Politehnica University Timișoara
Faculty of Automation and Computers
Computers and Information Technology

IMPLEMENTATION OF A MULTI SENSOR BASED SYSTEM FOR ENVIRONMENTAL CONDITIONS MONITORING IN A SMALL-SCALE GREENHOUSE

Candidate:
Nicu-Șerban POP

Supervisors:
Șl. dr. eng. **Raul Ciprian IONEL**

Timișoara
2017

DECLARAŢIE DE AUTENTICITATE A LUCRĂRII DE FINALIZARE A STUDIILOR

Subsemnatul _____,
legitimat cu _____seria _____nr. _____,
CNP _____autorul lucrării_____
_____
_____
elaborată în vederea susţinerii examenului de finalizare a studiilor de licență
_____
organizat de către Facultatea de Automatică și Calculatoare din cadrul Universităţii Politehnica Timişoara, sesiunea _____ a anului universitar_____, luând în considerare conţinutul art. 39 din RODPI – UPT, declar pe proprie răspundere, că această lucrare este rezultatul propriei activităţi intelectuale, nu conţine porţiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislaţiei române şi a convenţiilor internaţionale privind drepturile de autor.

Timişoara,
Data Semnătura

_____ _____
Declaraţia se completează „*de mână*" şi se inserează în lucrarea de finalizare a studiilor, la sfârşitul acesteia, ca parte integrantă.

Contents

List of Figures

List of Tables

List of Formulas

1. Introduction

The project, called *Farm Box*, is a project designed for plant lovers, growing enthusiasts and everyone who wants to automate the plant growing systems making use of light bulbs and water pumps installed inside together with a file backup system.

1.1. Problem Statement

In the present days, traditional ways of managing and growing plants are still vastly used, but they require human labors to periodically check the temperature, humidity and soil moisture manually, thus being time consuming and they require a lot of work. On the other hand, some plants require very strict conditions to grow properly and are very hard to achieve without automation. They may require a constant light level, constant temperature and a very strict watering schedule.

1.2. Solution

The proposed system manages all the parameters needed for the plant by itself: light level, soil humidity and others and provides real-time information to the user via a web page, the windows application and the 16x2 LCD mounted on the box with the help of an Arduino UNO and a Raspberry Pi. Light is provided by a light bulb and watering is provided by a water pump connected to a water container inside the box.

When booted, the system initializes its own growing parameters that provide light and water but can be switched to manual mode by using the desktop application provided.

1.3. Related Works

Nowadays, technology has reached a point where it's widely available and present in everyone's life. Software programs are easier to produce due to the vast majority of resources while hardware components are cheap and easily obtainable. Together with the IoT, more complicated and advanced systems can be built for observing, monitoring and controlling. While wired communications are often used, wireless communication brings a level of freedom to any project such as portability and ease-of-use [1].

In Codogno, Italy, Auroras is a technological start-up that focuses on environmental monitoring systems in order to provide reliable and efficient management control in growing plants. A simplified version of their implementations is illustrated in Figure 1. And we will use it as reference.

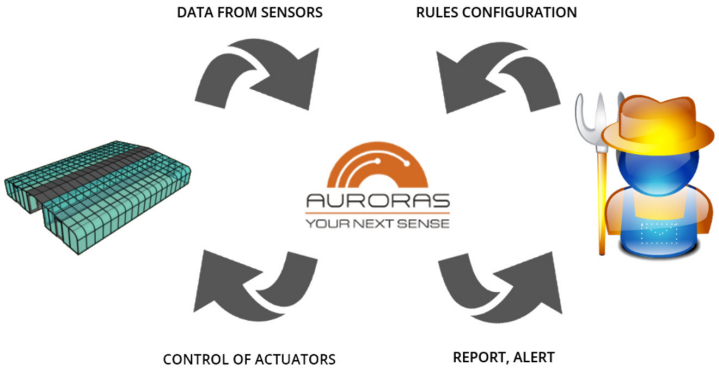


Figure 1. Auroras Greenhouse Implementation

2. State of the art

The internet of things has started to grow exponentially over the last few years and, according to Cisco, in 3 to 5 years we could see an astonishing number of 50 billion of connected devices meaning over 6.5 devices for every person in the world as seen in Figure 2. it is safe to assume they bring an important value to everyone from simple things such as growing plants to more important and complicated devices such as healthcare devices.

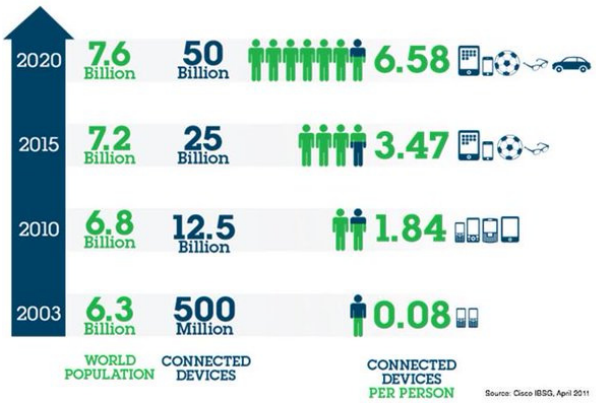


Figure 2. IoT Growth | Cisco

While the traditional ways of growing plants, fruits and vegetables are often used today, they require time and hard work in order for the plants to grow properly. They require the person to move to the physical location of the plants, observe the environment and try to adjust some of the growing factors of the plant, such as watering. One downside being, if the plant is not in a protected spot from wind and excessive sun, the person cannot do anything if needed.

In order to overcome the issues of traditional ways of growing plants, IoT technologies can be used. By placing the plants in a conservatory or a greenhouse the plants are already protected from wind and other external factors. Sensors of different types and different locations such temperature, light and soil moisture would be used. Again, for water, light and temperature, actuators of different types would be used such as a water pump for watering the plants, light bulbs for better lighting and ventilators and heaters for temperature control. A central system will be used in order to read sensors data and control the actuators such as the Raspberry Pi. The data could be sent over different IoT communication protocols such as the 6lowPAN or ZigBee to the user to view.

After researching different types of greenhouse projects [1][2], a prototype design was created and drawn as a reference for when building the actual project as can be seen in Figure 3. The idea behind the chosen box architecture is it can be stacked, multiple boxes can be pun on top of each other, all running on the same board.

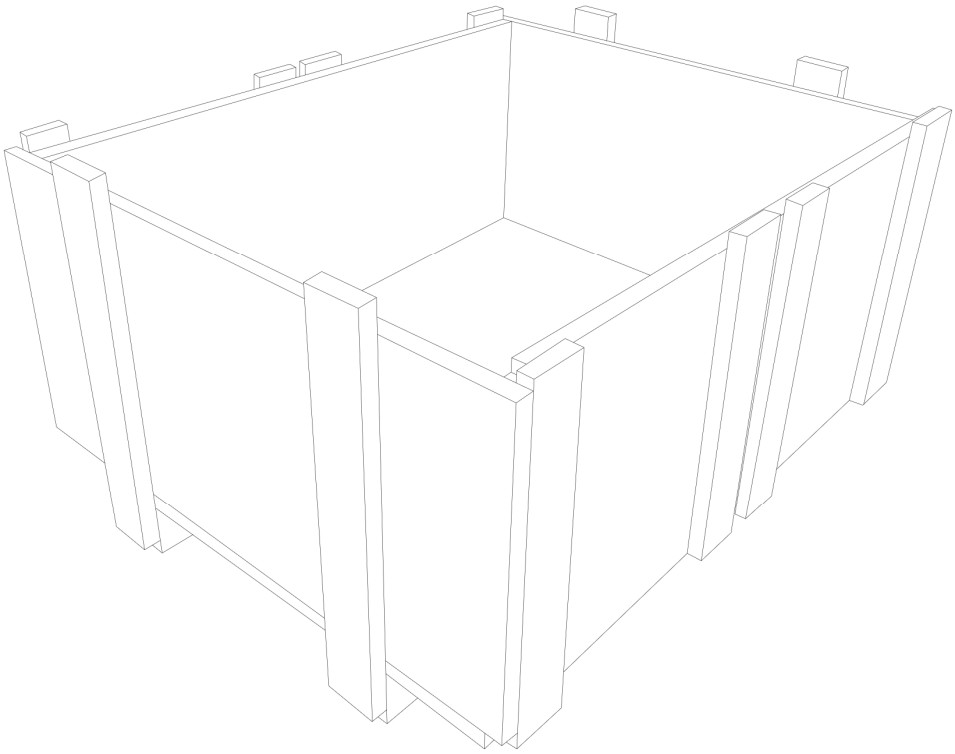


Figure 3. Proposed Greenhouse Design

After building the greenhouse box, the boards, sensors and actuators will be added in order to control together with a water container, soil and the plants that need to be grown inside.

3. Overview

The system communicates wirelessly with external devices such as laptops and desktop computers and serially via short wireless with sensors and actuators. A general overview of how the communication works can be seen in Figure 4.

The main board, the Raspberry Pi, contains all the necessary assemblies and libraries in order to make a common node between the user, the sensors and the actuators. It comes bundled with a handful of servers such as a file server, web server, FTP server and Database server. The web server is used a primary interface for the user to work with the system while the Database server communicates both with the server and the desktop application allowing for user credentials and server logging to be done.

The system can be accessed both locally (i.e. a computer connected to the same network as the system) and remotely from another location. If the user choses to connect to the server from a remote location, on some routers/switches, the ports have to be unblocked in order to allow the devices to communicate.

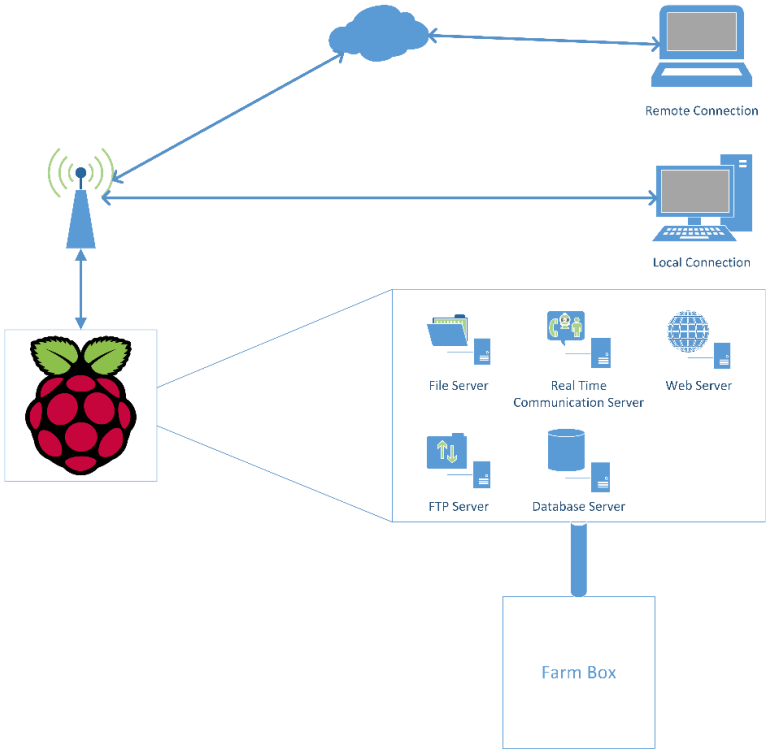


Figure 4. Connection Schema

4. Architecture

The main platform used for this project is a Raspberry Pi Model B V3. Power is supplied from a 220V-5V wall adaptor. The system contains external storages linked to the Raspberry Pi for file storage and backup, a 16x2 LCD for displaying file storage usage and plant box data such as temperature, humidity, light intensity and soil moisture. There are 2 relays, one LCD and an Arduino UNO connected to the Raspberry for reading the analog sensors as shown in Figure 5. The reason for using an Arduino is because the Raspberry Pi natively can only read digital sensors of 3V while the Arduino can read both analog and digital sensors and on both 3.3V and 5V. Sensors connection can be seen in Figure 6. and Figure 12. Extensively, one can replace the Arduino with an external ADC.

