption and weather are not currently monitored.

The heating element inside of the coop is manually turned on/off, which sometimes makes the coop too hot in the morning.

## Interviews:

### Interview #1:

#### General questions:

1. What part of your system would you upgrade? – I would like to have a better way of monitoring how much food my birds consume and how many eggs they produce. I also have some fledging birds which require me to turn on a heating light for them when temperatures are too low.
2. What are some of the problems with the current system? – I keep losing the slips of paper where I write how many eggs I have or when I last bought grains. I would also like to know how much water they drink and how the weather affects them. I would also prefer a more visual representation of the data. Turning the light for the little ones is also a hassle.
3. How advanced is your use of a computer? – Fairly limited. I know how to turn on and off the computer and access the internet.

3b.Who else is going to be using this system? – Probably, only me and my son when he comes visiting.

1. What operating system do you use? – Microsoft windows, I believe.
2. Do you have an internet connection? – Yes a do. Dial-up I believe.

5b.If yes, do you have a wireless router? – I do have one.

1. What kind of data are you interested in receiving from your new system? – Temperature, humidity, intensity of the light, how much water has been used, how much food there is left,
2. In what form would you like the data to be presented? (Graphs, charts, records etc.) – Whatever you think is more adequate.
3. What is your budget? – I think 500 euros should cover it. In any case, I could spend 200 more, just to get a better result.
4. When is the deadline for the finalization and implementation of this new system? – I would like to have it by the end of September 2016.

#### Specific questions:

1. What kind of food do you feed your animals with? (Cereal, vegetables etc.) – I feed them both, but I am only interested in the amount of cereal.
2. How frequently do you feed them? – Twice, maybe 3 times a day. I make sure they always have enough food in their trays.
3. Do you feed them differently in relation to the season? – They eat slightly less in the winter, but they also do not produce as many eggs.
4. Do the fledging birds have a different diet? - No
5. How many food trays do you need? – Only one that is about 1 meter long.
6. How much water do the birds drink? – I have no idea, I just keep their tray filled and maybe change it when it is too hot or it froze.
7. Does the temperature of the water matter? – It should not be very warm, but generally, it does not matter that much.
8. Would you like the water to be constantly refiled as it drops below a certain level or periodically to make sure that it stays at a certain temperature? – You can decide which is more economical. As I said, the temperature is not very important.
9. Do you need any video monitoring device? – Not really, I still want to check on them sometimes.
10. Would you like to have temperature sensors in different parts of the coop? – Yes, one outside the coop and one inside.
11. What kind of climate control would you like? – A heating lamp should do it.

#### User interface questions:

1. Do you want a monitoring system or a control system? – I would like If possible, for the system to be autonomous, but still allow me to change the thresholds, like what is too cold or too hot.
2. Would you prefer a printed or digital report? – I do not have a printer, so a digital one should do.
3. How often would you like to receive the report? - I would like to check it every day and see the numbers from the last month or week.
4. What kind of platform are most familiar with: Apps or Web Sites? – Mostly Web sites.
5. Would you like to share your statistics on social media? – If it is possible, sure, but I’m not keen on it.
6. How much time would you like to store the data for? – It’s enough if I can compare this year with my previous year.
7. Do you have a preference between cloud storage and local storage? – Local is fine.

### Objectives determined after interview #1:

-Food consumption monitoring

-Egg-count tables/graphs

-Control of temperature

-Water consumption monitoring

-Weather monitoring (humidity and light intensity)

-Visual representation of the values being monitored

-User friendly interface

-Budget under 500 euros

-Automatic refill of water trays

-Temperature monitoring

-Provide daily and monthly illustrations of the data as well as a comparison between this year and the previous one

- (Optional) provide the ability to share online

### Interview #2:

-Since the last time we have met, I have familiarized myself with the types of sensors that the system would require. Most of the measurements you request can be done very accurately and modestly precise within the budget. The only sensor that we might have a problem with is the weight sensor for the food monitoring part of the system. The problem is that there are no electronic weight sensors that can support this kind of weight and still give precise measurements and be modestly priced.

-I see. Is there any solution to this problem?

-By using a pressure sensor underneath the bag we might be able to roughly estimate how much food is left in it. Because the sensor gives an analogue value that is proportional to the square of the weight, we cannot easily map the weight to output value, but we can map it to some values that would indicate if the bag is full, almost full, half-full etc. What do you think of this idea?

-Well, in hindsight, I don’t need to know exactly how much they eat, but it would be nice to not have to go check how much is left in the bag. My husband sometimes feed them before he leaves home to go to work, so not having to go out to check if I have to buy more grains is very convenient.

-Also, this method is not adequate for estimating how much food was consumed as the uncertainty is far too great. How important it is for you to know the relationship between food and other measurements?

-It would be nice to know this, but if it is too complicated to implement and the data is not really that precise, than there is no point in wasting time trying to make it work.

-I am glad we managed to solve this problem. I think I have everything I need to start making a mockup of what the system would look like and a list of components needed to make it. I think I can finish them in 2 weeks’ time and we can meet then and see if you like the design.

-Great!

### Objectives determined after interview #2:

~~-Food consumption monitoring~~  Food level monitoring

-Egg-count tables/graphs

-Control of temperature

-Water consumption monitoring

-Weather monitoring (humidity and light intensity)

-Visual representation of the values being monitored

-User friendly interface

-Budget under 500 euros

-Automatic refill of water trays

-Temperature monitoring

-Provide daily and monthly illustrations of the data as well as a comparison between this year and the previous one

- (Optional) provide the ability to share statistics online

## Required Specifications

### Inputs:

Temperature sensor inside the coop

– Digital signal

- Range: -40 C to 125 C.

Temperature sensor outside the coop

– Analog signal (more rugged for outdoor use)

- Range: -50 C to 125 C.

Humidity sensor

– Digital signal

- Range: 0% to 100%

Water level – a vertical pipe that goes out of the bottom of the water tray and had a floater in it. When the floater drops below a certain level, a light-beam-interrupt sensor will send a signal to the microcontroller.

Food level – pressure sensor under the food tray detects when the tray is lighter, therefore has less food.

- Analog signal

Light sensor – Simple analogue LDR (light dependent resistor).

Number of eggs – manual input from the user, every day.

Manual adjustments from the user interface for the threshold values of the control system.

Amount of food left in the bag – pressure sensor under it.

- Analog signal

Internet connectivity for getting the time (cheaper than using a real-time clock module).

Water used – flow meter – digital reading

### Processing:

All the data gathering will be done by a microcontroller as it offers real-time monitoring and low level programming. This part of the code will be done in C++.

After all the sensors are read, the microcontroller decides if any control unit needs to be adjusted. The final measurements are sent over the I2C protocol to a raspberry pi.

The raspberry pi is full Linux computer that uses very little power and requires no fans, making it perfect for an environment like a coop. The raspberry pi acts like a server for the web page user interface from which the system can be controlled and monitored. The raspberry pi also saves and organizes all the data. All the code for that will be done in Python and PHP.

### Storage:

All the data received by the raspberry pi is saved on an external 8 GB USB flash drive as it provides enough storage for the system and it is more robust and uses less power than an HDD.

### Outputs:

The output from the database will be displayed on a web site hosted on the Raspberry Pi. This web site will be accessed from the desktop pc and the following sets of data will be shown:

* Temperature, current and a graph with readings over time, in and outside of the coop
* Humidity, current and a graph with readings over time
* Number of eggs graph against temperature, humidity or food consumption
* Amount of food and water, current and a graph with readings over time
* Alert to buy a new bag of grains
* Light intensity

### Alternative Approaches:

**Alternative Approach 1:**

A solution for my client would be using an off the shelf software to create and manage a database and manually input all the measurements.

**Pros:**

This is the easiest system to implement.

The database would be easier to upgrade in the future in terms of features and it would be very easy to port to another machine.

Graphs would also be easy to create in a report.

**Cons:**

Requires the most involvement from the user

This method would also add extra costs that would make the project go over the budget. (Ex: Microsoft Access alone costs 150$)

There is no way to display live measurements.

**Reasons for Rejection:**

This solution requires too much involvement from the user to be a suitable replacement of the current system which has been working so far due to its simplicity. Cost is another major drawback of this solution as the budget doesn’t cover the cost of a decent database management package.

Even if the client decides that live measurements are not mandatory, the previous arguments for rejection still stand and make this solution undesirable.

**Alternative Approach 2:**

Another approach would be using the desktop computer, with a custom software, that is connected directly to the microcontroller.

**Pros:**

Fairly easy to implement.

Easy to port over other computers.

It is very easy to add features during the development stage.

**Cons:**

Requires a USB cable to be wired over a long distance from the PC to the microcontroller. This poses as a problem, as long cables are notorious for picking up noise and dropping out signals because of the resistance of the wire and causing corruption of the data.

There is no possibility of adding features after the program has been compiled without recompiling and reinstalling the system.

**Reasons for Rejection:**

This method would be a great choice if not for the necessity of a long USB cable between the computer and the microcontroller which may affect the integrity of the data, it is almost impossible to troubleshoot and poses as a point of failure of the system after a long period of time.

**Why I chose my approach:**

Using this approach, the whole system is very well organized as it uses wireless networking to communicate between the system and the user. This eliminates the need for long cables which are to troubleshoot or maintain and makes the system easier to use. Furthermore, the system is very modular as you can access the web server with any device that has a browser and is connected to the LAN. This system also requires the least amount of user interaction as the whole control system is autonomous. A wireless implementation is also cheaper as the labor required for wiring up such a system would be very expensive and time consuming.

### Hardware requirements:

Desktop computer to access the web server (core 2 duo, 2 GB ram, 80 GB HDD) - provided by the client

Wireless router - already installed

Raspberry pi model b – 35$

8GB USB flash drive -5$

Arduino Nano microcontroller – 29$

1 analogue temperature sensor – 2$

1 digital temperature and humidity sensor - 4$

2 relay boards – 10$

LDR sensor – 1$

Wi-Fi USB dongle – 10$

1 pressure sensor – 8$

Water flow-meter – 10$

Light beam interrupt sensor – 2$

Custom built case for the raspberry pi, the Arduino and the power supply. – 15$

### Software requirements:

Python, PHP, Apache 2 and I2C libraries running of the Raspberry pi

Raspbian OS running on the Raspberry pi

Linux mint running on the desktop pc with a web browser installed (google chrome or any other web-kit based browsers are recommended).

# Client Agreement

By signing this document, I confirm that I accept the following list of objectives for the new system and that I agree to proceed to the design stage of the new system.

**The system will meet the following objectives:**

-Food level monitoring

-Egg-count tables/graphs

-Control temperature

-Water consumption monitoring

-Weather monitoring (humidity and light intensity)

-Visual representation of the values being monitored

-User friendly interface

-Automatic refill of water trays

-Temperature monitoring

-Provide daily and monthly illustrations of the data as well as a comparison between this year and the previous one

-Provide the ability to share statistics online

**Signature of client: Signature of provider:**

# Design:

## Objectives:

**Food level monitoring**

There will be a pressure sensor, in the form of a force-sensitive resistor, which will be placed under the bag of grains and it will convert force into a potential difference (voltage) across the input pin of the microcontroller.

**Control of temperature**

A temperature sensor will be placed in the coop and a heater will be connected to a relay which will be controlled by the microcontroller. The thresholds for which the heater will work will be set by the user in the website.

**Water consumption monitoring**

A water flowmeter will be connected to the microcontroller and the data will be sent to the Raspberry Pi to be stored and displayed.

**Weather monitoring (humidity and light intensity)**

A humidity sensor and a Light-Dependent-Resistor will provide the necessary information about the weather condition.

**Visual representation of the values being monitored**

Graphs will display the data stored in the database. Highcharts is a JavaScript library for graphs that displays the data in a website. I chose it because it is very easy to implement, yet it provides beautiful results.

\*Note: Highcharts requires an internet connection to work.

**User friendly interface**

To achieve this, the web site will have large buttons and a minimalistic design.

**Egg-count tables/graphs**

An input box will be paced at the top of the website, where the user can input the number of eggs collected that day. The input box will keep displaying the egg-count throughout the day in case the user wants to add more eggs.

**Automatic refill of water trays**

A light gate with a floater in a tube will be placed on the side of the water tray. As the water level goes down, the floater, goes down which makes the beam to get to the receiver connected to the microcontroller. If the beam is received, a water valve will open and the trays will be filled with water until the beam is broken again.

**Temperature monitoring**

Temperature sensors will be placed inside and outside the coop. They will be connected to the microcontroller and the data will be sent to the Raspberry Pi to be stored and displayed.

**Budget under 500 euros**

An estimate of the cost of the system would be around 150 euros. That doesn’t include the cost of implementation and development fees.

|  |  |
| --- | --- |
| **Item** | **Price** |
| Raspberry Pi | $40.00 |
| 8GB USB drive | $5.00 |
| Arduino Nano | $28.99 |
| TMP36 | $1.50 |
| DHT11 | $9.95 |
| 5v Relay boards | $9.21 |
| LDR | $1.00 |
| Wi-fi dongle | $9.99 |
| pressure sensor | $7.95 |
| flow meter | $9.95 |
| light beam | $1.95 |
| Power supplies | $12.55 |
| wires | $4.99 |
| SD card | $6.20 |
| Software | $0.00 |
| **total** | $149.23 |
|  |  |
| 1$ = | $0.89 |
| **converted** | € 132.81 |

|  |
| --- |
| **Installation fee:** |
| € 200.00 |

**Provide daily and monthly illustrations of the data as well as a comparison between this year and the previous one**

The database will have tables storing the averages per month of the current year and the previous year.

**(Optional) provide the ability to share online**

Highcharts has a button on every chart to save it as an image.

## Intended benefits:

* Save time. The new system will save time because the user will not have to go and check on the water trays. She will also not have to turn on/off the heater for the fledging birds.
* Accuracy. The heater will provide autonomous and accurate control of the temperature inside the coop. Furthermore, my solution provides accurate reading of the weather and conditions inside and outside of the coop.
* Allows data analysis. Because of the digital nature of the solution, it is very easy to compare egg production against other factors and get insight into how the farm can be improved.

## Limitations:

Due to limited storage, the system will only record average values for each day throughout the year. Furthermore, the system will only store data for the current year and the previous year for comparison.

Another limitation is the fact that the Web interface may not scale properly on some devices like phones or high resolution monitors.