4.1. Hardware Architecture

The system uses 4 analog sensors connected to the Arduino: temperature, humidity, light sensitivity, soil moisture and 2 digital sensors connected directly to the Raspberry: temperature and humidity.

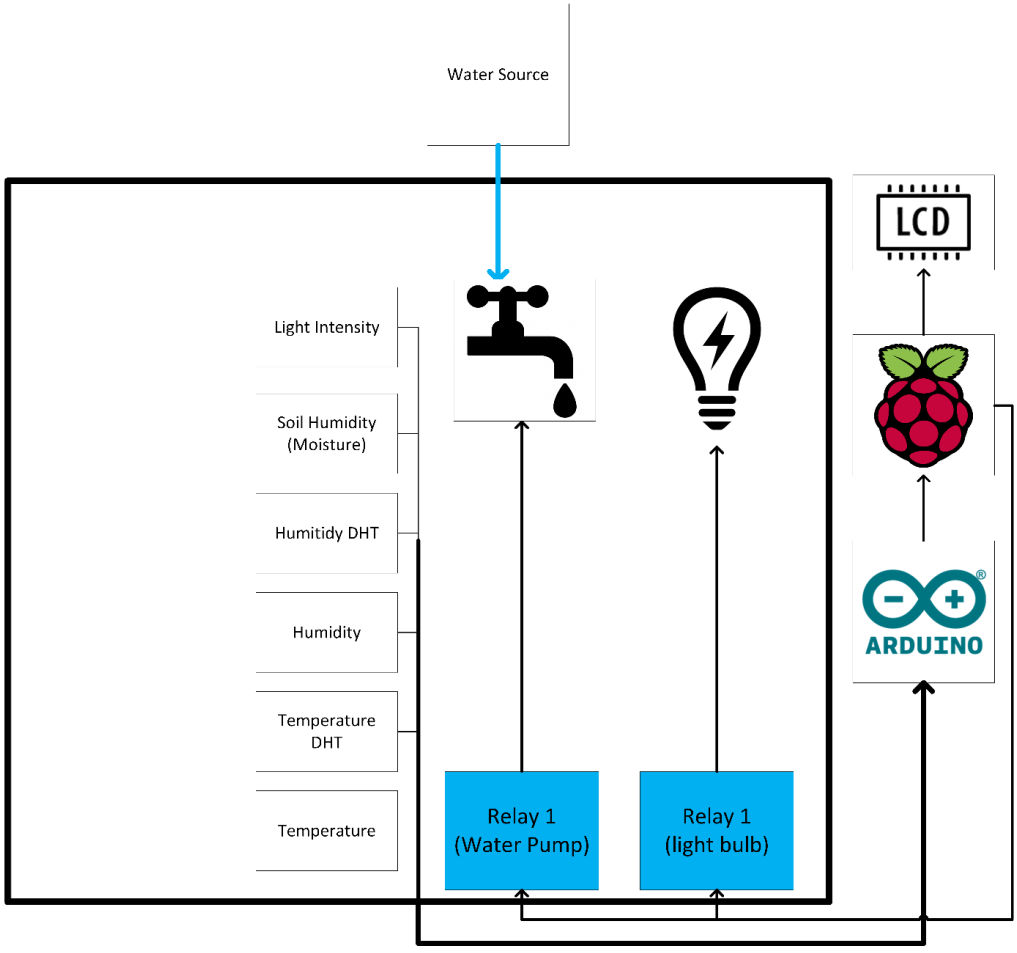


Figure 5. Hardware Schema

The circuit for the Raspberry Pi is presented in Figure 6. showing how the sensors, the LCD and the relays are connected. The DHT22 consists of two digital sensors, one for humidity and one for temperature and need to be connected to the 3V3 pin because the Raspberry Pi GPIOs are only 3.3V tolerant.

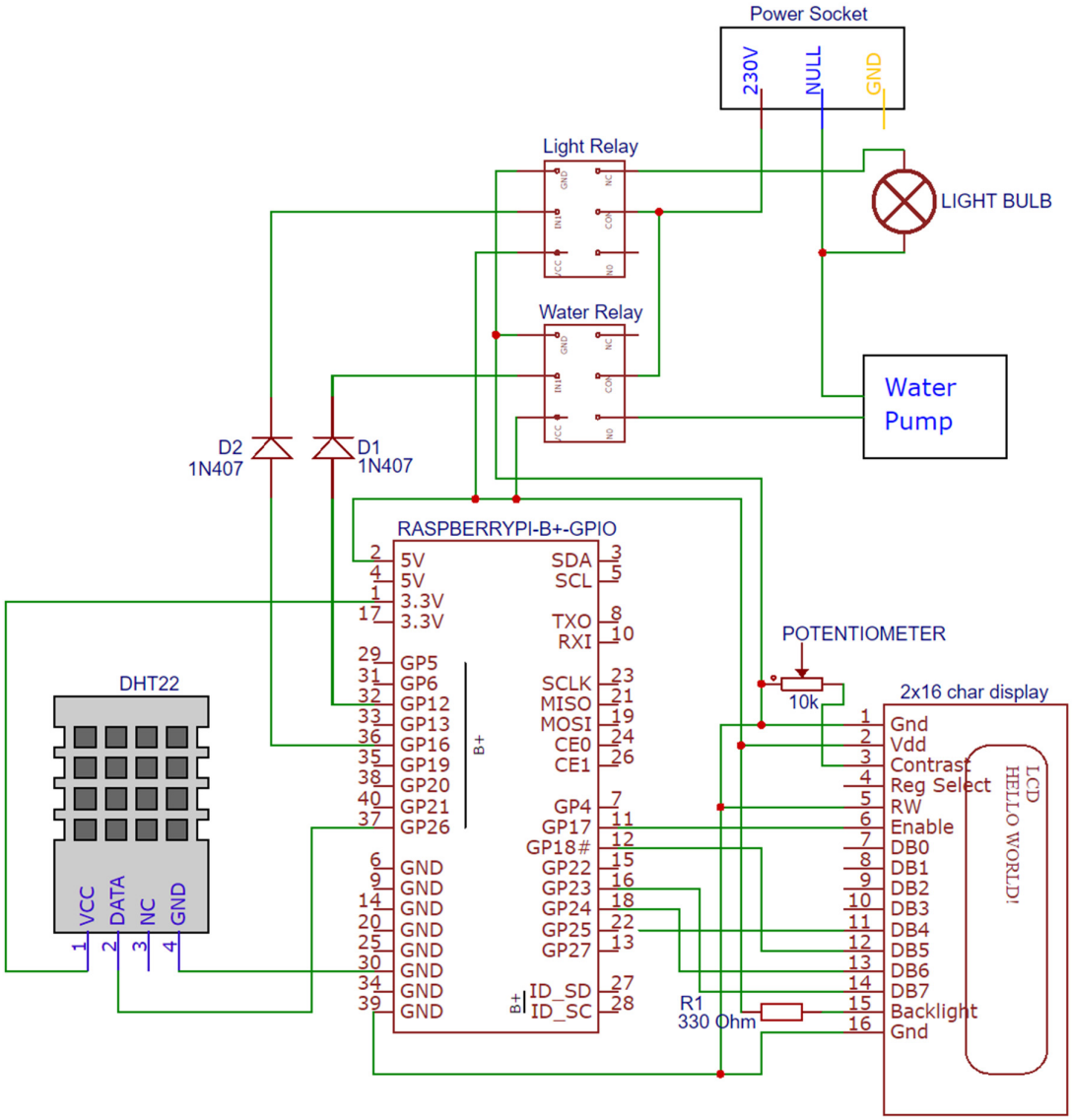


Figure 6. Raspberry Pi Sensors Connection

For driving large loads that require AC such as the light bulb and the water pump we use relays controlled by the Raspberry Pi. Relays function by creating an electromagnetic field using a coil that is activated by a signal. When turning on the relay, there is no problem, but when turning it off, there is still current in the coil that can return back to the Raspberry's GPIO and destroy the GPIO or in worst cases the entire board. For solving this, we use a diode in order to keep the current to flow back into the board.

4.2. Software Architecture

The system uses a variety of software, libraries and servers in order to produce the desired outcome. The Raspberry software architecture layout is shown in Figure 7. The board makes use of the Apache bundle which includes email, HTTP, MySQL and PHP support while separate libraries such as ProFTPD takes care of FTP and SFTP.

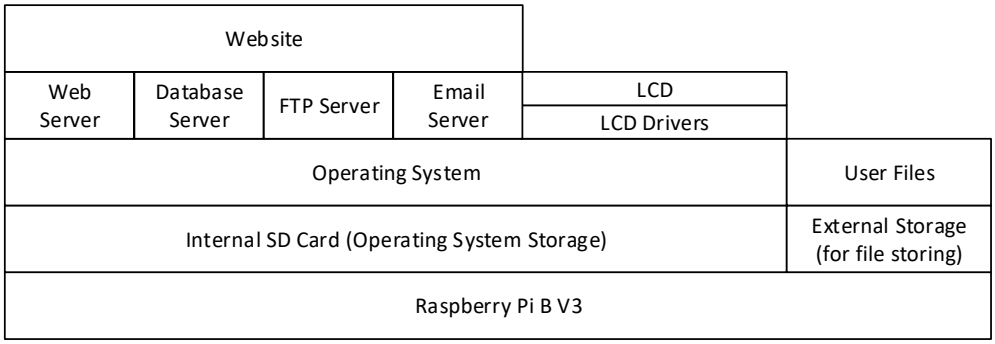


Figure 7. Raspberry Pi Software Layout

File storage is done on the external USB Flash Drive connected to the Raspberry Pi with each user having its own directory with a path of length 20 generated at random and stored in the database. One user cannot access another's user files unless provided that users password. In order to do this, each user has an account protected by a password.

User credentials such as username, password, email and directory path, with the password being encrypted with *bcrypt* are safely stored in the database. There is another field in the database table as shown in Figure 8. called *Admin* with an initial value of 0 that can be changed only by the systems administrator. This field denotes if the user has administrative access to the server or not.