The estimated file size of each table is the following:

Monthly table: \*26 bytes per record \* 356 days = 9.5 kB

Daily table: \*19 bytes per record \* 1440 minutes in a day = 27.4 kB

Yearly table: \*31 bytes per record \* 12 months = 0.4 kB

Thresholds table: \*11 bytes

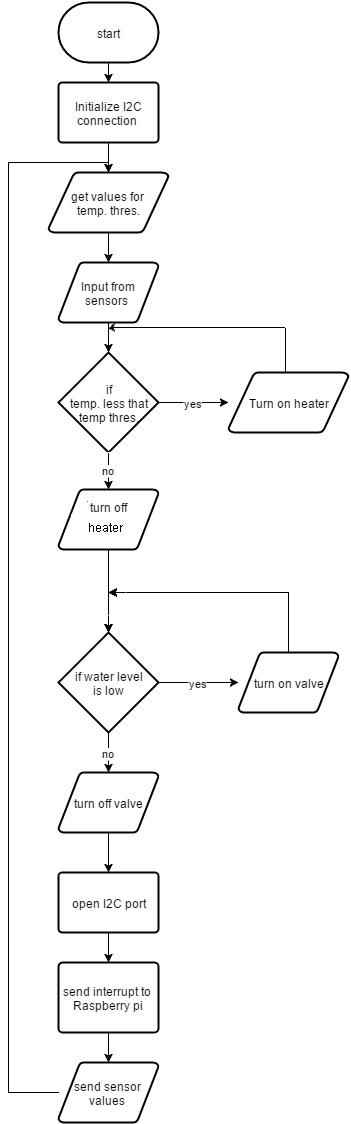
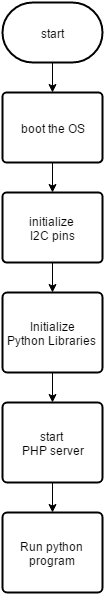
\*A detailed table of the size of each field can be found in Design > Database structure.

Lastly, the system will require a specialist to add features or change the design of the interface, but this does not have a big impact on my client.

## Raspberry pi Python flowchartProgram flowchart

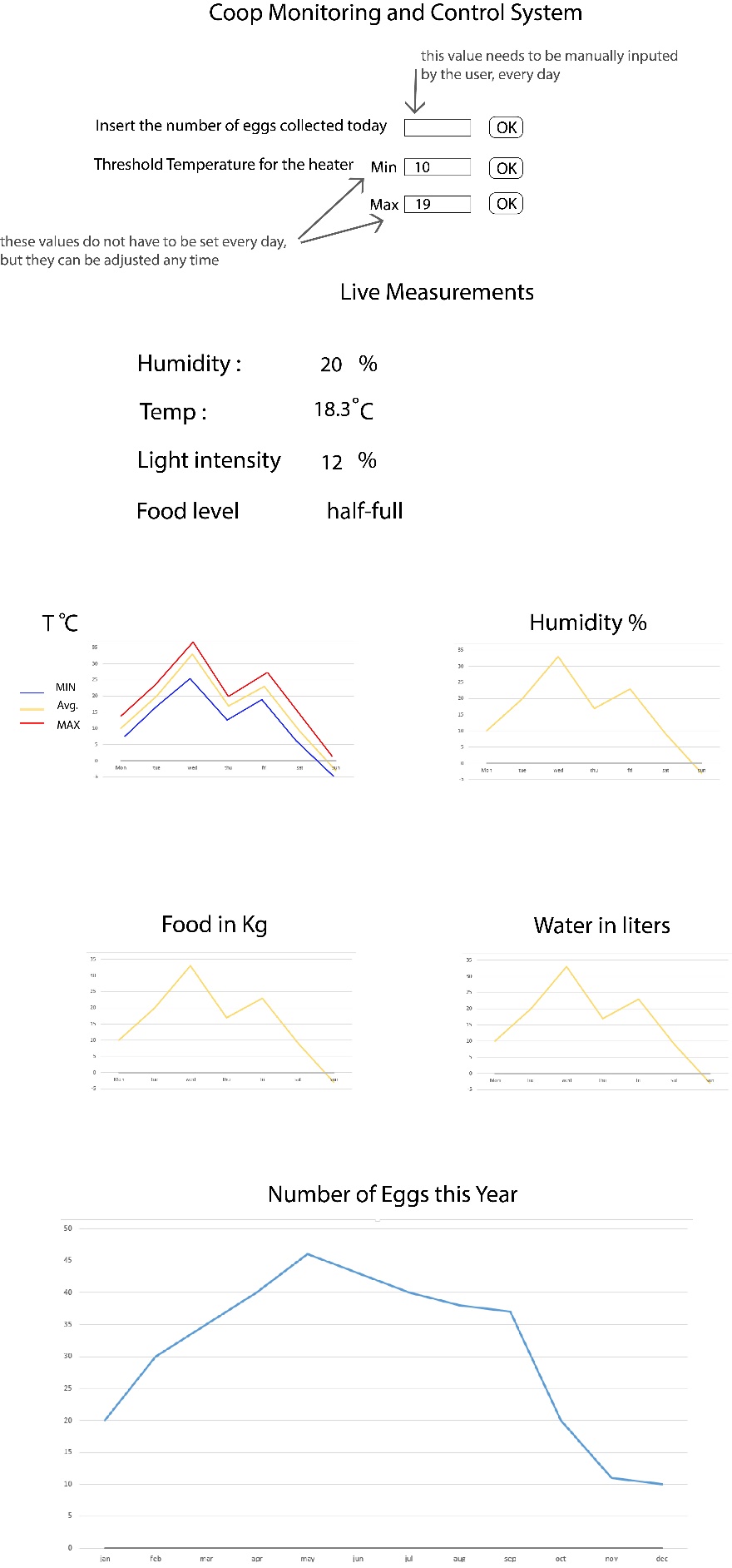
This shows the autonomous heating and water tray filling system flowchart

This shows the database management flowchart



This shows the boot sequence

## Wireframe:



This is a mockup of the website.

Some graphs may be adjusted, some may be added and some may be merged to ensure a better representation and illustration of the data being shown.

## Database structure:

Monthly Table (one record every day)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field Name | Type | Length  /bytes | Validation | Description | Example |
| Date | String | 8 | Length check | Date in the format ddmmyyyy | 13012016 |
| Food\_Used | Float | 3 | Presence check | How much food was used this day | 4.8 |
| Water\_used | Float | 3 | Presence check | How much water was used this day | 6.2 |
| Humidity | Integer | 1 | Range check  0-100% | Humidity measured | 44 |
| Temp\_Min | Float | 3 | Range check  -40C – 25C | Minimum temperature measured today | -13.8 |
| Temp\_Max | Float | 3 | Range check  -5C – 50C | Maximum temperature measured in today | 29.1 |
| Temp\_Avg | Float | 3 | Range check  -15C – 40C | Average temperature measured today | 21.5 |
| Eggs | integer | 2 | Range check  0-50 eggs | Input from user, only positive, if no input = 0 | 23 |
| Luminosity | integer | 1 | Range check  0-100% | Luminosity | 32 |

Daily table (one record every minute starting at 2 am)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field name | Type | Length /bytes | Validation | Description | Example |
| Index | Integer | 4 | Range check  0-1440 | Index | 0005 |
| Temperature outside | Float | 4 | Range check  -20C – 50C | Temperature outside the coop measured at that time | 13.6 |
| Temperature inside | Float | 4 | Range check  -20C – 50C | Temperature inside the coop measured at that time | 19.5 |
| Humidity | integer | 1 | Range check  0-100% | Humidity measured at that time | 60 |
| Luminosity | integer | 1 | Range check  0-100 | The amount of light hitting the coop | 55 |
| Water amount | Float | 4 | Range check  0-30l | How much water flowed in the last minute | 0.8 |
| Food Level | Integer | 1 | Range check  0-4 | Level of the food storage 0-5 | 4 |

Yearly Table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field Name | Type | Length  /bytes | Validation | Description | Example |
| Date | String | 8 | Length check | Date in the format ddmmyyyy | 13012016 |
| Food\_Used | Float | 4 | Presence check | How much food was used this day | 4.8 |
| Water\_used | Float | 4 | Presence check | How much water was used this day | 6.2 |
| Humidity | Integer | 1 | Range check  0-100% | Humidity measured at 2 pm | 44 |
| Temp\_Min | Float | 4 | Range check  -40C – 25C | Minimum temperature measured today | -13.8 |
| Temp\_Max | Float | 4 | Range check  -5C – 50C | Maximum temperature measured in today | 29.1 |
| Temp\_Avg | Float | 4 | Range check  -15C – 40C | Average temperature measured today | 21.5 |
| Eggs | integer | 2 | Range check  0-50 eggs | Input from user, only positive, if no input = 0 | 23 |

Thresholds Table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Fields | Type | Length /bytes | Validation | Description | Example |
| Update | String/Boolean | 3 | Format check  Must be “Yes” or “No” | If the value is “Yes” than the user has updated the thresholds but they haven’t been sent yet | Yes |
| Lower Threshold | Float | 4 | Must be smaller than the upper threshold | This is the lower threshold of the heating system | 8.6 |
| Upper Threshold | Float | 4 | Must be larger than the lower threshold | This is the upper threshold of the heating system | 13.0 |

## I2C transition table

|  |  |  |  |
| --- | --- | --- | --- |
| *Variable* | *1st byte* | *2nd byte* | *3rd byte (if available)* |
| Sent by Microcontroller |  |  |  |
| Temp. Out | Sign  0 is +, 1 is - | Value without a decimal point (12.6 = 12) | One decimal point after the decimal point |
| Temp. In | Sign  0 is +, 1 is - | Value without a decimal point (12.6 = 12) | One decimal point after the decimal point |
| Humidity | Value 0 - 100 | - | - |
| Luminosity | Value 0 - 100 | - | - |
| Water amount | Value without a decimal point (12.6 = 12) | One digit after the decimal point |  |
| Food Level | Value from 0 - 5 |  |  |
| Received By Microcontroller |  |  |  |
| Lower threshold | Value without a decimal point (12.6 = 12) | One digit after the decimal point |  |
| Upper threshold | Value without a decimal point (12.6 = 12) | One digit after the decimal point |  |

## Testing Plan:

| # | Objectives | Test Steps | Test Data | Expected Result | Actual Result | Pass/ Fail |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Check if the tables populate | Add randomly generated data to the tables and see if the data is displayed | Randomly distributed between reasonable thresholds  Temp : -20 to 40  Light and humidity: 0-100  Water flow : 0.0- 5.0 | Graphs show the data |  |  |
| 2 | Check if the tables have appropriate units of measurement and that they are shown where necessary | Open the website and browse through the tables | N/A | Tables have appropriate units of measurement and that they are shown where necessarily |  |  |
| 3 | Check if the website loads | -Start computer  -Connect to LAN  -Click on link | N/A | Webpage loads and all elements are present |  |  |
| 4 | Check if the values update the onscreen output | Slide the sliders  Increase/decrease the number of eggs | Mouse clicks/drags | All values update on input |  |  |
| 5 | Test validation of eggs input | input values that are not possible | Negative, decimal point and gigantic numbers Characters and symbols | Characters and symbols are not allowed  Numbers must be positive, whole and reasonable in size |  |  |
| 6 | Test validation of sliders input | Make both sliders identical  Make the first one greater than the second one and vice-a-versa | Mouse clicks/drags | Only when the first slider is lower than the second one, the submission is valid |  |  |
| 7 | Test if thresholds are updated | Change the value of the sliders | Mouse clicks/drags | Values are updated on refresh |  |  |
| 8 | Test if egg-count is updated | Change the number of eggs | Mouse clicks  Numerical input | Egg-count is updated |  |  |
| 9 | Test if the thresholds are sent to the microcontroller | -Open a python console  -Update the thresholds  -Wait the confirmation message | Mouse clicks/drags | The console displays: "The thresholds were set." |  |  |
| 10 | Test if the heater starts before the first set of thresholds are sent | Turn on the server box  Check on the heater | N/A | The heater should not start until the thresholds are sent |  |  |
| 11 | Test if live measurements update | Wait a few minutes and refresh the page | N/A | Live measurements change |  |  |
| 12 | Test if daily table refreshes at the end of the day and a new record is made in the monthly table | Check the next day/ change the time of the server | N/A | Daily table should be cleared, except for the first row and a new record should be added to the monthly table |  |  |
| 13 | Test if the monthly table refreshes at the end of the month and a new record is created in this year’s table | Change the servers date and time | N/A | Monthly table should be cleared, except for the first row and a new record should be added to this year’s table |  |  |
| 14 | Test if this year’s table refreshes at the end of the year and its contents are transferred to last year’s table | Change the servers date and time | N/A | this year’s table should refresh at the end of the year and its contents are transferred to last year’s table |  |  |
| 15 | Test if the server updates the date and time on power on | -Shut down the server  -Wait 5 min.  -Check the date and time | N/A | Date and time should be accurate within few seconds |  |  |
| 16 | Ability to share online | Check if the tables can be saved and if they are displayed correctly on social media platforms | N/A | Tables can be saved in standard formats like .png and .jpeg and are displayed properly on social media |  |  |
| 17 | Test automatic refill of water trays | Drain some water out of the trays and wait for them to be filled back up | N/A | As the water drops below a certain level, the water valve opens and fills the trays back up |  |  |

# Client Agreement

By signing this document, I confirm that I have read the Design Specifications and I understand the objectives, costs and limitations of the system. Furthermore, I agree to move to the Development stage of the project.

**Signature of client: Signature of provider:**

# Implementation:

All the code is split into three parts:

* Arduino Code (for monitoring and automation)
* Raspberry pi database handling code
* Server code

The Arduino code is only one file called “FarmProject.ino” and needs to be compiled using the Arduino IDE. The IDE can be downloaded from <http://arduino.cc> . The code can be viewed in Appendix 1.

The database is written in Python and is only one file called “farm\_project.py” which manages all of the tables (dailytable.csv, monthlytable.csv… etc.). It is placed in the root folder of the Raspberry pi and runs at start-up. The code can be viewed in Appendix 2.1.

The Server has three important files that take care of the User interface, Submission of inputs and an ajax.js file which is used to run the submission code in the background.