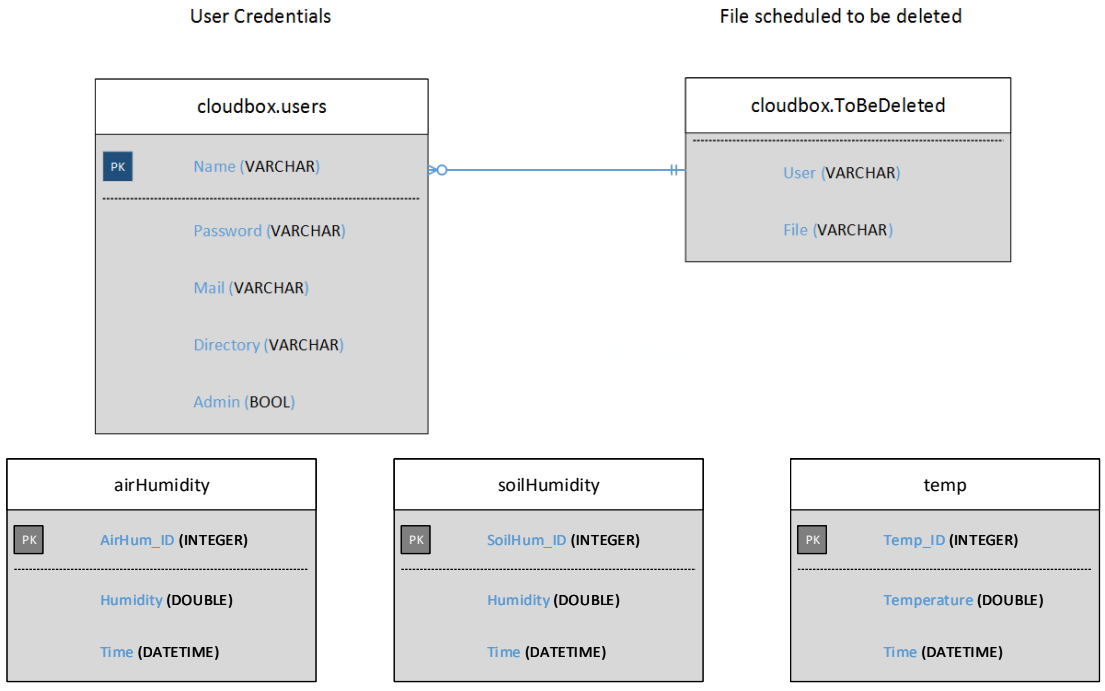


Figure 8. Database

5. Implementation

5.1. Hardware Implementation

Before beginning the development process, the actual greenhouse box had to be built. Having the designed drawing as a reference, in Figure 9. the resulting box can be seen. As it can be seen, it is a very close design as the prototype one.

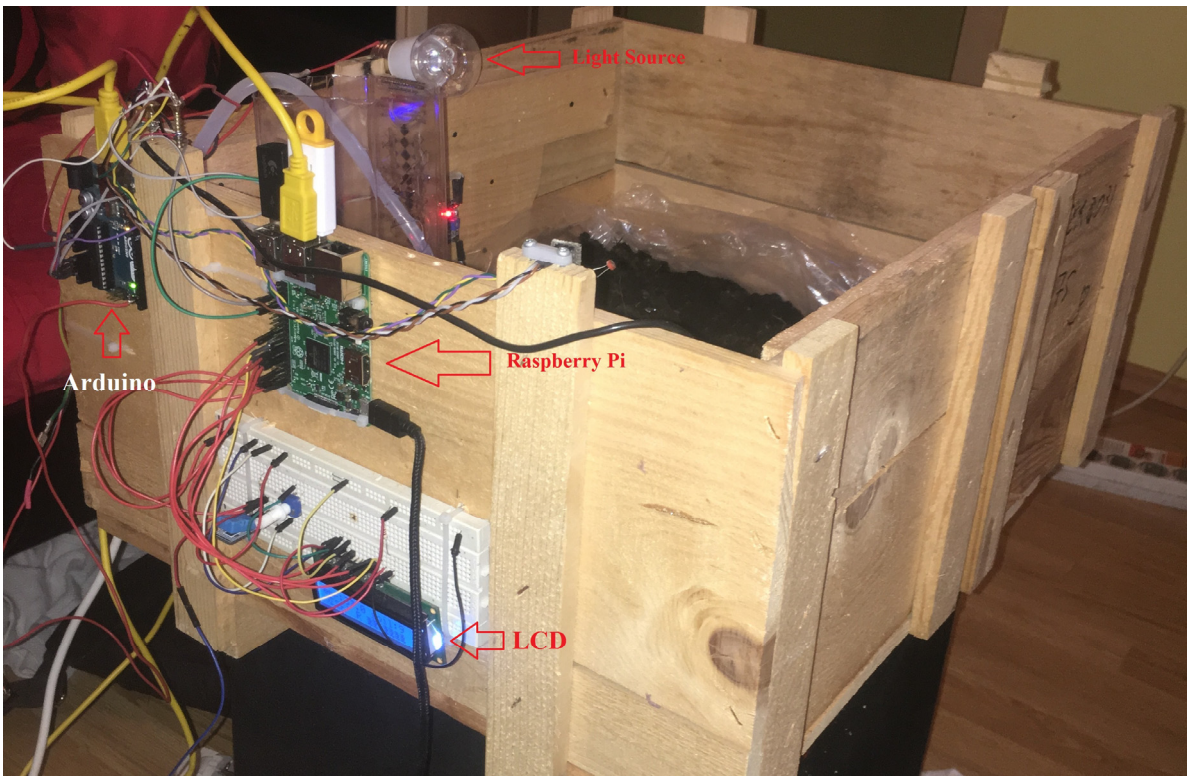


Figure 9. Greenhouse Box

Water is provided via a water pump having the input connected to the water container and passing water through the underground water pipes placed inside the box as shown in Figure 10. The red lines represent the underground pipes, the blue dots represent pipe outputs while the blue circles represent the watering radius of each output.

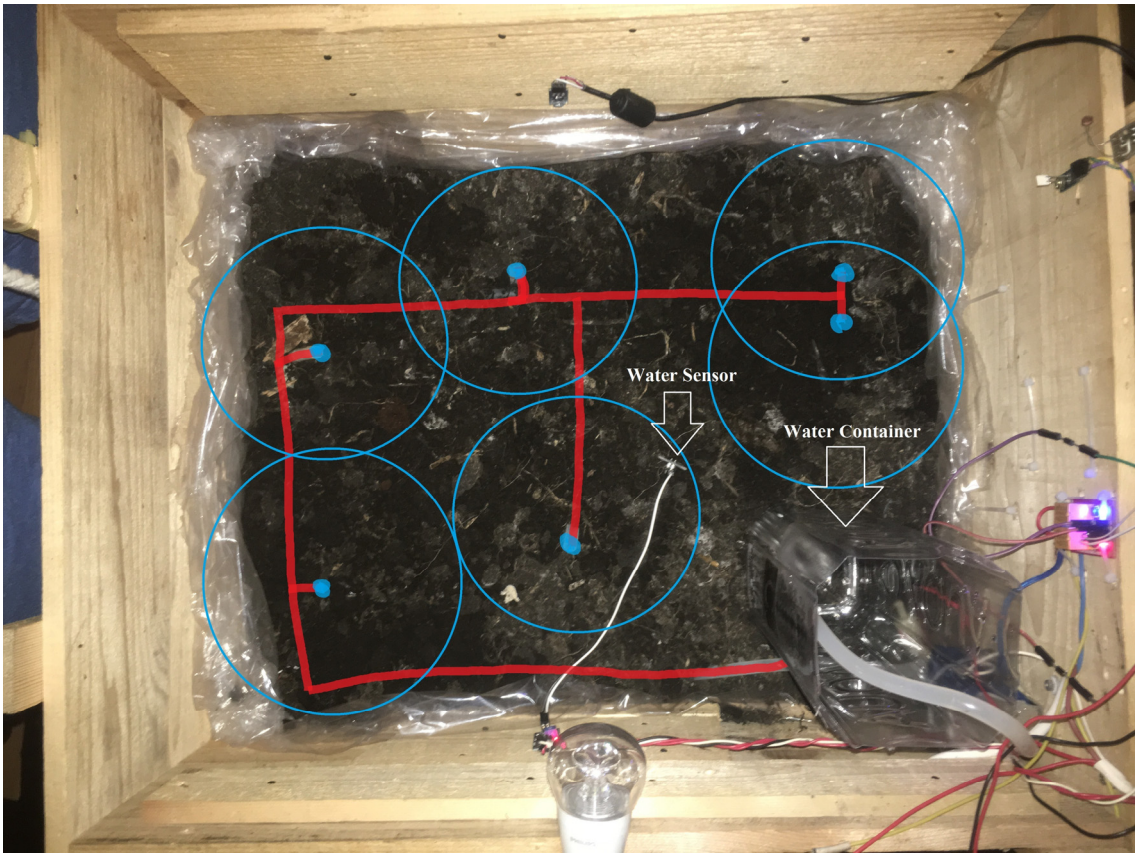


Figure 10. Greenhouse Watering

There are 5 hardware elements connected to the main board: external USB drive, two relays, one DHT22 sensor (Figure 11.a) and one Arduino Uno board. The Arduino itself has 4 other hardware elements (sensors) connected: soil moisture (Figure 11.b), air humidity (Figure 11.c), light intensity (Figure 11.d) and temperature (Figure 11.a).

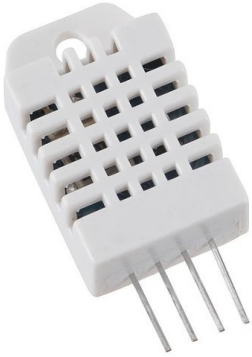


Figure 11.a DHT Sensor

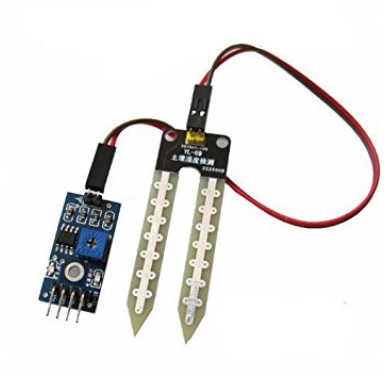


Figure 11.b Soil Moisture Sensor

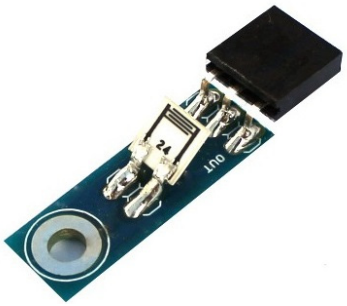


Figure 11.c Humidity Sensor

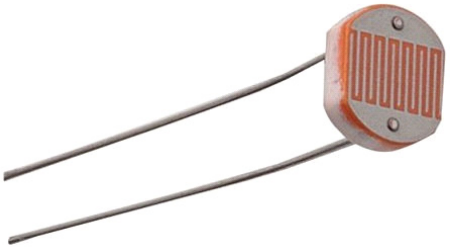


Figure 11.d Photo resistor

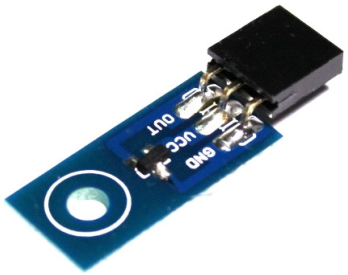


Figure 11.e Temperature Sensor

The sensors are connected to the Arduino board as shown in Figure 12.

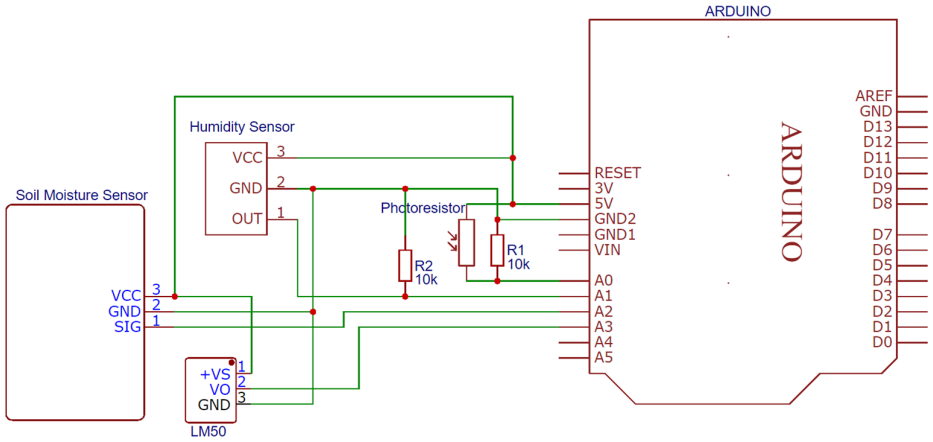


Figure 12. Arduino Sensors Connection.

The USB drive used is a high speed (*5 Gbit/s*) flash drives that stores data outside the Raspberry Pi for two main reasons. One is storage capacity as we can increase this to as much as we want by adding bigger drives or even more devices in RAID or NAS modules, and second, is file security, as the files will not be lost or damaged if something happens to the board, the server or on the software side.

The LCD is a 16x2 display that communicates serially with the server through Raspberry's GPIO pins as shown in Figure 13. and in Figure 14. and is placed on the outside of the box in order to provide exterior information on the values inside. It supports 4-bit and 8-bit transmission mode. We

are using the 4-bit mode that takes twice as much to send data than the 8-bit mode in order to use less pins to connect to the main board.

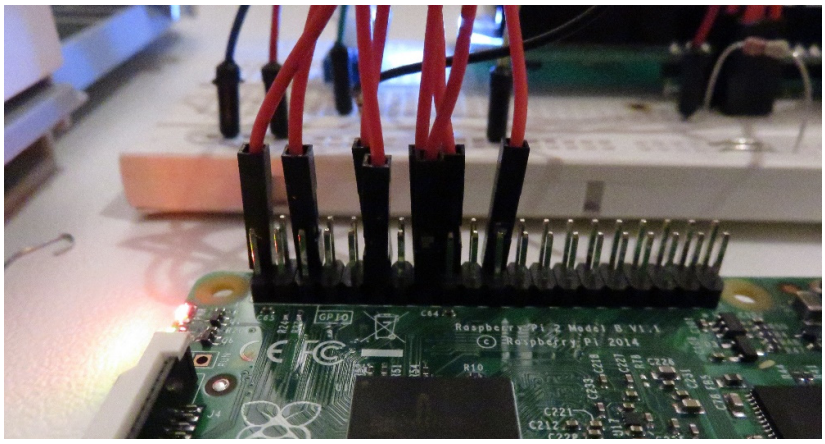


Figure 13. LCD Connection to Raspberry Pi Board

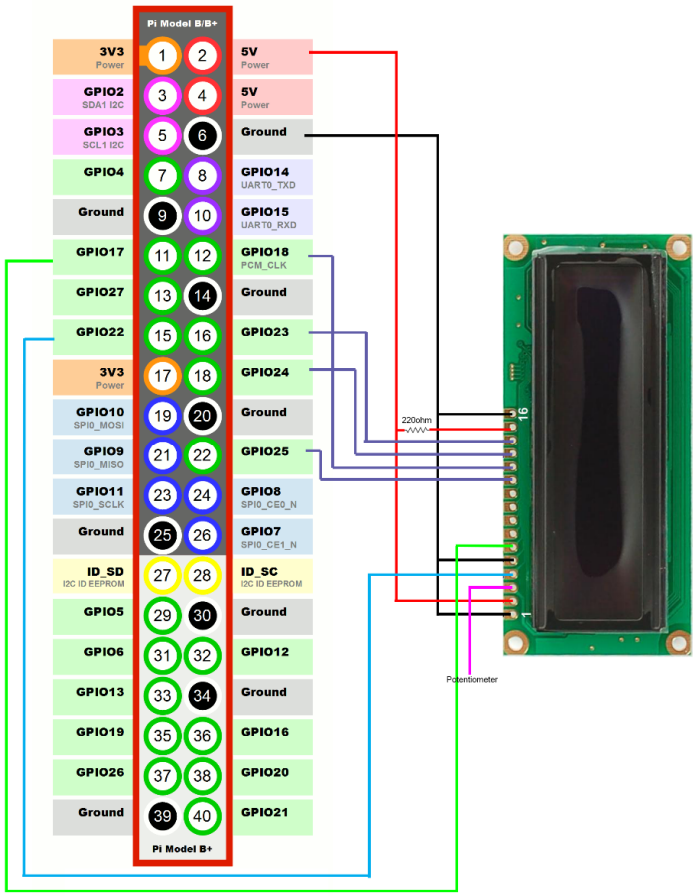


Figure 14. LCD Connection Schematic to Raspberry Pi

5.2. Software Implementation

The system comes with multiple ways for the user to interface with, the LCD, the website, the desktop application and via remote or SSH given administrative credentials.

The DHT sensor is a digital sensor consisting of two actual sensors, one for air humidity and one for temperature. It functions by sending out 40 bits of data to the board split into 3 segments. The first 16 represent the relative humidity of the sensor followed by 16 bits of temperature and last 8 bits of checksum. The bits are decoded into decimal number and passed through Formula 1.a and Formula 1.b in order to get the actual sensor value. The checksum is the binary addition result of the temperature and humidity as seen in Figure 1.c.

$$Relative\ Humidity_{10} = \frac{Relative\ Humidity\ Data_{10}}{10_{10}}$$

Formula 1.a Humidity Formula

$$Temperature_{10} = \frac{Temperature\ Data_{10}}{10_{10}}$$

Formula 1.b Temperature Formula

$$Checksum_2 = Relative\ Humidity_2 + Temperature_2$$

Formula 1.c Checksum Formula

The LCD that is present in the project is mainly controlled from the website but it can also be launched by anyone with administrator rights. The script will parse the data from the external drive and will display the results. In order for us to use the display while someone else is using the website for uploading and/or downloading files we need to use multitasking. The reason behind this is that the program running on the Raspberry Pi that gathers data from the system and outputs it on the display is a continuous process thus can mess up with other continuous processes, such as uploading a file. The solution is simple, use multitasking by running it as a background job that is not interacting with the user. This is done by starting the program, interrupting it by sending a SIGTSTP signal (CTRL-Z) and typing **bg**, thus sending the program in the background. When SIGSTOP is sent to a process, the usual behavior is to pause that process in its current state. The process will only resume execution if it is sent the SIGCONT signal. An example of using and launching the LCD is shown in Figure 15.

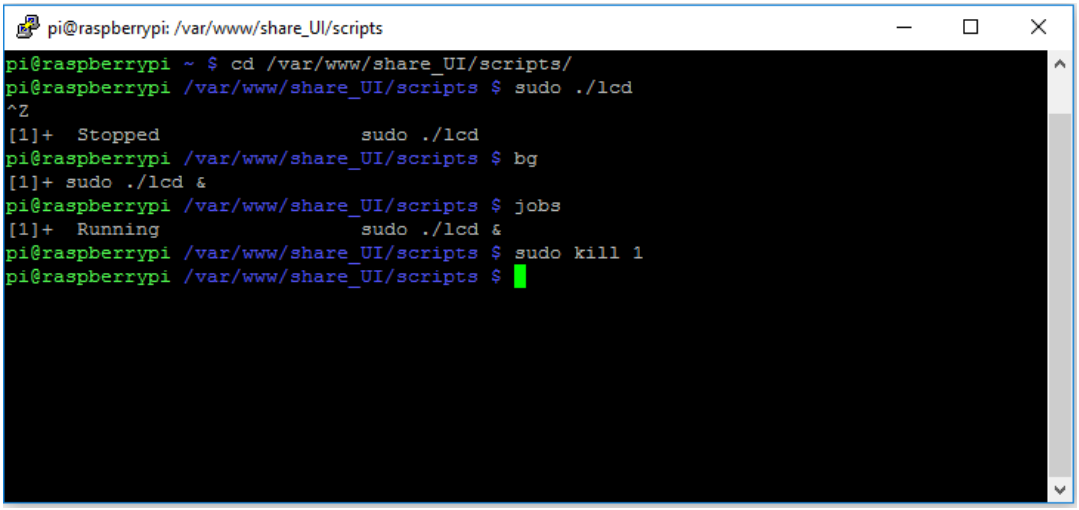


Figure 15. LCD terminal launch

5.2.1. Raspberry Pi Server

The main code that runs on the system is written in python. It allows the system to connect all the parts together. It performs the following functions:

- Creates a socket from the *socket* library for external communication such as the desktop application.
- Connects to the Arduino serially with the *serial* library in order to get sensor values.
- Computes received sensors values and acts on relays using the GPIO functions from the *RPi.GPIO* library.
- Connects to the local database and inserts the values from the sensors every one hour for data logging.

The FTP and SFTP allows the user to upload, download and delete personal files to and from the server via the website or the desktop application.

5.2.2. Website

The website is mostly written in PHP that has a direct and local connection to the python socket and the database and is stylized using HTML and CSS plus AJAX functions to create uploading, downloading bar and name suggestions.

5.2.2.1. Login Page

The first thing that will be displayed on the website is the login page. The user can then login with an existing account (Figure 16.a), create a new account (Figure 16.b) or recover a lost password (Figure 16.c).

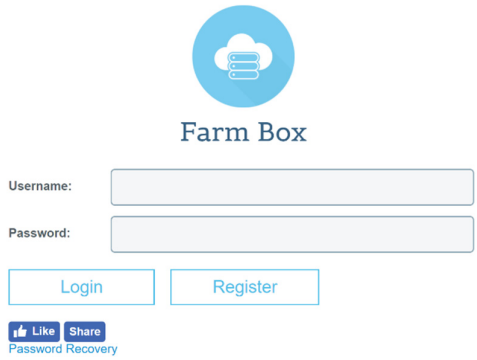


Figure 16.a Login Page

Figure 16.b Register page

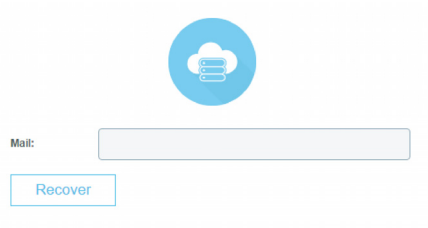


Figure 16.c Recover Lost Password

5.2.2.2. Main Page

After the user performs the login or registration action, the main page will be displayed as seen in Figure 17.

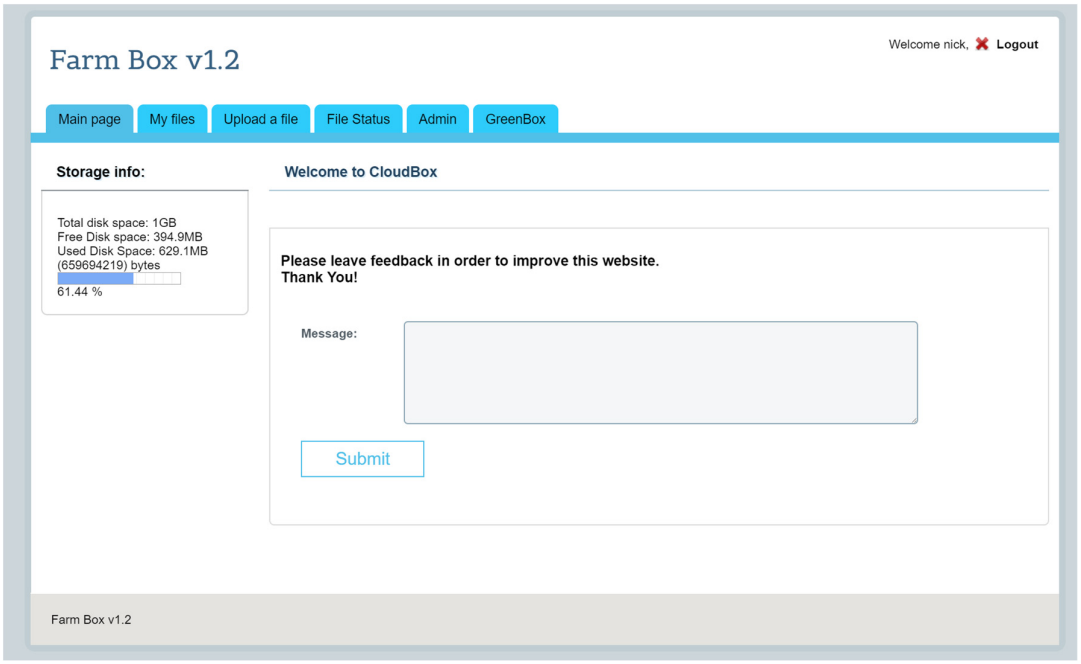


Figure 17. Website Header

The main page implementation diagram can be seen in Figure 11.a and the registration page diagram is shown in Figure 18.b and the red errors are explained in Table 1. and they appear in the system logs or on user page.