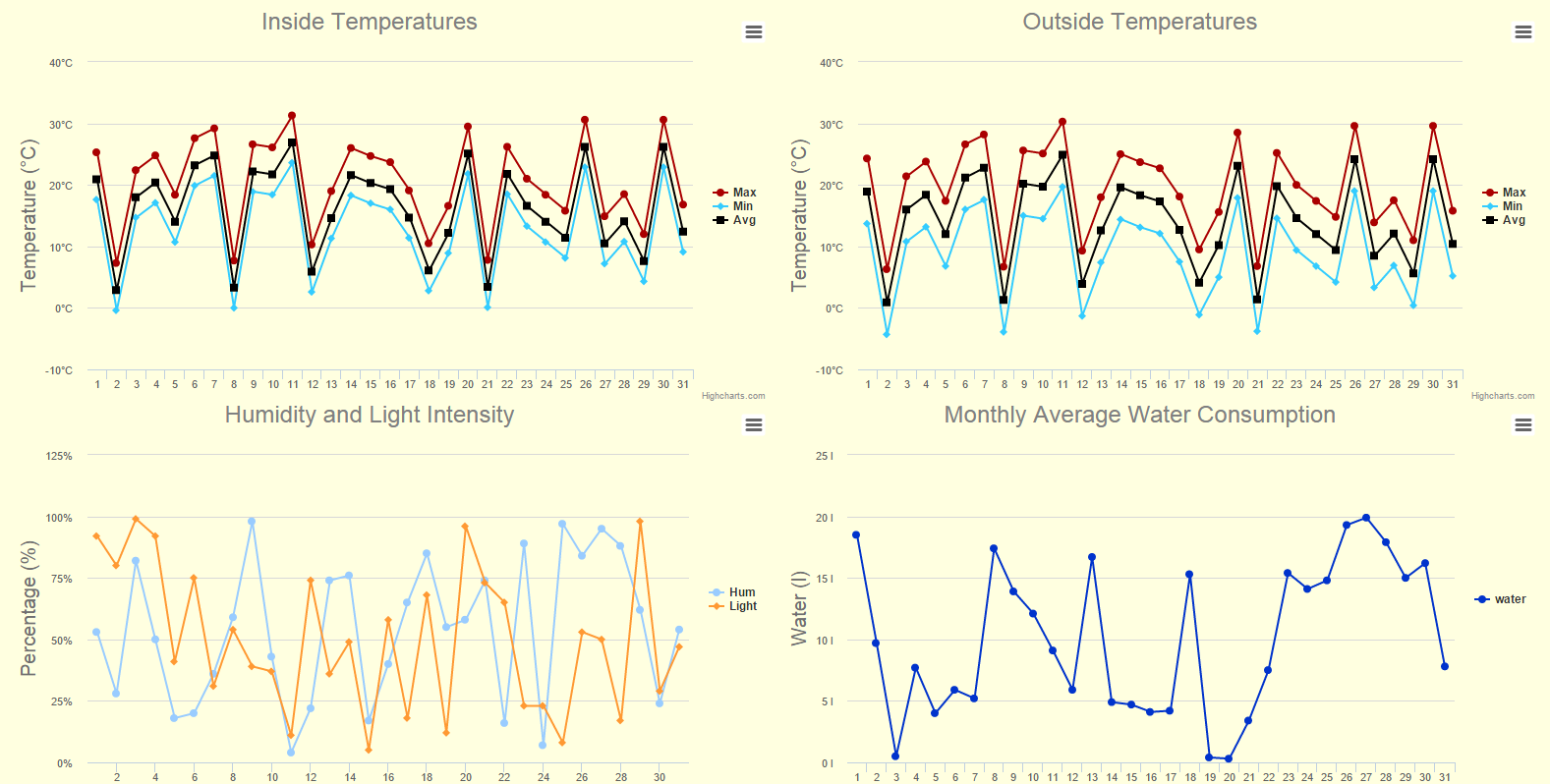
The website which handles the user interface is called “form.php” and can be viewed in Appendix 2.2. The input side is done in HTML5. For the file handling part of the code, PHP was mostly used as it is easily integrated in HTML. When the submission button is pressed, the code calls a function in the ajax.js file (see Appendix 2.4) called “update()” which sends the values from the form to “formhandler.php“. The code in this file can be viewed in Appendix 2.3. This is the file that takes care of the file handling and updates the database. All of these files are stored in the apache server folder at “/var/www” .

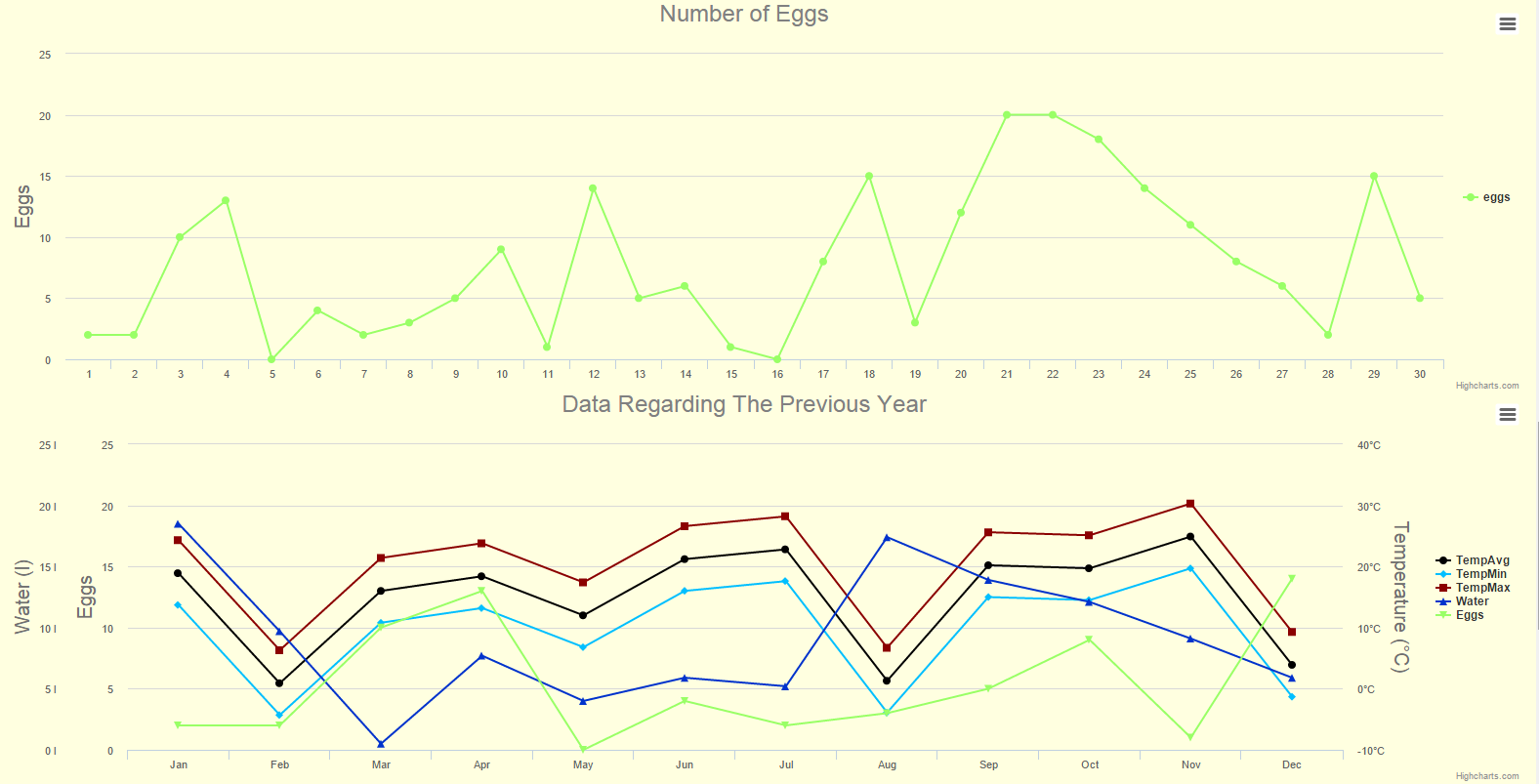
## Testing:

| # | Objectives | Test Steps | Test Data | Expected Result | Actual Result | Pass/ Fail |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Check if the tables populate | Add randomly generated data to the tables and see if the data is displayed | Randomly distributed between reasonable thresholds | Graphs show the data | All graphs are displayed properly | ✓ |
| 2 | Check if the tables have appropriate units of measurement and that they are shown where necessarily | Open the website and browse through the tables | N/A | Tables have appropriate units of measurement and that they are shown where necessarily | Appropriates units are used every where | ✓ |
| 3 | Check if the website loads | -Start computer  -Connect to LAN  -Click on link | N/A | Webpage loads and all elements are present | Webpage loads and all elements are present | ✓ |
| 4 | Check if the values update the onscreen output | Slide the sliders  Increase/decrease the number of eggs | Mouse clicks/drags | All values update on input | All values update on input | ✓ |
| 5 | Test validation of eggs input | input values that are not possible | Negative, decimal point and gigantic numbers Characters and symbols | Characters and symbols are not allowed  Numbers must be positive, whole and reasonable in size | Validation works | 🗶 |
| 6 | Test validation of sliders input | Make both sliders identical  Make the first one greater than the second one and vice-a-versa | Mouse clicks/drags | Only when the first slider is lower than the second one, the submission is valid | Only when the first slider is lower than the second one, the submission is valid | ✓ |
| 7 | Test if thresholds are updated | Change the value of the sliders | Mouse clicks/drags | Values are updated on refresh | Values are updated on refresh | ✓ |
| 8 | Test if egg-count is updated | Change the number of eggs | Mouse clicks  Numerical input | Egg-count is updated | Egg-count is updated | ✓ |
| 9 | Test if the thresholds are sent to the microcontroller | -Open a python console  -Update the thresholds  -Wait the confirmation message | Mouse clicks/drags | The console displays: "The thresholds were set." | The console displays: "The thresholds were set." | ✓ |
| 10 | Test if the heater starts before the first set of thresholds are sent | Turn on the server box  Check on the heater | N/A | The heater should not start until the thresholds are sent | The heater does not start until the raspberry pi boots up | ✓ |
| 11 | Test if live measurements update | Wait a few minutes and refresh the page | N/A | Live measurements change | All live measurements change | ✓ |
| 12 | Test if daily table refreshes at the end of the day and a new record is made in the monthly table | Check the next day/ change the time of the server | N/A | Daily table should be cleared, except for the first row and a new record should be added to the monthly table | Daily table is cleared and a new record was added to the monthly table | ✓ |
| 13 | Test if the monthly table refreshes at the end of the month and a new record is created in this year’s table | Change the servers date and time | N/A | Monthly table should be cleared, except for the first row and a new record should be added to this year’s table | Monthly table is cleared and a new record was added to this year’s table | ✓ |
| 14 | Test if this year’s table refreshes at the end of the year and its contents are transferred to last year’s table | Change the servers date and time | N/A | This year’s table should refresh at the end of the year and its contents are transferred to last year’s table | This year’s table refreshed at the end of the year and its contents were transferred to last year’s table | ✓ |
| 15 | Test if the server updates the date and time on power on | -Shut down the server  -Wait 5 min.  -Check the date and time | N/A | Date and time should be accurate within few seconds | Date and time is only updated when connected via Ethernet | 🗶 |
| 16 | Ability to share online | Check if the tables can be saved and if they are displayed correctly on social media platforms | Mouse clicks | Tables can be saved in standard formats like .png and .jpeg and are displayed properly on social media | Tables can be saved in standard formats like .png and .jpeg and are displayed properly on social media | ✓ |
| 17 | Test automatic refill of water trays | Drain some water out of the trays and wait for them to be filled back up | N/A | As the water drops below a certain level, the water valve opens and fills the trays back up | The trays were successfully refiled | ✓ |

### Test #1:

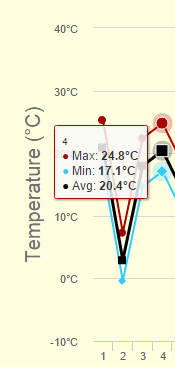
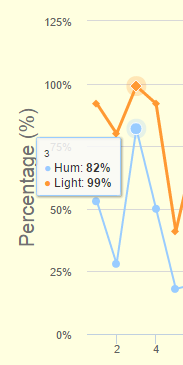
I have used randomly generated data to test the graphs.



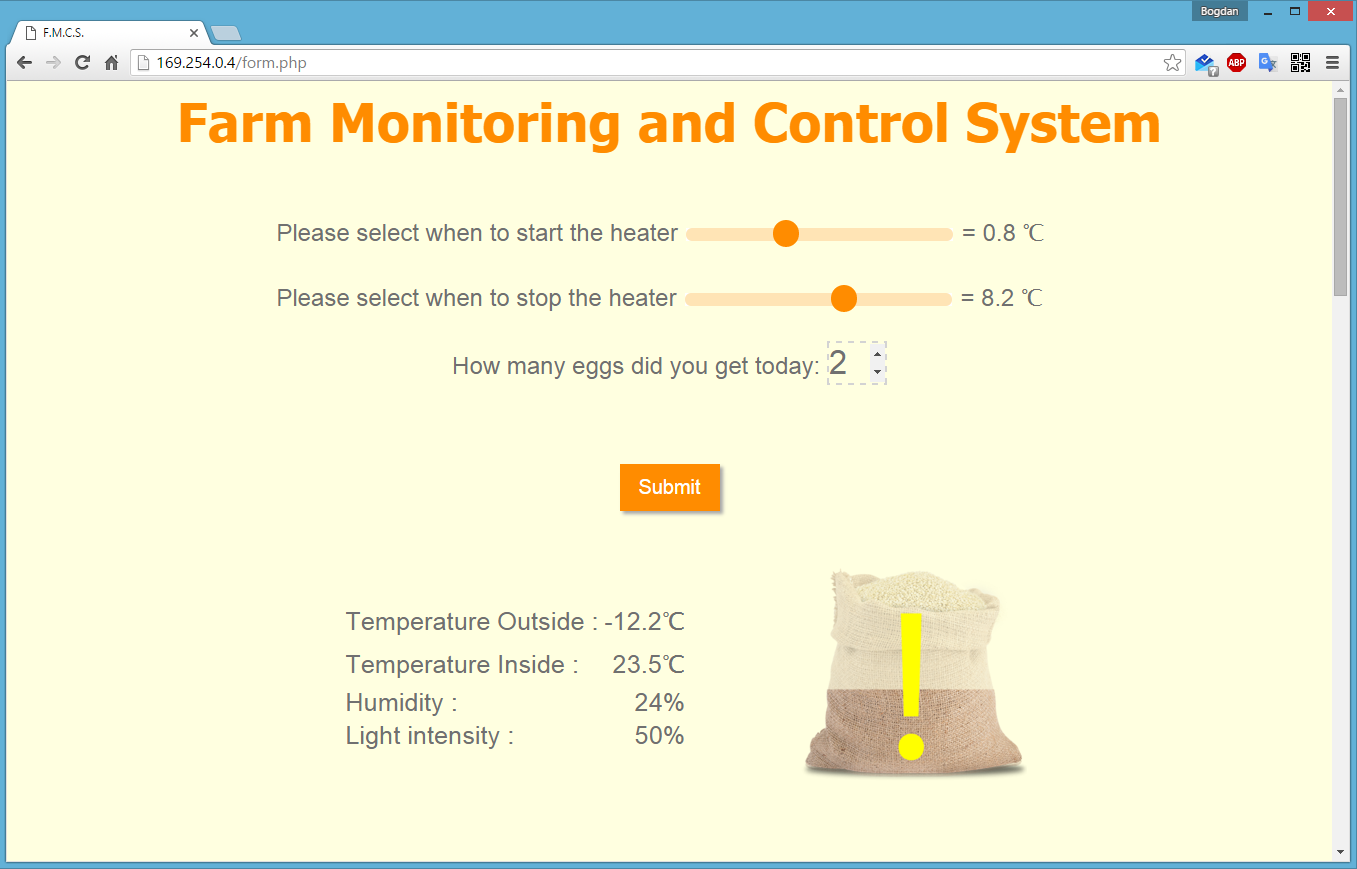


### Test #2:

As you can see, the tooltips show the all values for the day that the mouse hovers over. Also, all the suffixes are correct.



### Test #3:



**This IP address was only used for testing. To see the final one, see Installation/Network.**

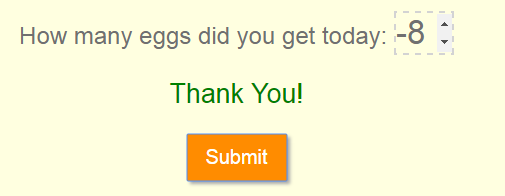
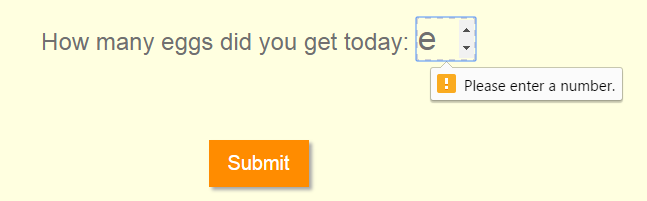
### Test #4:

This is how the sliders look when they are adjusted. As you can see, the temperature shown on the left changes.

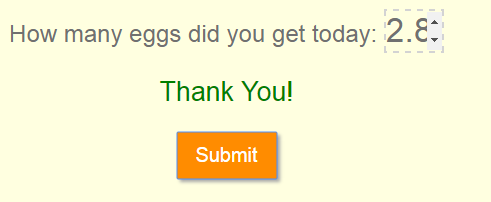
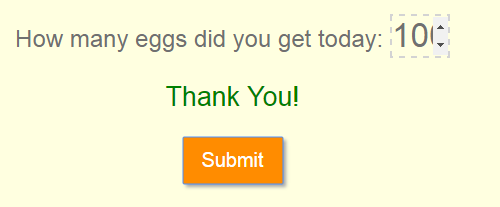


### Test #5:

No characters or symbols allowed:  **PASS** No negative numbers: **FAIL**

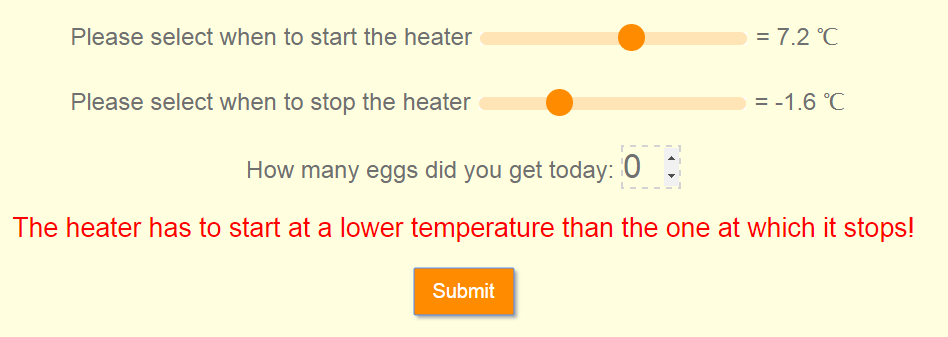
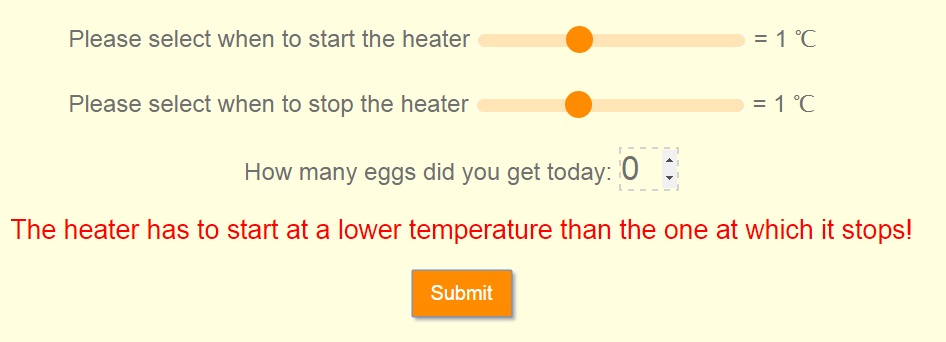


No more than 30 eggs in one day:  **FAIL** Only integers are allowed: **FAIL**



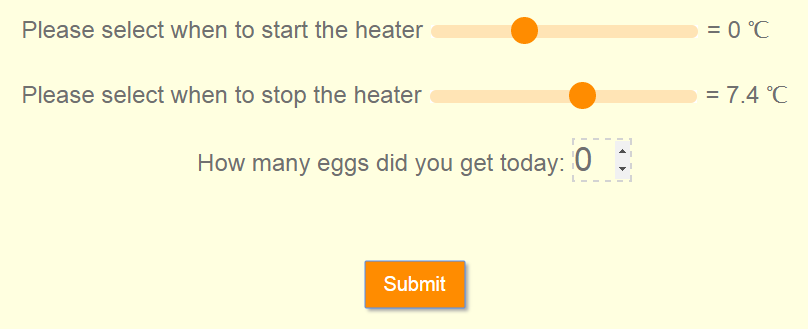
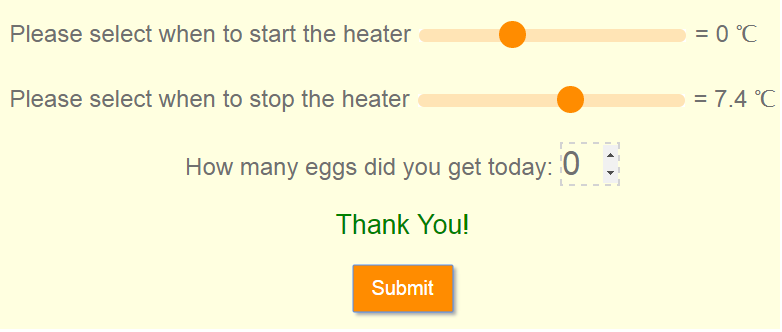
### Test #6:

Values cannot be equal: **PASS**  Upper threshold is lower than the lower threshold: **PASS**

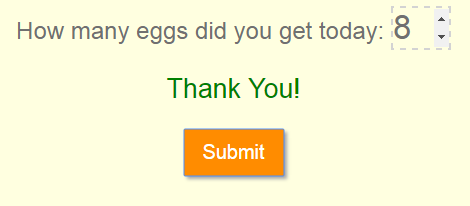
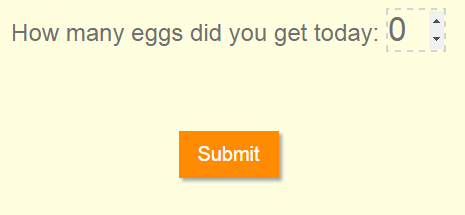


### Test #7:

After the thresholds were set After the page refreshes



### Test #8:



### Test #9:



### Test #10:

The test was done with the following method:

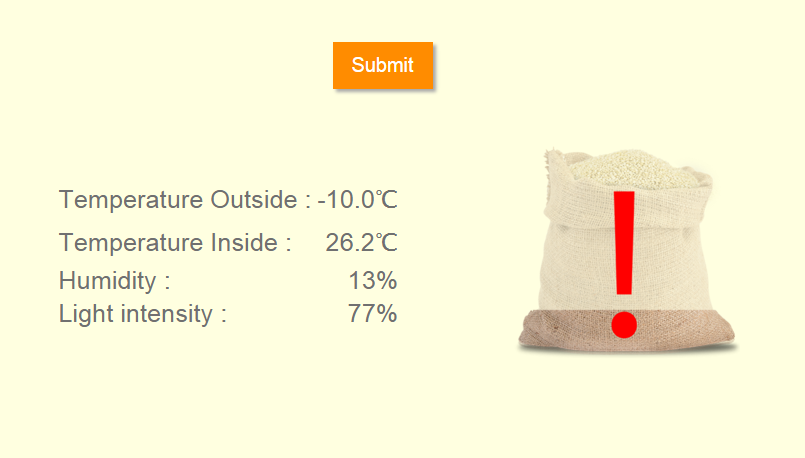
* The lower threshold was set well above the temperature inside the coop
* Power was cut from the server and the microcontroller
* Power was restored and the temperature of the heater was monitored
* The thresholds were set back to normal

The results were the following:

* The temperature of the heater did not rise after the power was restored
* After the thresholds were set back to normal, the heater started and stopped at the upper threshold

### Test #11:

The following screenshots were taken a few hours apart. We can see that the live measurements change over time.



### Test #12-14:

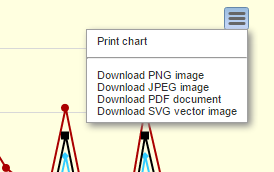
For these tests, the same procedure was applied to all the different cases. In order to test if the tables update, the date and/or time were changed in the Raspberry pi’s settings and the server was rebooted. The system has passed all 3 tests.

### Test #15:

The server only updates the time and date automatically when connected via Ethernet. A Wi-fi connection therefore is not adequeate for this system to work properly.

### Test #16:

The button in the following screenshot is available for all graph shown on the website.



# Implementation Method:

As this system has many complex elements which work over a long period of time, the best method of implementation would be a parallel implementation for the first 6 months. Any bugs that have passed through the testing phase should be discovered and after they are fixed and the client is satisfied with the new system, the Farm Monitoring and Control System will take over the notebook and visual checks based system that is currently in place.

The new system will have to be installed and tested again. The client will be given a short training on the basics of using the system along with the full technical documentation, installation manual and operation guide.

Implementation plan:

1. Hardware installation
2. Software installation
3. Initial testing
4. Client training
5. Client acceptance
6. Parallel running for 6 months
7. Phasing out the old system

Training plan:

1. Brief description of the system
2. Explanation on the main features of the system
3. Hands-on training with the system
4. Q&A

# Client Agreement

By signing this document I agree to the method of implementation chosen by Bogdan Ionescu and outlined in the Implementation Method section of the implementation stage. Furthermore, I acknowledge the cost and installation fee for the system. Furthermore, I confirm my participation in the testing and evaluation stage of the new system.

**Signature of client: Signature of provider:**

# Evaluation:

The system passed 16 out of the 18 tests. Some adjustments had to be done to solve the 2 problems that caused the tests to fail.

For test #5 extra validation rules have been added to the code to forbid the input of negative and non-integer numbers.

Test #15 failed because the system wouldn’t update the time after a reboot if it is connected to the LAN via Wi-Fi. The solution was to connect the Raspberry pi via an Ethernet cable. This also makes setting or changing the static IP much easier.

Next, we have to see if the initial objectives were met:

**Food level monitoring**

The system provides a user friendly way of monitoring the food level due to the use of pictograms instead of raw data. The client was very pleased with this solution.

**Egg-count tables/graphs**

All of the data regarding the production of eggs by the farm is beautifully displayed in line graphs. The number of eggs each month can be compared with other factors like temperature and humidity.

**Control of temperature**

The temperature can be monitored and controlled via the web interface. This in turn controls a heater inside the coop. This is a very simple and effective way of controlling the temperature inside the coop.

**Water consumption monitoring**

The flow meter measures the water consumption every minute and then a total for each day is displayed in the line graphs.

**Weather monitoring (humidity and light intensity)**

The humidity and luminosity sensors work as expected and give useful data about the weather on each specific day as well as live measurements.

**Visual representation of the values being monitored**

The food level is the only measurement that is displayed with pictograms. This is due to the fact that all of the other measurements should be as accurate as possible, therefore an illustrated output would not be adequate.

**User friendly interface**

The interface contains large numbers and buttons which makes it very user friendly for elderly people. Furthermore, all error messages are written in simple terms which makes troubleshooting easier.

**Budget under 500 euros**

The final cost for the whole system plus the installation costs comes around 350 euros, which is well below the initial budget.

**Automatic refill of water trays**

One large water tray had an automatic refilling system installed. This proved to work very well for the coop and was also very efficient in water consumption. The benefit of not having to manually check and refill the water trays was very much appreciated by the client.

**Temperature monitoring**

The temperature is monitored inside and outside of the coop by to sensors. This is data helps the client see if the coop needs more insulation or less depending on the season.

**Provide daily and monthly illustrations of the data as well as a comparison between this year and the previous one**

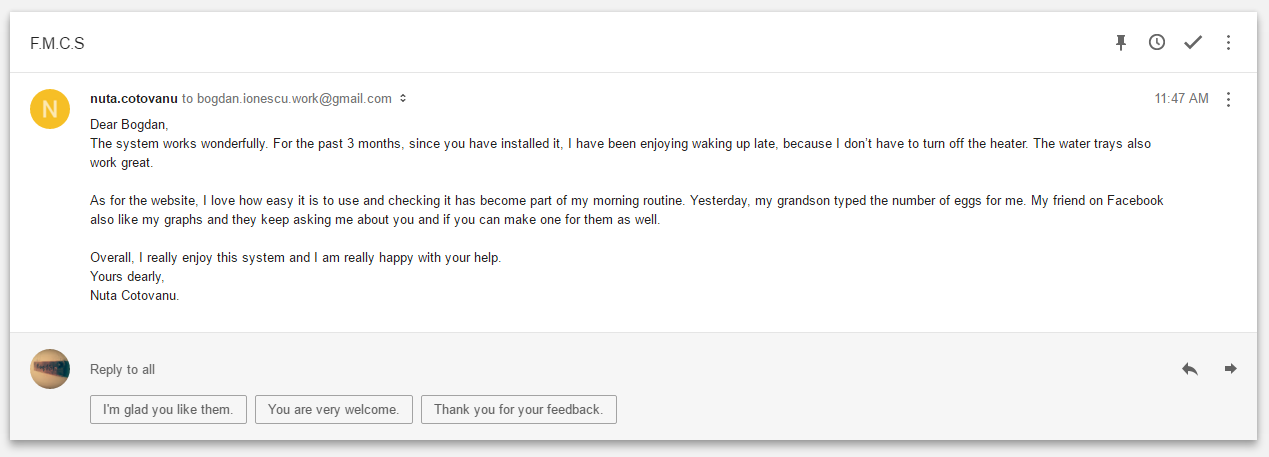
All of this is covered by tables at the very bottom of the website. The graph is plotted with monthly averages, minimums and maximus. All of the measurements are in one table to make it very easy to see trend between egg production and other external factors.