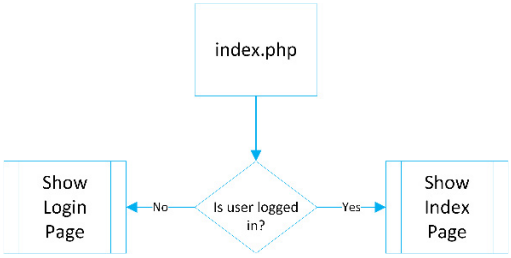


Figure 18.a Index Page Flow Diagram

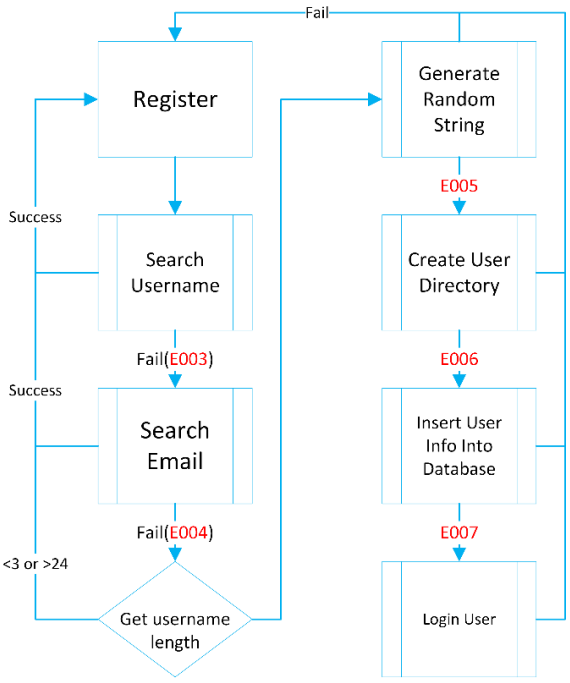


Figure 18.b Register Flow Diagram

Table 1. Error Description

Code	Error Description
001	Selected file is not OK. File too big, not enough free space on the account or the extension is forbidden.
002	File Upload error. Something went wrong while uploading the file.
003	Username already registered.
004	Email already registered.
005	Directory name already exists.
006	Error while creating user directory.
007	Database error. User could not be created.
403	Forbidden access.
404	File not found. Incorrect link.
503	Service unavailable. Server side problem.

After the user has logged in, they will be redirected to the main page. Here, on the left part of the screen, as shown in Figure 92. is displayed the storage limit, free space left for storage and how much storage is used in a readable way (MB, KB, etc.) as well as in bytes and a horizontal bar that gives a visual representation of those numbers.

On the main page, there is also present a feedback box that any user can use to give feedback or to send a message to the system administrator. These messaged are then emailed by the server directly to the administrator's email.

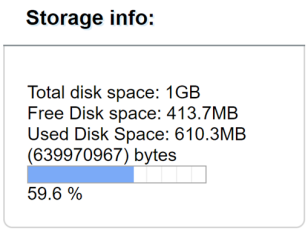


Figure 19. Storage Info

5.2.2.3. My Files Page

Once on the main page, the user has multiple pages to go to. One of them is the **My Files** page that contains all the files of the user as shown in Figure 20.

On the left container, the user can see the file usage just like in the main page plus how many files of each category there are.

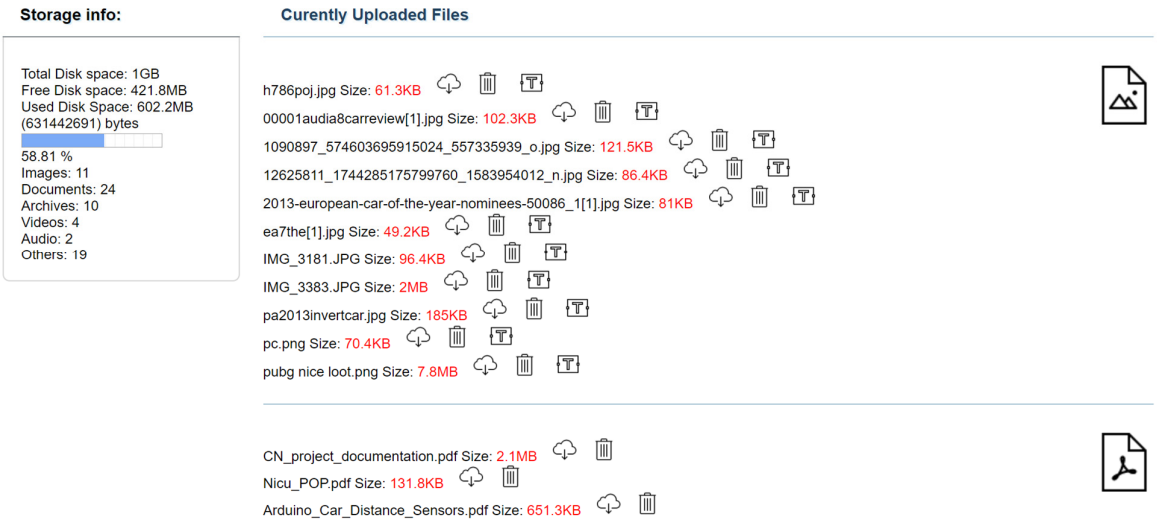


Figure 20. My Files Page

On this page, the files are placed into categories of *Images, Videos, Documents, Archives, Audio* and *Others* and the user can choose to download a file using the button or delete a file by using the button.

In Table 2. is presented the file extension by which the files are distributed into categories and their corresponding images shown on the right part of the category.

Table 2. File Classification

File type	File Extension	Category Image
Images	.jpg .jpeg .png .bmw .gif	
Documents	.doc .docx.pdf .pdf_ .xls .xlsx .mpt .msw .word	
Archives	.rar .zip .zipx .7z ,cab .shar .tar .gz .s7z .tgz .tlz	
Videos	.mp4 .avi .mov .flw .mkv .wmv .3gp	
Audio	.mp3 .aac .wav .wma	
Other	Any other file extension	

5.2.2.4. File Upload Page

The File Upload page very simple and intuitive as shown in Figure 21.a. It consists of only two buttons, one for choosing the file that is desired to be uploaded which by clicking it opens a file browsing dialog, while the other button, the **Upload** button will commence to upload the file to the server as in Figure 21.b if all the conditions are met: there is enough space on the server for the file and the user has enough space in its personal directory.

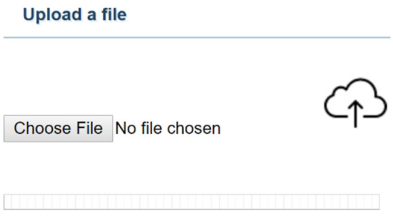


Figure 21.a Upload Page

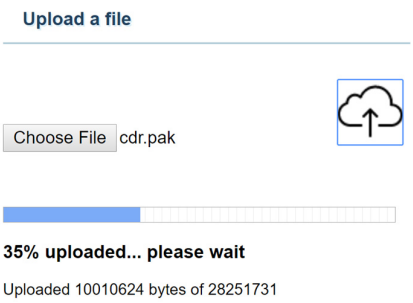


Figure 21.b Upload in Progress

5.2.2.5. File Status Page

The File Status webpage is only an informative one, it displays all the information from the *File Upload Page* but, additionally it adds a more user-friendly look by introducing a chart for a visual feeling on the used and free space left for the user as shown in Figure 22.a plus two pie-charts as shown in Figure 22.b and Figure 22.c.

Figure 22.a Used Space vs. Free Space

The pie-charts used are created by using the Google Charts API and are responsive: when hovering the mouse over the chart it will display a textbox with information about that part of the chart. In Figure 22.b is illustrated how many files the user has in each category, while in Figure 22.c is shown how much space each category takes on the server.

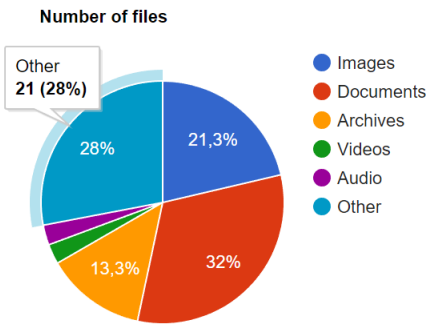


Figure 22.b Files Distribution

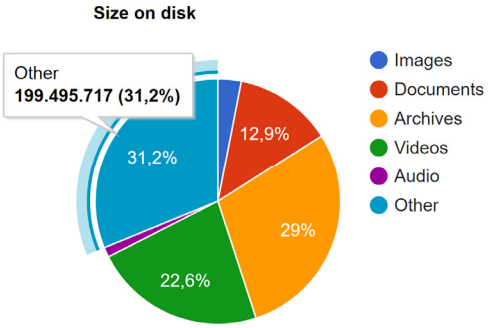


Figure 22.c File Size Distribution

The website allows for user friendly interface that displays data sensors and relays state using simple yet intuitive icons such as in Figure 23. and file manipulation as well as administrative control of the system.

5.2.2.6. Plant Data Page

The Plant page is the only page on the website that allows the user to see data from the server as shown in Figure 23. It allows the user to visualize all the parameters from the box such as temperature, humidity, relays state and others.

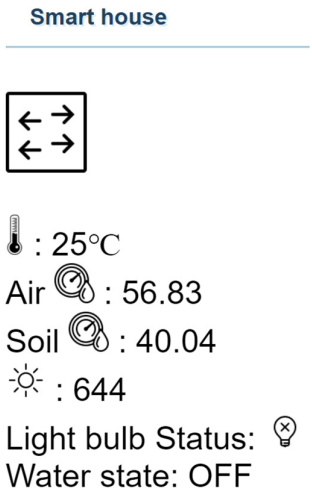


Figure 23. Sensor values on website.

The icons from 23. represent:

-  - connected, communicating with the Raspberry Pi.
-  - disconnected, connection to socket failed.
-  - Temperature.
  -  - Increasing Temperature.
  -  - Decreasing Temperature.
-  - Light level.
-  - Air Humidity/Soil Moisture, as stated on the web page.
-  - Light Bulb OFF.
-  - Light Bulb ON.

5.2.2.7. Admin Page

The Admin page, as the name suggests allows for system administrators to visualize and modify user parameters such as name, password, directory path, email and administrative level as seen in Figure 24.b, with the first line on the page being the number of user on the website. While searching for a user to modify, the website gives suggestion to the users for easier user search. This is done by having a background AJAX script that searches for the semi-complete username in the database after each character written in the box as seen in Figure 24.a.

This page also allows the administrator to run server-side commands, such as modifying the user storage limit or to run the LCD program as shown in Figure 24.c and Figure 24.d.