**Provide the ability to share statistics online**

All graphs can be saved in an image format and shared with social media.

Taking in consideration all of the above mentioned points, I would conclude that the system was successfully implemented and meets all the required specifications.

# Client’s response to the new system:



After the system was installed in parallel with the old one, a training session followed, after which the client was left for 2 weeks to get used to the new system. After the 2 weeks, I have met in person with the client and asked if she found any bugs or if she has any problems with it. Since then, I have only received this email.

The client seems satisfied with my solution. She really appreciates the fact that the system is autonomous so she doesn’t have to wake up early to check on the birds. Furthermore, she thinks it is very user friendly and she enjoys showing her friends her new system.

Overall, I think the system was a success and the client got exactly what she expected.

# Appendix 1 – Arduino Code:

/\*

FarmProject.ino

 \*/

/\* list of pins used

Digital

2 = flow meter

3 = DHT22 sensor

4 = water level light beam break sensor with a 10k pull-up resistor

5 = water valve

6 = heater relay

Analog

0 = light sensor

1 = outside temperature sensor

2 = food pressure sensor

4 = I2C data pin

5 = I2C clock pin

\*/

#include <Wire.h> //include the I2C library

#include <DHT.h> //include the DHT22 temperature and humidity sensor

DHT dht(3, DHT22); //declare a DHT22 sensor on pin 3 with the name dht

**int** address = 0x04 ; /\*this is the address that the Arduino has on the i2c bus ( addresses from 0x00 to 0x02 are not available)\*/

**int** LDRPin = 0; /\*the light sensor pin is connected to the analog pin 0 with a 10k pull-up resistor \*/

**int** TempOutPin = 1; /\*the outside temperature sensor is connected to the analog pin 1\*/

**int** FlowPin = 2; //the flow sensor is connected to the digital pin 2

**int** FoodSensorPin = 2; /\*the food pressure sensor is connected to the analog pin 2 with a 10k pull-up resistor \*/

**int** WaterLevelPin = 4; /\*the water level light gate is connected to the digital pin 4\*/

**int** WaterValve = 5; /\*the water valve is connected to the digital pin 5

**int** HeaterRelay = 6; /\*the relay for the heater is connected to digital pin 6\*/

**float** LowerThres=999; /\* create a floating variable for the lower threshold and use 999 as a temporary variable\*/

**float** UpperThres=999; /\* create a floating variable for the upper threshold and use 999 as a temporary variable\*/

byte thresh[4]; // create a byte array for the received thresholds

**float** TempIn ; //create a floating variable for the inside temperature

**float** Humidity ; //create a floating variable for the humidity

byte DATA[10]; //create a byte array for the data that has to be sent

/\* The hall-effect flow sensor outputs approximately 4.5 pulses per second per liters/minute of flow.\*/

**float** calibrationFactor = 4.5;

**volatile** byte pulseCount = 0; /\*volatile variable to store the pulses from the flow sensor \*/

**float** totalLitres = 0; /\*amount of water that has passes in the last minute\*/

**void** setup()

{

Wire.begin(address); /\*join the I2C bus as a slave with address 2, the master will have the address 1 \*/

dht.begin(); /\*start the communication with the DHT22 temperature and humidity sensor\*/

pinMode(FlowPin, INPUT\_PULLUP); //set the flow sensor as an input

pinMode(LDRPin, INPUT); //set the LDR as an input

pinMode(FoodSensorPin,INPUT); //set the food sensor as an input

pinMode(WaterLevelPin,INPUT); /\*set the water level sensor as an input\*/

pinMode(WaterValve, OUTPUT); /\*set the water valve as an output\*/

digitalWrite(WaterValve,LOW); //turn the valve off

pinMode(HeaterRelay,OUTPUT); //set the heater relay as an output

digitalWrite(HeaterRelay,LOW); //turn the relay off

Wire.onRequest(SendData); /\*when the master asks for data, send data\*/

Wire.onReceive(setThres); /\*when you receive data from the master, set them as the thresholds\*/

attachInterrupt(0, pulseCounter, FALLING); /\*interrupt set to trigger pulseCounter when signal drops from a high to a low state\*/

}

**void** loop() //this code is going to run in a loop forever

{

Water\_Level(); //call the water level subroutine

Heater(); //call the heater subroutine

}

**void** Water\_Flow()

{

totalLitres = ((pulseCount/60)/calibrationFactor); /\*calculate the amount of water in liters that has passed through the sensor in the last 60 seconds \*/

pulseCount = 0; //restart the counter

}

**void** pulseCounter()

{

//Increment the pulse counter

pulseCount++;

}

**void** Water\_Level()

{

**if** (digitalRead(WaterLevelPin) == HIGH){ /\*if the sensor receives light ( the water level is low)\*/

digitalWrite(WaterValve, HIGH); //turn on the valve

}

**else**

{

digitalWrite(WaterValve,LOW); //turn of the valve

}

}

byte Food\_Level()

{

**int** reading = analogRead(FoodSensorPin); /\* create an integer value to store the value of the food sensor \*/

**int** Level = map(reading, 720, 1023, 0, 5); /\* map the weight of the grains to a number from 0 to 5 indicating how much is left ( empty, almost empty, half full, almost full, full )\*/

**return** byte(Level);

}

**float** TempOut() /\*this function reads the outside temperature an returns the value in Celsius \*/

{

**int** reading = analogRead(TempOutPin); /\*create a variable to store the temperature reading \*/

// converting that reading to voltage

**float** voltage = reading \* 5.0;

**float** temperatureC = (voltage - 0.5) \* 100 ; /\*converting from 10 mV per degree with 500 mV offset to degrees ((voltage - 500mV) times 100)\*/

**return** temperatureC ;

}

**void** CheckTempHumIn() /\*this subroutine updates the inside temperature and humidity variables\*/

{

TempIn = dht.readTemperature();

Humidity = dht.readHumidity();

}

byte Light() /\*this function reads the light sensor and returns the value as a percentage\*/

{

**int** LightPercent = map(analogRead(LDRPin), 10, 1000, 0, 100); //map the analog value to a percentage

**return** byte(LightPercent);

}

**void** Heater() //this subroutine controls the heater

{

**if** (TempIn < LowerThres && LowerThres != 999) /\*999 is used so that the heater doesn't run when the thresholds are not set and the thresholds cannot take this value\*/

{

digitalWrite(HeaterRelay,HIGH);

}

**else** **if** (TempIn > UpperThres && UpperThres !=999) /\*999 is used so that the heater doesn't run when the thresholds are not set and the thresholds cannot take this value\*/

{

digitalWrite(HeaterRelay,LOW);

}

}

**void** ConvertToBytes() /\*this subroutine takes all the measurements and puts them in the array to be sent \*/

{

//the first 3 bytes are for the outside temperature

**int** TempHolder = TempOut(); /\*use a variable to hold the temperature reading \*/

// send the sign of the variable

**if** (TempHolder < 0)

{

DATA[0] = 1;

}

**else**

{

DATA[0] = 0;

}

DATA[1] = byte(**int**(**sqrt**(sq(TempHolder)))); /\*make the second byte of the array the absolute integer value of the variable\*/

DATA[2] = byte(**int**((TempHolder - DATA[1])\*10)); /\*take the decimal value out and then make it an integer \*/

//the following 3 bytes are for the inside temperature

CheckTempHumIn(); // call the routine

//send the sign of the variable

**if** (TempIn < 0)

{

DATA[3] = 1;

}

**else**

{

DATA[3] = 0;

}

DATA[4] = byte(**int**(**sqrt**(sq(**int**(TempIn))))); /\*make the second byte of the array the absolute integer value of the variable (two int() converting functions are used to avoid unexpected result)\*/

DATA[5] = byte(**int**((TempIn - DATA[4])\*10)); // take the decimal value out and then make it an integer

//the following byte is for the humidity

DATA[6] = byte(**int**(Humidity));

//the following byte is for the luminosity

DATA[7] = Light();

//the following 2 bytes are used for the amount of water used

Water\_Flow();

DATA[8] = byte(**int**(totalLitres));

DATA[9] = byte(**int**((totalLitres - DATA[8])\*10));

//the last byte is for the food level

DATA[10] = Food\_Level();

}

**void** setThres(**int** howMany) /\*this subroutine takes the data received and sets it as the thresholds \*/

{

**while**( Wire.available() ) /\*transfer all the data from the I2C buffer into thresh\*/

{

Wire.read(); /\*here we are ignoring the first 2 bytes of the transmission because they are used by the master to broadcast a command to the bus\*/

Wire.read();

**for**(**int** recieve = 0; recieve < 5; recieve++) /\*read the following 4 bytes and put them in an array\*/

{

thresh[recieve] = Wire.read();

}

}

LowerThres = thresh[0] + (**float**(thresh[1])/10);

UpperThres =thresh[2] + (**float**(thresh[3])/10);

}

**void** SendData() /\*this is a subroutine dedicated to sending the data \*/

{

noInterrupts(); /\*interrupts are disabled so that the transmission is uninterrupted \*/

ConvertToBytes();

Wire.write(DATA,**sizeof**(DATA)); /\*the wire.write() function needs to have the lenght of the array that is to be sent specified \*/

interrupts(); // interrupts are enabled again

}

# Appendix 2 Raspberry Pi Code:

## Appendix 2.1 farm\_project.py:

**import** smbus #import i2c library

**import** datetime #import system date and time library

**import** time # import time library (used for delays/sleep)

bus = smbus.SMBus(0) #initiate i2c bus

#change SMBus(0) to SMBus(1) if you are using a rev2.0 board

address = 0x04

# this is the address of the Arduino microcontroller on the i2c bus

DATA = [] #this array is used to store the incoming data blocks

#these are the global variables used for each reading/received data block

table\_index = 0

Temp\_out = 0.0

Temp\_in = 0.0

Humidity = 0

Luminosity = 0

Food\_level = 0

Water\_amount = 0.0

#this subroutine reads the thresholds from the file and

#sends them to the Arduino microcontroller

**def** Send\_Thresholds():

Thresholds\_DATA = [0,0,0,0]

Thres\_file = **open**('thresholds.csv','r')

Thres\_line = Thres\_file.readline()

Thres\_file.close()

Thresholds\_buffer = Thres\_line.strip('\n')

# remove carrige returns and commas

Thresholds = Thresholds\_buffer.split(",")

#here, the thresholds are converted to be sent

#\*\*see transfer protocol table

Thresholds\_DATA[0] = **int**(**float**(Thresholds[1]))

Thresholds\_DATA[1] = **int**((**float**(Thresholds[1])-**int**(**float**(Thresholds[1])))\*10.0)

Thresholds\_DATA[2] = **int**(**float**(Thresholds[2]))

Thresholds\_DATA[3] = **int**((**float**(Thresholds[2])-**int**(**float**(Thresholds[2])))\*10.0)

bus.write\_block\_data(address,0,Thresholds\_DATA)

#here, we send the data

**print** ("The thresholds were set.")

#here, we replace the first cell of the table with "No" to specify that #there have been no changes to the file since the last time we have checked

Thres\_file = **open**('thresholds.csv','w')

Thres\_file.write("No, " + Thresholds[1] + ", " + Thresholds[2])

Thres\_file.close()

#this subroutine interprets the incoming data blocks and transfers the #values to our global variables

**def** Interpret\_DATA():

#"global" needs to be used to specify that we are using the global #variables and not creating new local ones

**global** table\_index

**global** Temp\_out

**global** Temp\_in

**global** Humidity

**global** Luminosity

**global** Food\_level

**global** Water\_amount

Temp\_out = DATA[1] + (DATA[2]/10.0)

**if** DATA[0] == 1:

Temp\_out = -Temp\_out

Temp\_in = DATA[4] + (DATA[5]/10.0)

**if** DATA[3] == 1:

Temp\_in = -Temp\_in

Humidity = DATA[6]

Luminosity = DATA[7]

Water\_amount = DATA[8] + (DATA[9]/10.0)

Food\_level = DATA[10]

#this subroutine appends a new record to the daily table

**def** Write\_record():

**global** table\_index

dailyTable = **open**('dailytable.csv', 'a')

#for the values to be written to the file, they must first be converted to #strings with the str() function

dailyTable.write(**str**(table\_index))

table\_index+=1 #increment the index

dailyTable.write(", ")

dailyTable.write(**str**(Temp\_out))

dailyTable.write(", ")

dailyTable.write(**str**(Temp\_in))

dailyTable.write(", ")

dailyTable.write(**str**(Humidity))

dailyTable.write(", ")

dailyTable.write(**str**(Luminosity))

dailyTable.write(", ")

dailyTable.write(**str**(Water\_amount))

dailyTable.write(", ")

dailyTable.write(**str**(Food\_level))

dailyTable.write('\n') #this line writes a carriage return to the file

dailyTable.close()

#this function averages out the day and returns a record

**def** Average\_Day() :

#here, we create local temporary variables to store our sums, minimums, #maximums and the counter

total\_TOut = 0

total\_TIn = 0

total\_humidity = 0

total\_luminosity = 0

water = 0

lowest\_TOut = 100.0

lowest\_TIn = 100.0

highest\_TOut = -100.0

highest\_TIn = -100.0

count = 0

dailyTable = **open**('dailytable.csv', 'r')

skip = dailyTable.readline()

#here, we use a dummy variable to skip the first line

#this loop does the calculation for every line in the file

**for** line **in** dailyTable :

thisline = line.split(',')

total\_TOut += thisline[1]

total\_TIn += thisline[2]

total\_humidity += thisline[3]

total\_luminosity += thisline[4]

water += thisline[5]

count += 1

**if** thisline[1] < lowest\_TOut :

lowest\_TOut = thisline[1]

**if** thisline[2] < lowest\_TIn :

lowest\_TIn = thisline[2]

**if** thisline[1] > highest\_TOut:

highest\_TOut = thisline[1]

**if** thisline[2] > highest\_TIn:

highest\_TIn = thisline[2]

dailyTable.close()

record = **str**(total\_TOut/count) + ", " + **str**( lowest\_TOut) + ", " + **str**( highest\_TOut) + ", " + **str**( total\_TIn/count) + ", " + **str**( lowest\_TIn) + ", " + **str**( highest\_TIn) + ", " + **str**( humidity/count) + ", " + **str**( luminosity/count) + ", " + **str**(water)

**return** record

#this function averages the month and returns a record

**def** Average\_Month() :

#here, we create local temporary variables to store our sums, minimums, #maximums and the counter

total\_humidity = 0

total\_luminosity = 0

total\_TOut = 0

total\_TIn = 0

lowest\_TOut = 100.0

lowest\_TIn = 100.0

highest\_TOut = -100.0

highest\_TIn = -100.0

count = 0

water = 0

montlyTable = **open**('monthlytable.csv', 'r')

skip = dailyTable.readline()

#here, we use a dummy variable to skip the first line

#this loop does the calculation for every line in the file

**for** line **in** monthlyTable :

thisline = line.split(',')

total\_TOut += thisline[1]

lowest\_TOut += thisline[2]

highest\_TOut += thisline[3]

total\_TIn += thisline[4]

lowest\_TIn += thisline[5]

highest\_TIn += thisline[6]

total\_humidity += thisline[7]

total\_luminosity += thisline[8]

water += thisline[9]

count += 1

montlyTable.close()

record = **str**(total\_TOut/count) + ", " + **str**(lowest\_TOut/count) + ", " + **str**(highest\_TOut/count) + ", " + **str**(total\_TIn/count) + ", " + **str**(lowest\_TIn/count) + ", " + **str**(highest\_TIn/count) + ", " + **str**(humidity/count) + ", " + **str**(luminosity/count) + ", " + **str**(water) + ", " + "0"

**return** record

Send\_Thresholds()

#here, we send the thresholds when the program is started

#this is the main function which runs in a loop forever

**while** True:

timer = datetime.datetime.now()

#here, we read the current time and date of the system

**if** timer.second<10 :

#the following lines run every minute and we use less than 10 seconds so it #doesn't run for the whole minute, but we still have a threshold if #previous tasks take longer than expected

DATA = bus.read\_i2c\_block\_data(address,0)

Interpret\_DATA()

Write\_record()

time.sleep(10)

#here, we skip 10 seconds

#so there isn't any chance to repeat the previous lines

#here, we check if the thresholds have been update

check\_file = **open**('thresholds.csv','r')

check\_line = check\_file.readline()

check = check\_line.split(",")[0]

check\_file.close()

**if** check == "Yes" :

Send\_Thresholds()

**if** timer.hour == 0 **and** timer.minute == 0 **and** timer.second <= 10 :

#the following lines run every day and writes a record in the monthly table

monthlyTable = **open**('monthlytable.csv','a')

mothlyTable.write(timer.day + Average\_Day())

mothlyTable.close()

#here, we are resetting the daily table

dailyTable = **open**('dailytable.csv', 'w')

dailyTable.write("Index, Temperature Outside, Temperature Inside, Humidity, Luminosity, Food Level, Water amount" + '\n')

dailyTable.close()

time.sleep(10)

#here, we skip

#10 seconds so there isn't any chance to repeat the previous lines

**if** timer.day == 1 **and** timer.hour == 0 **and** timer.minute == 1 **and** timer.second < 10 :

#the following lines run at the beginning of every month and writes a #record in the year table

thisYear = **open**('thisyear.csv','a')

thisYear.write( timer.month + Average\_Month())

thisYear.close()

**if** timer.month == 1 :

#at the beginning of a new year we transfer the contents on the previous #year in another table and we reset the year table

thisYear = **open**('thisyear.csv','r')

lastYear = **open**('lastyear.csv','w')

**for** line **in** thisYear :

lastYear.write(line)

lastYear.close()

thisYear.close()

thisYear = **open**('thisyear.csv','w')

thisYear.write("Index, Average Temperature Outside, Minimum Temperature Outside, Maximum Temperature Outside, Average Temperature Inside, Minimum Temperature Inside, Maximum Temperature Inside, Humidity, Luminosity, Water amount")

thisYear.close()

time.sleep(10)

#here, we skip 10 seconds

#so there isn't any chance to repeat the previous lines

## Appendix 2.2 form.php

<?php

//here, we are reading the thresholds from the file

$file = fopen('/home/pi/thresholds.csv','r');

$**line** = fgets($file);

fclose($file);

$low = 0.0;

$high = 0.0;

$eggs = 0;

//here, we are transfering the thresholds **to** variables

$values = explode(',', $line);

$low = **trim**($values[1]);

$high = **trim**($values[2]);

$fileeggs = file('/home/pi/monthlytable.csv');

/\*this only opens the file for reading and there is no need for a closing as it loads the file in RAM (only good for small files)\*/

$**time** = strtotime("now"); //reading system time and date

$day = **date**('j',$time);

/\*this loop finds today in the table and takes the number of eggs from that record \*/

foreach($fileeggs **as** $**line**){

$values = explode(',', $line);

**if** ($values[0] == $day){

$eggs = $values[10];

}

}

$eggs = **trim**($eggs); //getting rid of unnecessary spaces

/\*The following lines initiate a very efficient and versatile method **for** reading the last line/lines of a large file when using limited RAM

This code has been taken from [http://www.geekality.net/2011/05/28/php-tail-tackling-large-files\*/](http://www.geekality.net/2011/05/28/php-tail-tackling-large-files*/)

$lines=10;

$buffer=4096;

$live = fopen('/home/pi/dailytable.csv','rb');

$cursor = -1;

fseek($live, $cursor-1, SEEK\_END);

**if**(fread($live, 1)!= "\n") $lines -=1;

$output ='';

$chunk ='';

**while**(ftell($live) > 0 && $lines >=0)

{

$**seek** = min(ftell($live), $buffer);

fseek($live, -$**seek**, SEEK\_CUR);

$output = ($chunk = fread($live,$**seek**)).$output;

fseek($live, -mb\_strlen($chunk,'8bit'), SEEK\_CUR);

$lines -= substr\_count($chunk,"\n");

}

**while**($lines++ <0) //adjust this **to** output more/less lines/characters

{

$output = substr($output, strrpos($output,"\n"));

}

fclose($live);

//this is where the borrowed code ends

//here, we are splitting the output into an **array**

$values = explode(",", $output);

$values = array\_map('trim',$values); //this gets rid of unnecessary spaces

/\*here we are transferring the values from the last **line** **to** variables in order **to** display them\*/

$tempOutnow = $values[1];

$tempInnow = $values[2];

$humiditynow = $values[3];

$luminositynow = $values[4];

$foodnow = $values[6];

?>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8">

<!-- The following Java Scripts are necessary for the graphs. Note that the jquery.min.js script will not be accessible while offline -->

<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery/1.8.2/jquery.min.js"></script>

<script src="../../js/highcharts.js"></script>

<script src="../../js/modules/exporting.js"></script>

<!-- this script takes care of the form submission via GET, so we don’t have **to** leave the page -->

<script src="ajax.js"></script>

<!--here starts the stylesheet-->

<style>

body {background-color:lightyellow; font-size:20pt }

h1 {font-family:Tahoma;color:darkorange; text-align:center;margin:0 auto;}

p {font-family:sans-serif ;font-size:18pt; color:#707070; text-align:center}

#Eggs {width:45pt; font-size:25pt; color:#707070;}

.slide {font-family:sans-serif ;font-size:18pt; color:#707070; }

#slide {width:590pt; margin:auto}

.message {text-align: center; width:100%; font-size : 20pt; color:green;font-family:sans-serif ;}

.**error** {text-align: center; width:100%; font-size : 20pt; color:red;font-family:sans-serif ;}

#submit {height:35pt; width:100 ; font-size:15pt; color:white; background-color:darkorange; border:2pt; box-shadow:2pt 2pt 3pt darkgrey;}

table {font-family:sans-serif ;font-size:19pt; color:#707070;}

**input**[type=range] {

-webkit-appearance: none;

}

**input**[type=range]:focus {

outline: none;

}

**input**[type=range]::-webkit-slider-runnable-track {

/\* this creates a custom range slider track\*/

width: 100%;

height: 10pt;

border-radius:5pt;

background: moccasin;

border:none;

}

**input**[type=range]::-webkit-slider-thumb {

/\* this creates a custom range slider \*/

height: 20pt;

width: 20pt;

border-radius: 50%;

background: darkorange;

cursor: pointer;

-webkit-appearance: none;

margin-top: -8px;

}

**input**[type=number]{

background: lightyellow;

border:2pt dashed ;

border-color:lightgrey;

}

**input**[type=number]::-webkit-inner-spin-button,

**input**[type=number]::-webkit-outer-spin-button {

opacity: 1; /\* this makes the number **input** arrows permanently visible\*/

}

</style>

</head>

<title>F.M.C.S.</title>

<body>

<h1>

Farm Monitoring and Control **System**</h1>

<br>

<br>

<**form** oninput = "x.value=LowT.value;y.value=HighT.value"> <!--this is sends the values of the sliders **to** the outputs **to** display them-->

<div **class**="slide" id="slide">

Please **select** when **to** start the heater

<**input** type = "range" style="width:200pt;" id="LowT" **name** ="LowT" min = "-10.0" max = "20.0" step = "0.2" value= "<?php echo $low;?>"> =

<output **name** = "x" **for**= "LowT" > <?php echo $low;?></output> &#x2103 <!--this is a special character **for** degrees Celcius-->

<br>

<br>

Please **select** when **to** **stop** the heater

<**input** type = "range" style="width:200pt;" id ="HighT" **name** ="HighT" min = "-10.0" max = "20.0" step = "0.2" value= "<?php echo $high;?>"> =

<output **name** = "y" **for**= "HighT" ><?php echo $high;?></output> &#x2103

</div>

<p>

How many eggs did you **get** today: <**input** type= "number" id = "Eggs" **name**="Eggs" value="<?php echo htmlspecialchars($eggs); ?>" min = "0">

</p>

<div id= "message"><br></div>

<p>

<**input** type = "button" id="submit" value="Submit" onClick = "update()">

</p>

</**form**>

<div style="width:100%; height=150pt">

<div style="margin: auto; width:40%; float:right">

<br>

<?php

echo "<img src='/bags/bag$foodnow.png'style='height:170pt;width:170pt;'>";

?>

</div>

<div style="margin: auto; width:50%; height:150pt">

<br><br>

<!--this is the live measurements table-->

<table>

<tr>

<td>Temperature Outside :</td><td style="text-align:right"><?echo $tempOutnow;?>&#x2103</td>

</tr><tr>

<td>Temperature Inside :</td><td style="text-align:right"> <?echo $tempInnow;?>&#x2103</td>

</tr><tr>

<td>Humidity :</td><td style="text-align:right"> <?echo $humiditynow;?>%</td>

</tr><tr>

<td>Light intensity :</td><td style="text-align:right"> <?echo $luminositynow;?>%</td>

</tr>

</table>

</div>

</div>

<br><br><br><br><br>

<?php

//these arrays are used **for** the graphs

$tempOut = **array**();

$tempOutMin = **array**();

$tempOutMax = **array**();

$tempIn = **array**();

$tempInMin = **array**();

$tempInMax = **array**();

$humidity = **array**();

$luminosity = **array**();

$water = **array**();

$eggs = **array**();

$file =fopen('/home/pi/monthlytable.csv','r');

fgets($file);

$**line**= fgets($file);

//this loop populates the arrays with all the values from the tables

**while**($**line** !== false){

/\* not false comparison is used because at the end of the document fgets() will return a 0\*/

$values = explode(',', $line);

array\_push($tempOut,**trim**($values[1], " "));

array\_push($tempOutMin,**trim**($values[2], " "));

array\_push($tempOutMax,**trim**($values[3], " "));

array\_push($tempIn,**trim**($values[4], " "));

array\_push($tempInMin,**trim**($values[5], " "));

array\_push($tempInMax,**trim**($values[6], " "));

array\_push($humidity,**trim**($values[7], " "));

array\_push($luminosity,**trim**($values[8], " "));

array\_push($water,**trim**($values[9], " "));

array\_push($eggs,**trim**($values[10], " "));

$**line**= fgets($file);

}

fclose($file);

$tempOutlastyear = **array**();

$tempOutMinlastyear = **array**();

$tempOutMaxlastyear = **array**();

$tempInlastyear = **array**();

$tempInMinlastyear = **array**();

$tempInMaxlastyear = **array**();

$humiditylastyear = **array**();

$luminositylastyear = **array**();

$waterlastyear = **array**();

$eggslastyear = **array**();

$file =fopen('/home/pi/lastyear.csv','r');

fgets($file);

$**line**= fgets($file);

//this loop populates the arrays with all the values from the tables

**while**($**line** !== false){

/\* not false comparison is used because at the end of the document fgets() will return a 0\*/

$values = explode(',', $line);

array\_push($tempOutlastyear,**trim**($values[1], " "));

array\_push($tempOutMinlastyear,**trim**($values[2], " "));

array\_push($tempOutMaxlastyear,**trim**($values[3], " "));

array\_push($tempInlastyear,**trim**($values[4], " "));

array\_push($tempInMinlastyear,**trim**($values[5], " "));

array\_push($tempInMaxlastyear,**trim**($values[6], " "));

array\_push($humiditylastyear,**trim**($values[7], " "));

array\_push($luminositylastyear,**trim**($values[8], " "));

array\_push($waterlastyear,**trim**($values[9], " "));

array\_push($eggslastyear,**trim**($values[10], " "));

$**line**= fgets($file);

}

fclose($file);

$tempOutthisyear = **array**();

$tempOutMinthisyear = **array**();

$tempOutMaxthisyear = **array**();

$tempInthisyear = **array**();

$tempInMinthisyear = **array**();

$tempInMaxthisyear = **array**();

$humiditythisyear = **array**();

$luminositythisyear = **array**();

$waterthisyear = **array**();

$eggsthisyear = **array**();

$file =fopen('/home/pi/thisyear.csv','r');

fgets($file);

$**line**= fgets($file);

//this loop populates the arrays with all the values from the tables

**while**($**line** !== false){

/\* not false comparison is used beacouse at the end of the document fgets() will return a 0\*/

$values = explode(',', $line);

array\_push($tempOutthisyear,**trim**($values[1], " "));

array\_push($tempOutMinthisyear,**trim**($values[2], " "));

array\_push($tempOutMaxthisyear,**trim**($values[3], " "));

array\_push($tempInthisyear,**trim**($values[4], " "));

array\_push($tempInMinthisyear,**trim**($values[5], " "));

array\_push($tempInMaxthisyear,**trim**($values[6], " "));

array\_push($humiditythisyear,**trim**($values[7], " "));

array\_push($luminositythisyear,**trim**($values[8], " "));

array\_push($waterthisyear,**trim**($values[9], " "));

array\_push($eggsthisyear,**trim**($values[10], " "));

$**line**= fgets($file);

}

fclose($file);