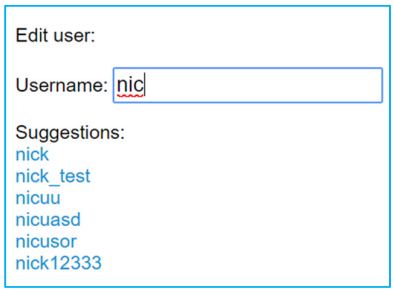


Figure 24.a User Search

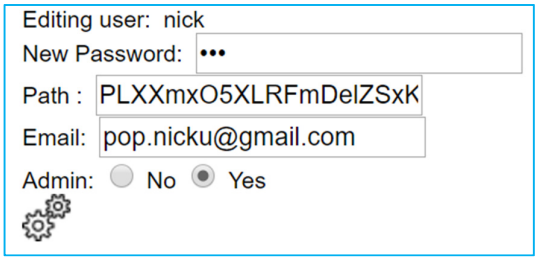


Figure 24.b User Edit

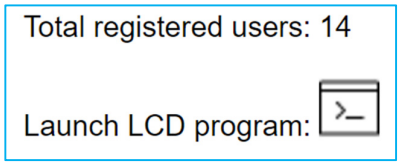


Figure 24.c LCD Launch

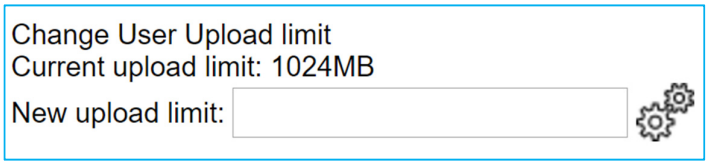


Figure 24.d Upload Limit Edit

After running the LCD program, the LCD located on the Farm Box will be initialized showing the used and free space remaining on the external drive for file storage as in Figure 25.

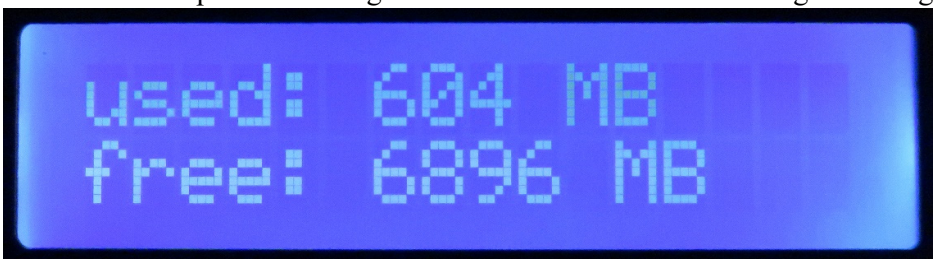


Figure 25. LCD Output

The last thing on the page that the administrator can do is to modify the per-user upload limit from the default 1 GB to anything they want as shown in Figure 24.d.

5.2.3. Desktop Application

The Farm Box comes with a desktop application that offers personal file usage on the server and control over the system. It is written in C# and can only be used on windows compatible devices. The main window can be seen in Figure 26. and Figure 27. and contains a history graphic, sensor values and control and file management.

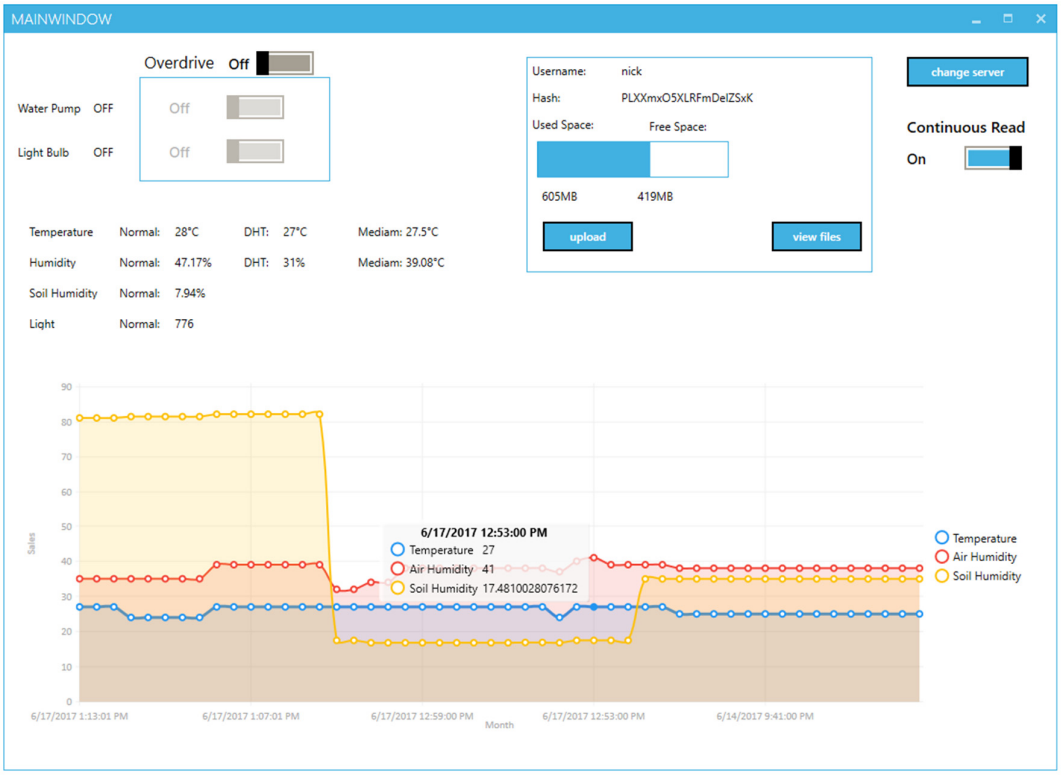


Figure 26. Desktop Application

The application theme uses the *Metro Mahapps* library in order to implement the UI that allows us to use the Windows 10-like elements. For graph plotting, the application uses the *LiveCharts* library.

The application performs the following functions:

- Connects to the Raspberry socket in order to communicate with the server and can be connected to multiple servers with switching between them is at a push of a button.
- Performs user login and user registration as.
- Gathers previous sensors data from the database (Figure 8.) in order to create a history graph.
- Can overdrive the relays for water and light.
- Uploading, downloading and deletion of personal files from the server.

We can observe that the desktop application has multiples elements, from box data to file management with the general program flow presented in Figure 28.a. In Figure 28. we can observe the various parts of the application. Now, the user can use all features of the application.

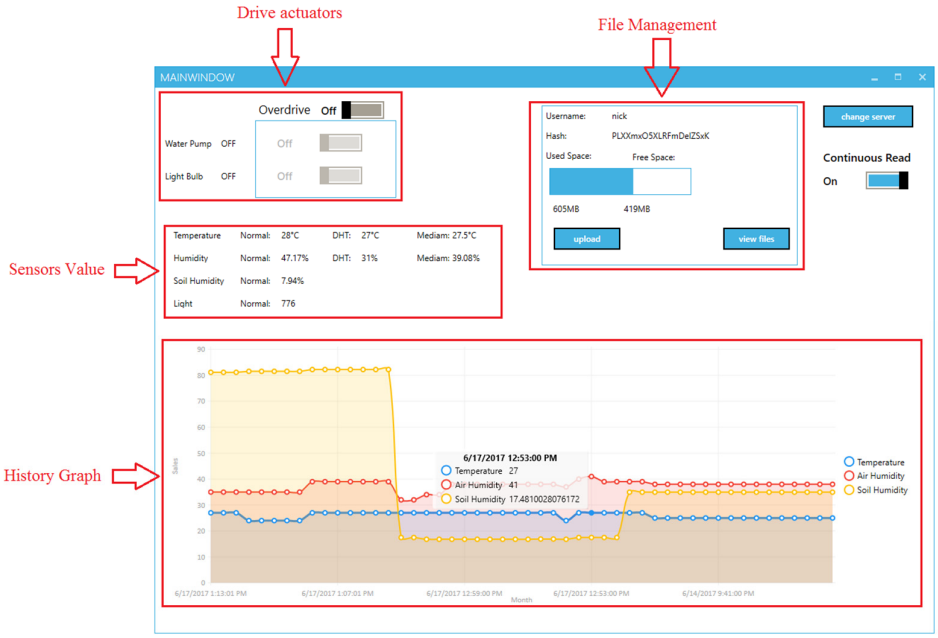


Figure 27. Desktop Application Explained

The history graph shows the variation of sensors data from the box. The python server located on the Raspberry collects data from the sensors and inserts them into the database every hour in order for the C# program to fetch them later and plot into the graph. The graph will display only the last 50 entries in the database but this can be easily modified by any administrator with access to the server.

The sensors value section gathers data directly from the python server and shows them on the application window. There are two sensors for temperature and humidity, an analog one *(Normal)* and a digital one *(DHT)*, and the software performs a median on those two values giving a more reading of the actual value.

The overdrive section allows the user to control the water and light actuators manually. The overdrive feature is disabled by default as the server controls them based on sensor values and the actuators, and cannot be used until the overdrive is turned ON. After switching the overdrive ON, it will remain on the ON position until the user switches it back OFF or the server restarts. The user can control each relay individually by a click of a button. Data reading and writing is presented in Figure 22.b. Color legend is present in Figure 22.c.