?>

<!--this is the JavaScript containing the graphs-->

<script type="text/javascript">

Highcharts.setOptions({

colors: ['#AA0000','#33CCFF','#000000'], //this setts the default/global colors

chart:{style:{fontFamily:'sans-serif', color:'#808080', fontSize:'15pt'}},

title:{style:{fontFamily:'sans-serif', color:'#808080', fontSize:'18pt'}}

});

$(**function** () {

$('#InTempsGraph').highcharts({

chart:{type:'line', backgroundColor:'rgba(0,0,0,0)'}, /\*this make the background transparent \*/

title: {

text: 'Inside Temperatures',

x: -20 //center

},

plotOptions:{

**line**:{

pointStart:1 /\* this makes the graph start at the first fay of each month as the default starting value is 0 \*/

}

},

xAxis: {

categories:[]

},

yAxis: {

title: {

text: 'Temperature (°C)'

},

plotLines: [{

value: 0,

width: 1,

color: '#808080'

}],

labels:{

**format**:'{value}°C'

}

},

tooltip: {

valueSuffix: '°C',

shared:true

},

legend: {

layout: 'vertical',

align: 'right',

verticalAlign: 'middle',

borderWidth: 0

},

series: [{

**name**: 'Max',

data: [<?php foreach($tempInMax **as** $value){echo $value.",";};?>] //these loops send all the values of the php arrays **to** javascript

}, {

**name**: 'Min',

data: [<?php foreach($tempInMin **as** $value){echo $value.",";};?>]

}, {

**name**: 'Avg',

data: [<?php foreach($tempIn **as** $value){echo $value.",";};?>]

}]

});

$('#OutTempsGraph').highcharts({

chart:{type:'line', backgroundColor:'rgba(0,0,0,0)'},

title: {

text: 'Outside Temperatures',

x: -20 //center

},

plotOptions:{

**line**:{

pointStart:1

}

},

xAxis: {

categories:[]

},

yAxis: {

title: {

text: 'Temperature (°C)'

},

plotLines: [{

value: 0,

width: 1,

color: '#808080'

}],

labels:{

**format**:'{value}°C'

}

},

tooltip: {

valueSuffix: '°C',

shared:true

},

legend: {

layout: 'vertical',

align: 'right',

verticalAlign: 'middle',

borderWidth: 0

},

series: [{

**name**: 'Max',

data: [<?php foreach($tempOutMax **as** $value){echo $value.",";};?>]

}, {

**name**: 'Min',

data: [<?php foreach($tempOutMin **as** $value){echo $value.",";};?>]

}, {

**name**: 'Avg',

data: [<?php foreach($tempOut **as** $value){echo $value.",";};?>]

}]

});

$('#HumLightGraph').highcharts({

chart:{type:'line', backgroundColor:'rgba(0,0,0,0)'},

colors: ['#99CCFF','#FF9933'], //this sets custom colors for the graph

title: {

text: 'Humidity and Light Intensity',

x: -20 //center

},

plotOptions:{

**line**:{

pointStart:1

}

},

xAxis: {

categories: []

},

yAxis: {

title: {

text: 'Percentage (%)'

},

plotLines: [{

value: 0,

width: 1,

color: '#808080'

}],

labels:{

**format**:'{value}%'

}

},

tooltip: {

valueSuffix: '%',

shared:true

},

legend: {

layout: 'vertical',

align: 'right',

verticalAlign: 'middle',

borderWidth: 0

},

series: [{

**name**: 'Hum',

data: [<?php foreach($humidity **as** $value){echo $value.",";};?>]

}, {

**name**: 'Light',

data: [<?php foreach($luminosity **as** $value){echo $value.",";};?>]

}]

});

$('#WaterGraph').highcharts({

chart:{type:'line', backgroundColor:'rgba(0,0,0,0)'},

colors: ['#0033CC'],

title: {

text: 'Monthly Average Water Consumption',

x: -20 //center

},

plotOptions:{

**line**:{

pointStart:1

}

},

xAxis: {

categories: []

},

yAxis: {

title: {

text: 'Water (l)'

},

plotLines: [{

value: 0,

width: 1,

color: '#808080'

}],

labels:{

**format**:'{value} l'

}

},

tooltip: {

valueSuffix: ' l',

shared:true

},

legend: {

layout: 'vertical',

align: 'right',

verticalAlign: 'middle',

borderWidth: 0

},

series: [{

**name**: 'water',

data: [<?php foreach($water **as** $value){echo $value.",";};?>]

}]

});

$('#eggstable').highcharts({

chart:{type:'line', backgroundColor:'rgba(0,0,0,0)'},

colors: ['#99FF66'],

title: {

text: 'Number of Eggs',

x: -20 //center

},

plotOptions:{

**line**:{

pointStart:1

}

},

xAxis: {

categories:[]

},

yAxis: {

title: {

text: 'Eggs'

},

plotLines: [{

value: 0,

width: 1,

color: '#808080'

}]

},

tooltip: {

valueSuffix: ' eggs',

shared:true

},

legend: {

layout: 'vertical',

align: 'right',

verticalAlign: 'middle',

borderWidth: 0

},

series: [{

**name**: 'eggs',

data: [<?php foreach($eggs **as** $value){echo $value.",";};?>]

}]

});

$('#lastyear').highcharts({

chart:{type:'line', backgroundColor:'rgba(0,0,0,0)'},

colors: ['#000000','#00BFFF','#8B0000','#0033CC','#99FF66'], //this sets custom colors for the graph

title: {

text: 'Data Regarding The Previous Year',

x: -20 //center

},

plotOptions:{

**line**:{

pointStart:0

}

},

xAxis: {

categories: ['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec']

},

yAxis: [{

title: {

text: 'Temperature (°C)'

},

plotLines: [{

value: 0,

width: 1,

color: '#808080'

}],

opposite:true,

labels:{

**format**:'{value}°C'

}

},{

title: {

text: 'Eggs'

},

plotLines: [{

value: 0,

width: 1,

color: '#808080'

}],

labels:{

**format**:'{value}'

}

},{

title: {

text: 'Water (l)'

},

plotLines: [{

value: 0,

width: 1,

color: '#808080'

}],

labels:{

**format**:'{value} l'

}

}],

tooltip: {

shared:true

},

legend: {

layout: 'vertical',

align: 'right',

verticalAlign: 'middle',

borderWidth: 0

},

series: [{

**name**: 'TempAvg',

data: [<?php foreach($tempOutlastyear **as** $value){echo $value.",";};?>],

type:'line',

tooltip:{valueSuffix:'°C'}

},{

**name**: 'TempMin',

data: [<?php foreach($tempOutMinlastyear **as** $value){echo $value.",";};?>],

type:'line',

tooltip:{valueSuffix:'°C'}

},{

**name**: 'TempMax',

data: [<?php foreach($tempOutMaxlastyear **as** $value){echo $value.",";};?>],

type:'line',

tooltip:{valueSuffix:'°C'}

},{

**name**: 'Water',

data: [<?php foreach($waterlastyear **as** $value){echo $value.",";};?>],

type:'line',

yAxis:2,

tooltip:{valueSuffix:' l'}

},{

**name**: 'Eggs',

data: [<?php foreach($eggslastyear **as** $value){echo $value.",";};?>],

type:'line',

yAxis:1,

tooltip:{valueSuffix:' eggs'}

}]

});$('#thisyear').highcharts({

chart:{type:'line', backgroundColor:'rgba(0,0,0,0)'},

colors: ['#000000','#00BFFF','#8B0000','#0033CC','#99FF66'], //this sets custom colors for the graph

title: {

text: 'Data Regarding This Year',

x: -20 //center

},

plotOptions:{

**line**:{

pointStart:0

}

},

xAxis: {

categories: ['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec']

},

yAxis: [{

title: {

text: 'Temperature (°C)'

},

plotLines: [{

value: 0,

width: 1,

color: '#808080'

}],

opposite:true,

labels:{

**format**:'{value}°C'

}

},{

title: {

text: 'Eggs'

},

plotLines: [{

value: 0,

width: 1,

color: '#808080'

}],

labels:{

**format**:'{value}'

}

},{

title: {

text: 'Water (l)'

},

plotLines: [{

value: 0,

width: 1,

color: '#808080'

}],

labels:{

**format**:'{value} l'

}

}],

tooltip: {

shared:true

},

legend: {

layout: 'vertical',

align: 'right',

verticalAlign: 'middle',

borderWidth: 0

},

series: [{

**name**: 'TempAvg',

data: [<?php foreach($tempOutthisyear **as** $value){echo $value.",";};?>],

type:'line',

tooltip:{valueSuffix:'°C'}

},{

**name**: 'TempMin',

data: [<?php foreach($tempOutMinthisyear **as** $value){echo $value.",";};?>],

type:'line',

tooltip:{valueSuffix:'°C'}

},{

**name**: 'TempMax',

data: [<?php foreach($tempOutMaxthisyear **as** $value){echo $value.",";};?>],

type:'line',

tooltip:{valueSuffix:'°C'}

},{

**name**: 'Water',

data: [<?php foreach($waterthisyear **as** $value){echo $value.",";};?>],

type:'line',

yAxis:2,

tooltip:{valueSuffix:' l'}

},{

**name**: 'Eggs',

data: [<?php foreach($eggsthisyear **as** $value){echo $value.",";};?>],

type:'line',

yAxis:1,

tooltip:{valueSuffix:' eggs'}

}]

});

});

</script>

<!--Here is where the graphs are actually placed by the JavaScript using their ID's -->

<div>

<div id="InTempsGraph" style=" height: 300pt; float:left; width:50%"></div>

<div id="OutTempsGraph" style=" height: 300pt;float:right; width:50% "></div>

</div>

<div>

<div id="HumLightGraph" style=" height: 300pt; float:left; width:50%"></div>

<div id="WaterGraph" style=" height: 300pt;float:right; width:50% "></div>

</div>

<div>

<div id="eggstable" style=" height: 300pt; float:left; width: 100%"></div>

</div>

<div>

<div id="lastyear" style=" height: 300pt; float:left; width:100%"></div>

</div>

<div>

<div id="thisyear" style=" height: 300pt; float:left; width:100%"></div>

</div>

<div style=" height: 100pt; float:left; width:100%"></div>

</body>

</html>

## Appendix 2.3 formhandler.php:

<?php

//fist we take the values from the URL via GET

$Low = $\_GET['LowT'];

$High = $\_GET['HighT'];

$Eggs = $\_GET['Eggs'];

//This condition makes sure the thresholds are properly set

**if**($Low < $High){

//if the statement is met, the thresholds are written **to** the table

$file = fopen('/home/pi/thresholds.csv', 'w');

fwrite($file,'Yes, '.number\_format($Low, 1).', '.number\_format($High, 1));

fclose($file);

$fileR = file('/home/pi/monthlytable.csv');

**if**($fileW = fopen('/home/pi/monthlytable\_buffer.csv','w+'))

/\*here we create a buffer file to transfer the values of the monthly table and update the egg-count \*/

$**time** = strtotime("now"); //take the system's time

$day = **date**('j',$time); // extract the day from the date

//The following **for** **loop** replaces the default value of eggs (0) **with** the **input** from the user

foreach($fileR **as** $**line**){

$values = explode(',', $line);

**if** ($values[0] == $day){

$values[10] = $Eggs;

**for** ($i = 0; $i<10; $i+=1){

fwrite($fileW, $values[$i].",");

}

fwrite($fileW, $values[10]."\n");

}

**else** {

fwrite($fileW,$**line**);

}

}

fclose($fileW);

unlink('/home/pi/monthlytable.csv'); //now we delete the old file

rename('/home/pi/monthlytable\_buffer.csv','/home/pi/monthlytable.csv'); // rename the new one to the correct name

echo '<div class="message">Thank You!</div>';

//if all goes well, this is the confirmation message

}

**else**{echo '<div class="error">The heater has to start at a lower temperature than the one at which it stops!</div>';}

//if the condition above is not met, an error message is displayed

?>

## Appendix 2.4 ajax.js:

**function** update() {

// this function is called when the submit button is clicked

**var** u = "formhandler.php";

//this is our URL for the handler function

//taking the values from the form

**var** lowT = document.getElementById('LowT').value;

**var** highT = document.getElementById('HighT').value;

**var** eggs = document.getElementById('Eggs').value;

u = u + "?LowT=" + lowT + "&HighT=" + highT + "&Eggs=" + eggs; //here we are adding the values to the url

**var** xhttp = **new** XMLHttpRequest();

xhttp.onreadystatechange = **function**() {

**if** (xhttp.readyState == 4 && xhttp.status == 200) {

document.getElementById("message").innerHTML = xhttp.responseText; // send the message from the handler

**setTimeout**(**function**(){ document.getElementById("message").innerHTML = "<br>" }, 3000);

//this makes the text disappear after 3 seconds

}

}

xhttp.open("GET", u, true);// opening the URL with GET

xhttp.send();//sending the data

}

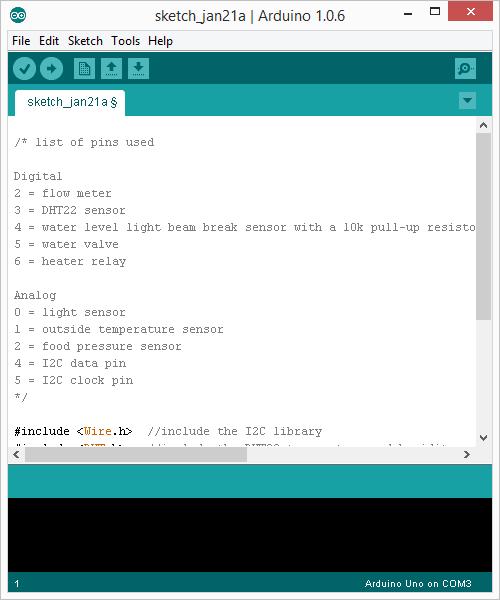
# Installation:

If the system needs to be reinstalled the following steps will create a fully working system.

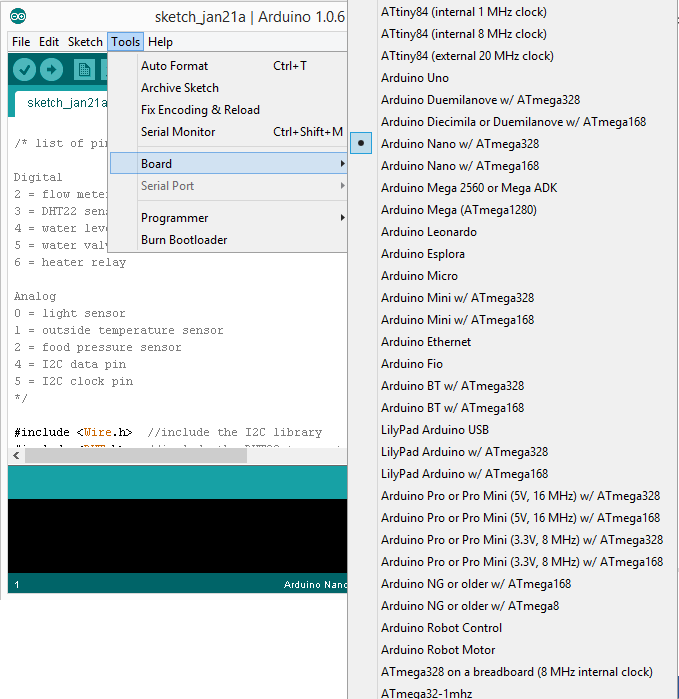
## Writing the firmware to the Arduino:

1. Connect the Arduino to a computer that has the Arduino IDE installed on it. It can be downloaded from <https://www.arduino.cc/en/Main/Software> .

2. Open the program and copy the code from Appendix 1 into the window.



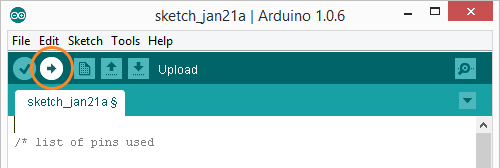
3. Click on Tools/Boards/ and select your board



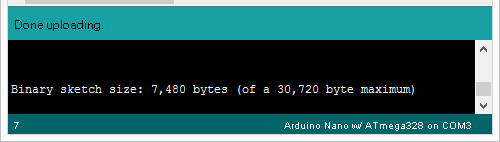
4. Next, in Tools/Serial Port, make sure there is only one serial port shown and that one is selected.

If you have more than one, unplug the Arduino and see which one disappears. That one is the one that you want to select. The ports should look like this: COM 5, COM 6 etc.

5. Click on Upload:



6. If everything goes well, a “Done Uploading” message will appear above the output console.



## Preparing the Raspberry Pi:

1. With the newest version of Raspbian running on the Raspberry Pi, open a new Terminal (or if you are in the command line interface) and write “sudo apt-get install update”. This will ensure that we will install the latest packages in the following steps.

2. To install the Apache server type “sudo apt-get install apache” and press Enter. Press Y and then press Enter again.

3. Close the Terminal, Open the File Explorer and go to /var/www/ and paste the files from Appendix 2.2, 2.3 and 2.4 with their respective names.

4. Download the Highcharts library from <http://www.highcharts.com/download> and extract it to /var/www/ .

5. Now go to the /home/pi/ and copy farm\_project.py from Appendix 2.1.

6. Next, we have to create the tables for the database. Create the following empty files: dailytable.csv, monthlytable.csv, thresholds.csv, thisyear.csv, lastyear.csv;

7. Open dailytable.csv and paste the following (it should be on one line only):

Index, "Temperature Outside", "Temperature Inside", Humidity, Luminosity, "Water amount", "Food Level"

And press Enter. (to create a new line )

8. Open monthlytable.csv and paste the following (it should be on one line only):

Index, Average Temperature Outside, Min Temperature Outside, Max Temperature Outside, Average Temperature Inside, Min Temperature Inside, Max Temperature Inside, Humidity, Luminosity, Water, Eggs

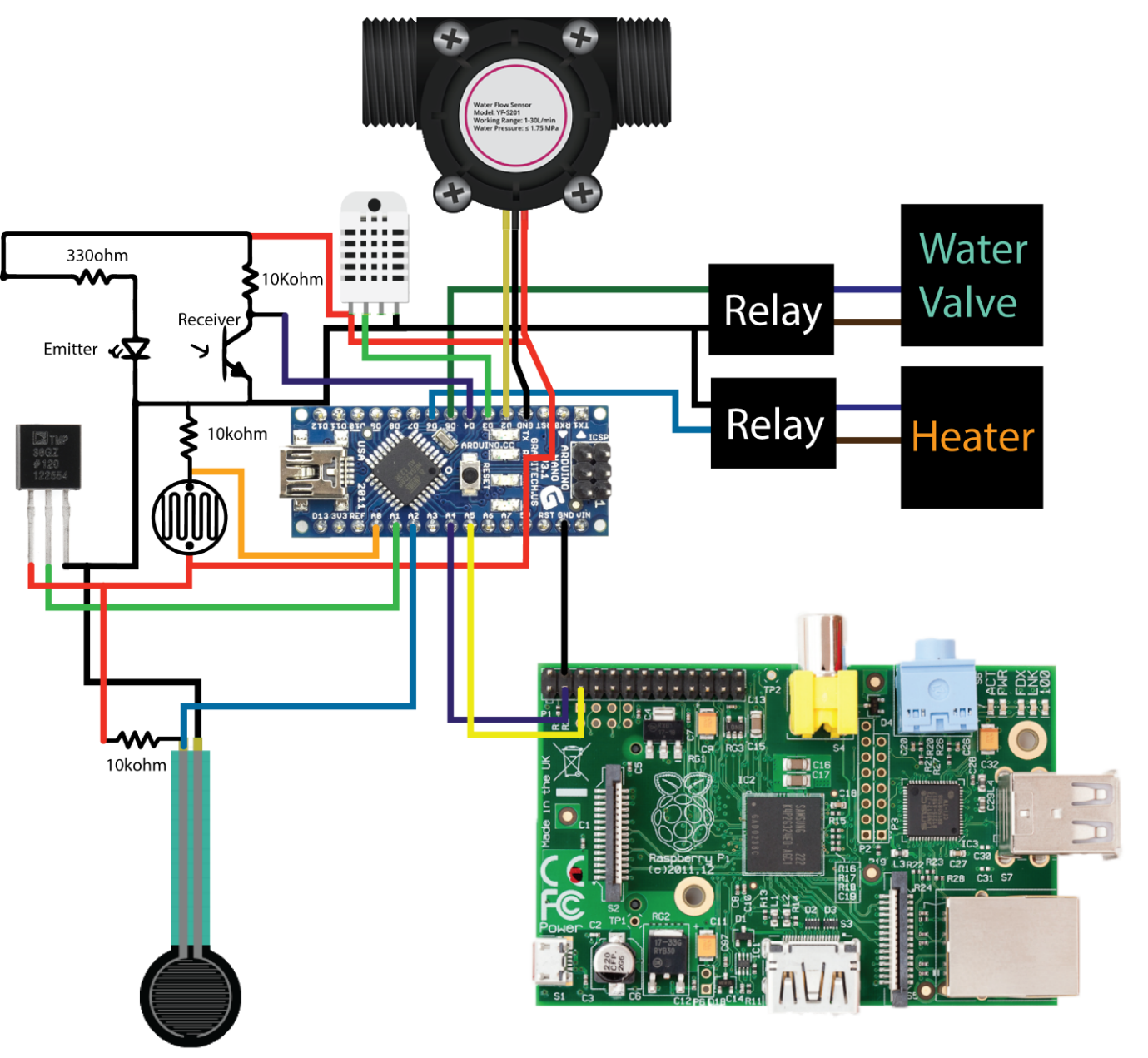
9. Lastly, open thresholds.csv and write the following:

Yes, 99.9, 99.9

You could replace “99.9” with some thresholds that seem reasonable. Anyway, you can adjust them later from the user interface

## Circuit:

This is the diagram of the complete circuit. The red lines are all connected to 5 volts and all the black wires (not the lines in the emitter-receiver circuit) are connected to ground.



## I2c configuration:

1. Open the Terminal and enter the following command:

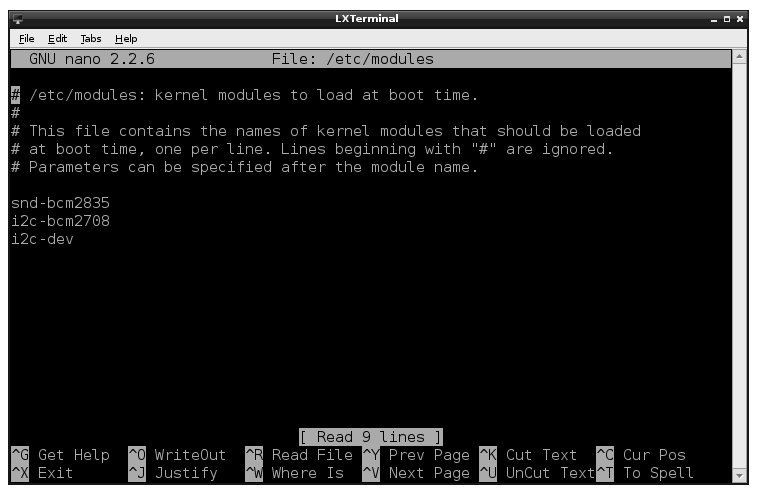
sudo nano /etc/modules

2. Add these two lines to the end of the file:

i2c-bcm2708

i2c-dev

The file should look something like this:



3. Press Ctrl+X Press Y and then press Enter to save and exit.

4. Your distribution may also have a file called /etc/modprobe.d/raspi-blacklist.conf . If you do not have this file then there is nothing to do, however, if you do have this file, you need to edit it and comment out the lines below by putting a # in front of them:

blacklist spi-bcm2708

blacklist i2c-bcm2708

This has to be done, just like the previous step, using the command “sudo nano” and then the path of the file.   
5. Press Ctrl+X Press Y and then Enter.

6. You will also need to update the /boot/config.txt file using sudo nano /boot/config.txt and then adding the following at the bottom of the file:

dtparam=i2c1=on

dtparam=i2c\_arm=on

7. Now just reboot the Raspberry pi (command: sudo reboot)

8. To test if the i2c connection is established, with the Arduino powered on, open the Terminal and enter the following command:

sudo i2cdetect -y 1

If you are using an older raspberry pi model B (256MB RAM) you need to change the “1” to a “0”.

You should see a device connected to address 00x4. Press Ctrl+C to exit.

\*\*Note make sure you have the appropriate i2c port (1 or 0) at the beginning of the farm\_project.py file.

## Network:

To connect the Raspberry Pi to the local area network we will have to set a fixed IP, but first ask your network administrator for the gateway address of your network.

1. Plug in the Ethernet cable into the Raspberry Pi.

2. Turn on the Raspberry pi.

3. Create a new file by typing

sudo nano SetStaticIP

4. Now paste the following lines:

#!bin/bash

sudo ifconfig eth0 192.168.x.99

Replace x with the number that is in the same spot on your gateway address

5. Save the file by pressing Ctrl+X, then Press Y and then Enter.

6. Now we have to make the file executable. Run the following command:

sudo chmod 777 SetStaticIP

7. Finally we need to make this run automatically at every boot. Run the following command:

sudo update-rc.d SetStaticIP defaults

To test if everything is working properly, reboot the Raspberry Pi, wait a few minutes and then on a computer connected to the same network, enter the address that you have set in step number 4.

# Operation guide:

## Starting and rebooting the system:

To turn on the system, just plug it in and wait for one minute.

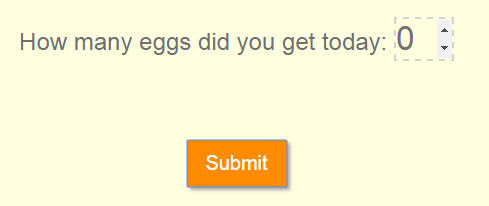
Check the lights on the Raspberry Pi! If they do not flicker, it probably means that the SD Card is not properly installed.

If everything goes well, you should be able to access the web page from a browser.

The system is made to run continuously, so there is no easily accessible reboot button. To reboot the system, you will have to connect to raspberry pi via SSH.

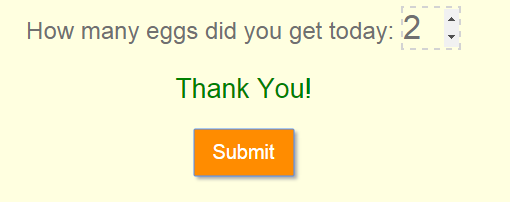
## Inputting the number of eggs:

To input the number of eggs, you can either use the arrows next to the input box or you can type it inside the box.



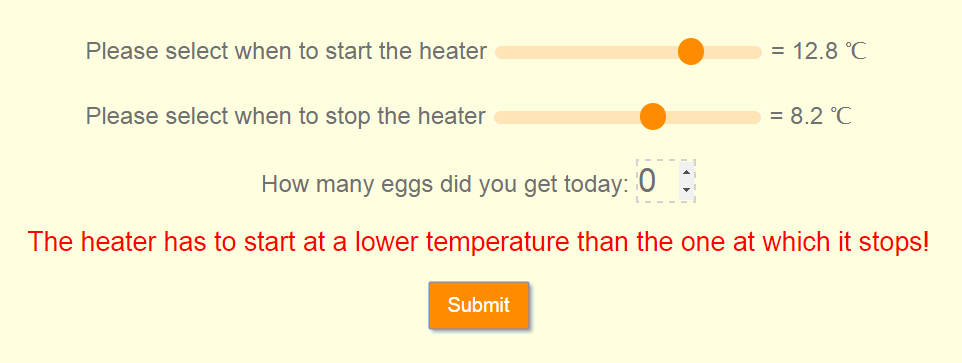
Now click on Submit.

If everything goes well, then you should get the following message:



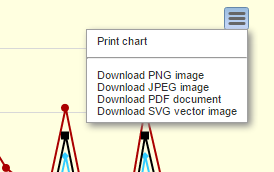
## Changing the thresholds for the heater:

Click on the circles on the sliders and drag them to the temperature that you desire. Make sure the lower threshold is not larger than the higher one. This will give you an error when you try submitting it.



## Saving a graph:

You can save any graph by clicking on the button in the top right corner of the table and chose your desired format (.jpeg is recommended)



## Glossary:

Apache Server – Software package used to create web servers

Arduino – The name of a microcontroller platform

Firmware – Code that tells the microcontroller what to do

Gateway address - A router interface connected to the local network that sends packets out of the local network

Highcharts – A library used to create charts

I2C – low level serial communications protocol for microcontrollers

IDE – An Integrated Development Environment is a piece of software that is used to create and upload code to a certain device

IP -A unique string of numbers separated by periods that identifies each computer using the Internet Protocol to communicate over a network

Library – A collection of code written by someone else that is used to facilitate development

Local area network – A computer network that spans a relatively small area

Raspbian – An operating system written for the Raspberry Pi

SD card – A Secure Digital Card is an ultra-small flash memory carddesigned to provide high-capacity memory in a small size

SSH – A Secure Socket shell is a safe method of remotely accessing a computer over the internet

Serial Port – A computer port used to identify external peripherals

Terminal – A command line interface for the operating system