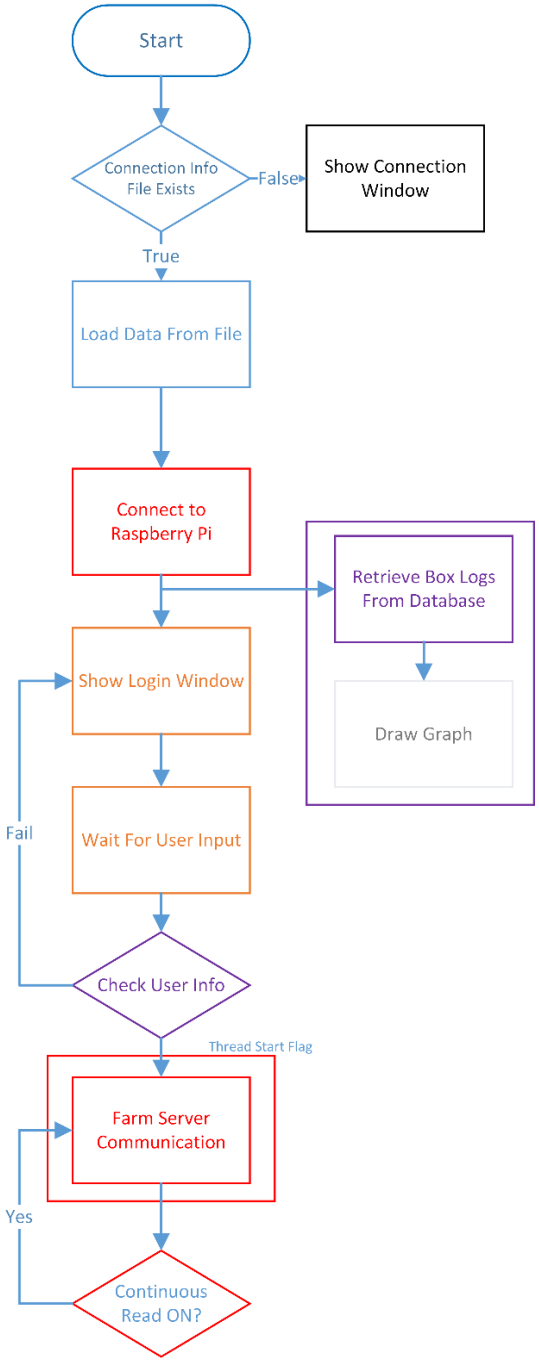


Figure 28.a General Program Overflow

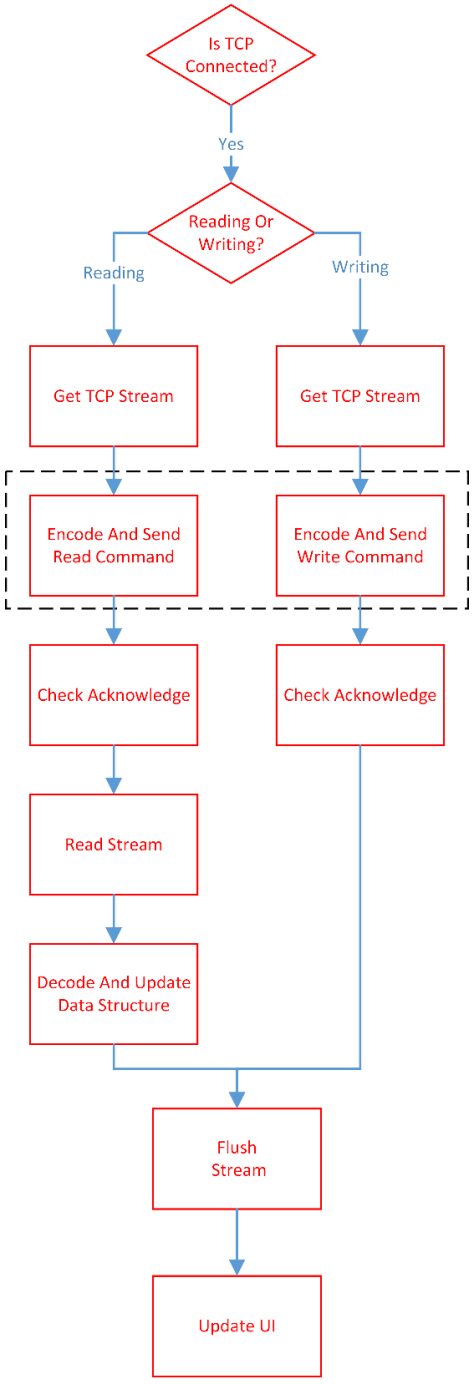


Figure 28.b Read and Write Flow

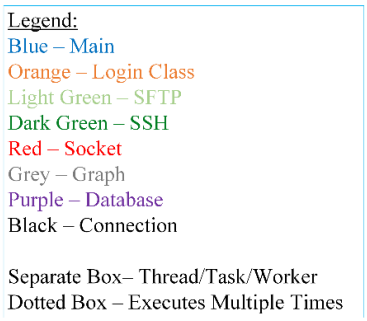


Figure 28.c Legend

When starting up the program, the first thing it does is searching for the file containing the connection data (IP and port). If the file cannot be found, the connection window (Figure 26.) will be displayed in order for the user to input the connection data and connect to the server.

After connecting to the server, the login window (Figure 33.) is displayed for the user to insert their credentials and login while a separate thread pulls data from the database for plotting the history graph and a separate task waits for the login or register event.

When the user finishes the login process, the communication with the python server on the Raspberry Pi is initiated and, by default, starts reading data and updating the UI through its own Dispatcher.

In our project, time is less important than hardware and proper communication, we don't need real-time communication with the device and thus, time redundancy[3] can be applied by performing the data transmission block multiple time in order to make sure the data is sent to the server and is illustrated in Figure 29.a. This way, errors prone to appear during data transfer between the user and the server are minimized.

Figure 29.a. Data Transmission Redundancy

This is done in order to avoid corrupted or incorrect data from being sent over the channel. An example of data transmission is provided in Figure 29.b and data receiving in Figure 29.c.

```
connected to server
Sent: {"light":"null", "command":"overdrive", "actuator":"water_pump", "value":"ON"}
Received: {"water_status": "ON"}: , different: False
connected to server
Sent: {"light":"null", "command":"overdrive", "actuator":"water_pump", "value":"ON"}
Received: {"water_status": "ON"}: , different: False
connected to server
Sent: {"light":"null", "command":"overdrive", "actuator":"water_pump", "value":"ON"}
Received: {"water_status": "ON"}: , different: False
```

Figure 29.b Overdrive Send Message

```
connection from ('192.168.100.2', 57910)
Received:  {"light":"null", "command":"overdrive", "actuator":"water_pump", "value":"ON"}
Received overdrive
Overdrive water to:  1
Send: {"water_status": "ON"}
no data
closing connection...
connection from ('192.168.100.2', 57911)
Received:  {"light":"null", "command":"overdrive", "actuator":"water_pump", "value":"ON"}
Received overdrive
Overdrive water to:  1
Send: {"water_status": "ON"}
no data
closing connection...
connection from ('192.168.100.2', 57912)
Received:  {"light":"null", "command":"overdrive", "actuator":"water_pump", "value":"ON"}
Received overdrive
Overdrive water to:  1
Send: {"water_status": "ON"}
no data
closing connection...
```

Figure 29.c Overdrive Receive Message

The desktop application will send a command 3 times to the server and then will listen for a response. If there is no response, the data will be sent again and, if there is a response, the received data gets compared in order to make sure the communication was correct.

The file management section shows user and file information from the system such as directory name and file usage. The *hash* represents the 20 characters long directory location that, when registering from the application, is generated at random using the *RNGCryptoServiceProvider* class that comes with the .Net framework under the *Security.Cryptography* reference in order to give a more random value than the ordinary random generator solutions. By default, the system comes with a limit of 1GB of available space for each user, but can be modified by any administrator on the website. The progress bar present in the file management box represents how much space the user is using out of the available one. The **upload** button opens a file browsing dialog for the user to choose what files to upload, while the **view files** button will open the window from Figure 30. and displays all the personal files for that user.

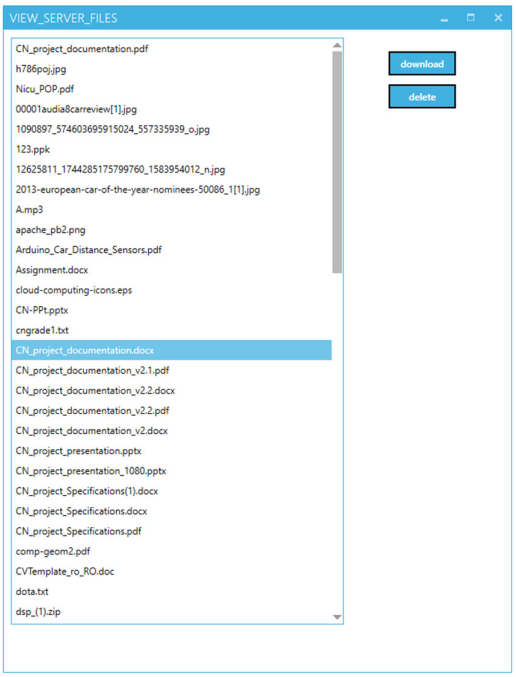
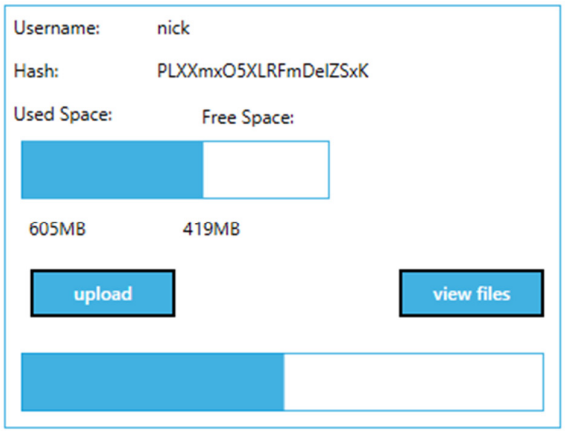


Figure 30. Personal Files Located on the Server

When choosing to upload a file, the file management box will expand downwards and a progress bar similar to the one that shows the amount of used space will appear that updates in real time as the file uploads and represents the percentage of the file uploaded as shown in Figure 31.a. When the system finishes to upload the file, a fly out will appear at the bottom left of the screen notifying the user that the file has been uploaded as shown in Figure 31.b and the file management box will return to its initial size.



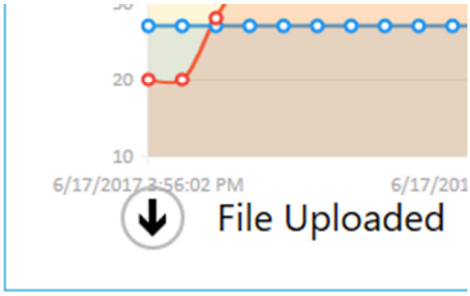


Figure 31.a Uploading in Progress *Figure 31.b Upload Finished*

The **change server** button in the upper right corner of the application will close the main window and open a new connection window as shown in Figure 25. that allows the user to switch between the current server and another one. This window also appears at the program startup if a connection has not been saved before. When checking the **Remember** checkbox, if the connection is successful, will save the connection data on the user's local disk and, the next time the program starts, it will fetch the data from that file and connect automatically.

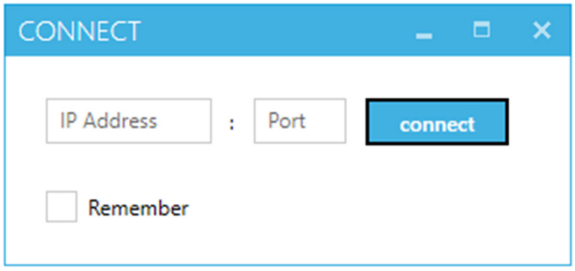


Figure 32. Connect Window

When starting the program, it will start reading data from the server and update the UI accordingly. This can be disabled and enabled at any time by the **Continuous Read** switch located in the upper right corner under the **change server** button.

After the program manages to connect to the server socket, it will show you a login window as shown in Figure 33.a where the user has to input its credentials, username and password, or switch to the register tab as shown in Figure 33.b to create a new account. When creating a new account, the user has to provide a username and email that is not already registered plus a password for lather

authentication. The system will then create the user directory on the server and create a new database entry inside the *users* table.

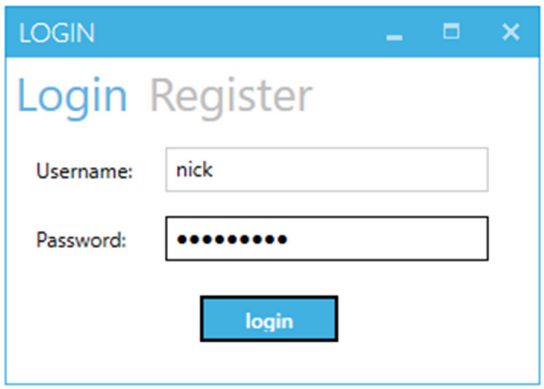


Figure 33.a Login Window

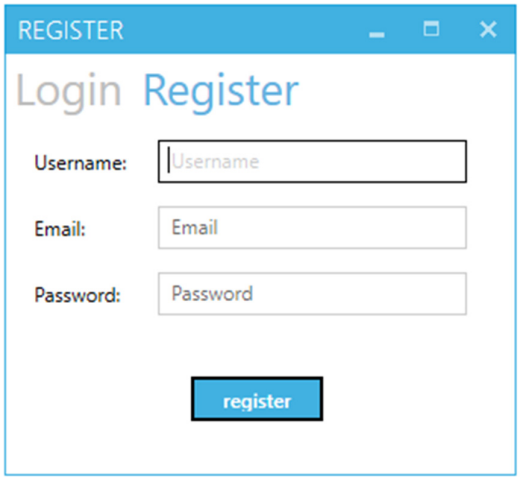


Figure 33.b Register window

After the user has clicked on the **login** or **register** button, the server will check if the credentials are correct and, on success, will show the main window from Figure 26.

5.2.4. Arduino

In order for the Raspberry Pi to read analog devices and sensors, we need to use an external analog-to-digital converter (ADC). This is needed because all the pins on the board are digital (ones and zeros) and cannot read any analog values. The difference between digital and analog[4] is presented in Figure 34.

An ADC functions by sampling the analog signal into digital values and is measured in Hertz [5] as shown in Formula 2. where T is the number of samples per second.

$$sample\ rate[HZ] = \frac{no.of\ readings}{s} = \frac{1}{T}$$

Formula 2. Sample Frequency

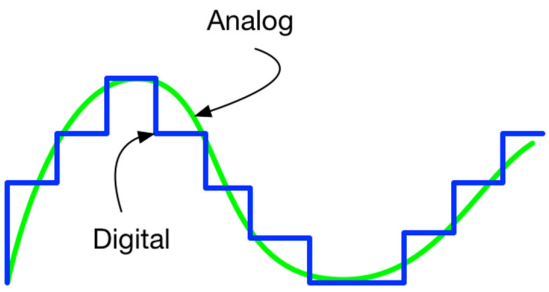


Figure 34. Analog vs. Digital

The digital value is not as accurate as the digital value; it is prone with a margin of error depending on the ADC's resolution. The resolution represents how well an analog value can be represented in digital values. For example, the Arduino has a 10-bit ADC meaning, that every analog value can be represented in the range of $0 \rightarrow 2^{10} - 1 = 1023$. Table 3. represents different ADC resolutions and errors (deviation from actual value) with constant 3.3V and 5V.

Table 3. ADC Resolution

ADC Resolution	Lower Margin	Upper Margin	3.3V Deviation	5V Deviation
8 bits	0	255	0.0129	0.0196
10 bits	0	1023	0.0032	0.0048
16 bits	0	65535	0.00005	0.00007

We can observe that the higher the resolution, the lower the quantization error and higher the efficiency. Using the Arduino, we have an error of ± 0.0048 Volts on our readings which translates to an accuracy of $\pm 0.049\%$ as of Formula 3. with a sampling rate of $10kHz$ (once every 100 microseconds).

$$Accuracy\ percent[\%] = \frac{Deviation * Resolution}{100}$$

Formula 3. ADC Accuracy

We chose to use the Arduino because it is fairly accurate and reliable. It can be replaced with a higher resolution dedicated ADC or with another microcontroller with built-in ADC like the ATMEGA16/32.

After fetching data from the sensors, the Arduino program applies Formula 4. on the temperature sensor value and Formula 5. on the soil and air humidity sensor values while leaving the light values in their raw form.

$$Temperature\ [°C]\left(\left(\frac{Temperature\ Sensor\ Value}{1024} * 5\right) - 0.5\right) * 100$$

Formula 4. Light Intensity Calculation

$$Humidity\ [\%] = \frac{Humidity\ Sensor\ Value}{1024} * 100$$

Formula 5. Humidity Calculation

After gathering and computing of all sensors, these values are transmitted serially to the Raspberry Pi at a rate of 9600 bits/s and the serial buffer is flushed after each transmission in order

for the system to not get blocked if anything fails. As an example, Figure 35. represents how the data sent looks like.

```
['Temperature:', '26']
['Humidity:', '67.25']
['Soil_Humidity:', '40.50']
['Light:', '864']
```

Figure 35. Sample Arduino Transmitted Data

The server will listen for any serial message coming through '/dev/ttyACM0' (this is the port the Arduino will connect to the Raspberry Pi) and, when available, will commence to read and process the data coming in.

6. User Testing

6.1. Single User Testing

The tests are done on the machine specified bellow, using the Google Chrome browser. Each page was tested 5 times using the Developer Tools provided by Google Chrome and a mean average was computed in milliseconds. The results are as follows:

- System used for testing
  - CPU: i7-5820k 6 core @ 4.43 GHz
  - RAM: 16GB DDR4 @ 2999.9 MHz
  - Storage: M.2 NVMe SSD @ 6Gb/S
  - Browser: Google Chrome

Table 4.a Average Page Loading Time

Page	1st run	2nd run	3rd run	4th run	5th run	Average [ms]
index.php	182	177	184	186	288	181.2
list.php	210	202	200	209	2011	206.4
stats.php	753	585	615	537	844	667.6
select.php	184	199	184	191	191	189.9
logout.php	93	96	98	94	95	95.2
login.php	104	98	98	99	101	100
register.php	95	103	100	111	100	101.8
passrecover.php	97	104	94	95	98	97.6

Table 4.b Data Downloaded Per Page

Page	Size
index.php (logged in)	139 KB
index.php (not logged in)	276 KB
list.php	199 KB
stats.php	510 KB
select.php	173 KB
register.php	164 KB
passrecover.php	164 KB

Table 4.c Number of Requests per Page

Page	No. of Requests
index.php	2
list.php	13
stats.php	13
select.php	8

As a comparison, Table 4.d contains the results from the same tests done on the Google Drive website.

Table 4.d Google Drive Testing

Page Loading Time	995 milliseconds
Page Size	1.8 MB

As we can see, the loading time is much higher than our own, but this is because our interface is simpler and uses just a few images and we use an insecure connection, HTTP instead of HTTPS as opposed to the Google website, thus not having to wait for an encryption to be done.

6.2. Multi User Testing

For multiple users testing we are using Testomato with 3 logged users at the same time for server file handling and database testing, and BlazeMeter with 500 simultaneously connected users for bulk testing. Details and results for each platform used are listed below with the results for more than one hour illustrated in Figure 36.a and the website statistics illustrated in Figure 36.b for the Testomato test and in Figure 37.a, Figure 37.b and Figure 37.c are the result for the BlazeMeter test.

- Testomato details
  - Automatically run every 5 minutes.
  - Uptime monitoring every 15 seconds.
  - Testing locations: Europe (Prague and Frankfurt), USA (Chicago, Dallas)
  - Timeout: 7 seconds. After this period, the tests are marked as failed.
  - Delay between subsequent tests: 50 milliseconds.
  - Browsers Used: TestTomatoBot and Mozilla
  - Users tested: 3 simultaneously.
  - Tested for 1+ hour.

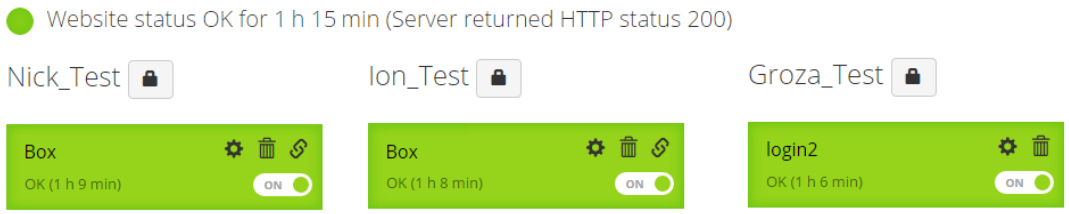


Figure 36.a Testomato Check

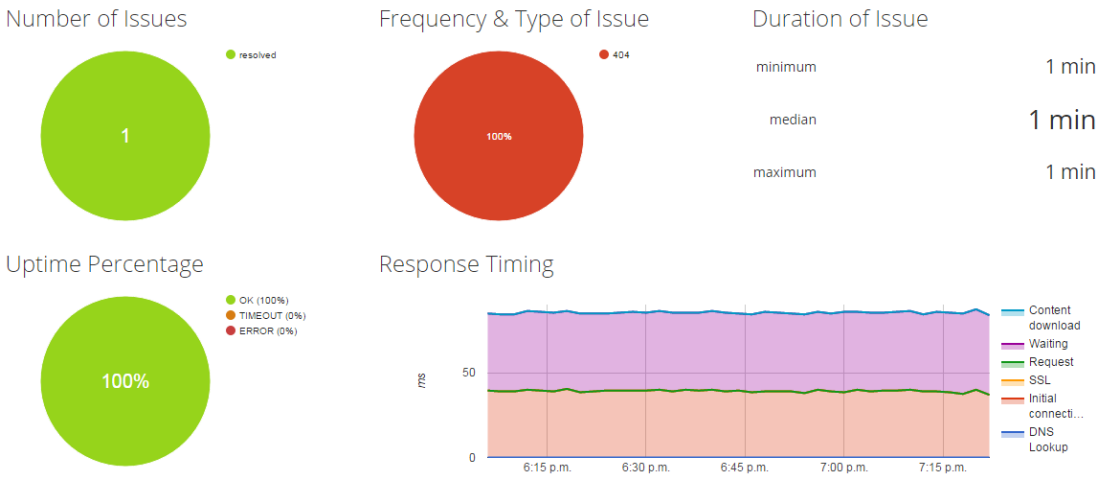


Figure 36.b Testomato Summary

BlazeMeter details:

- Threads per engine: 500
- No of iterations: ∞
- Ramp up: 300 seconds
- Test Duration: 15 minutes
- Delay between requests: 10 seconds
- RPS: 20

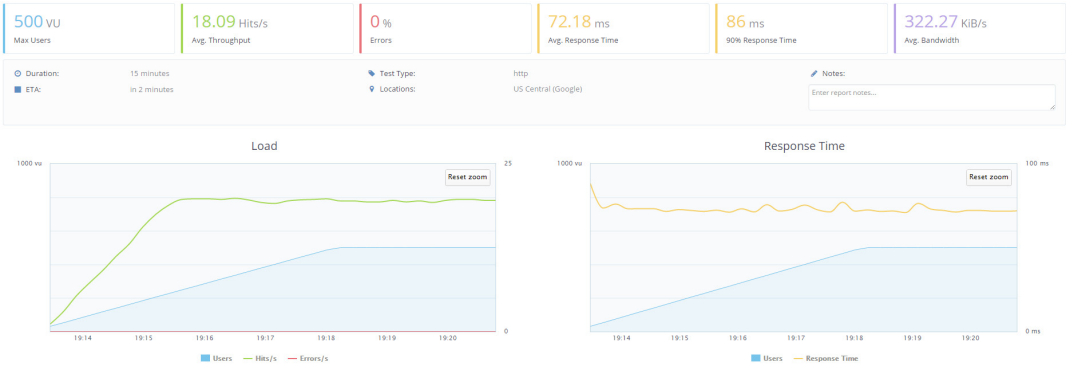


Figure 37.a Overall Statistics

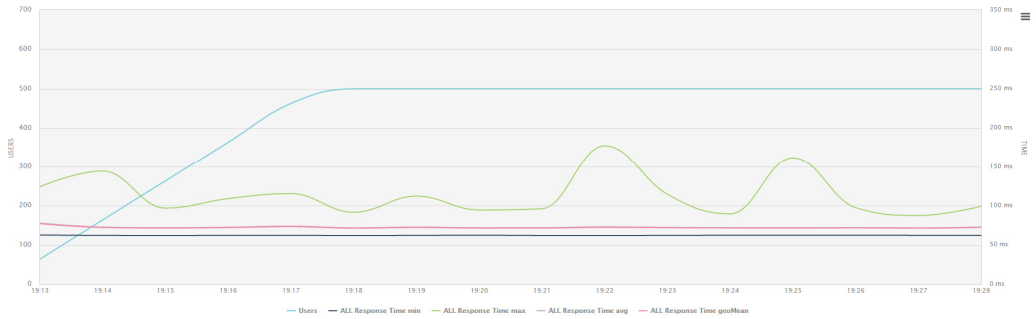


Figure 37.b Response Time

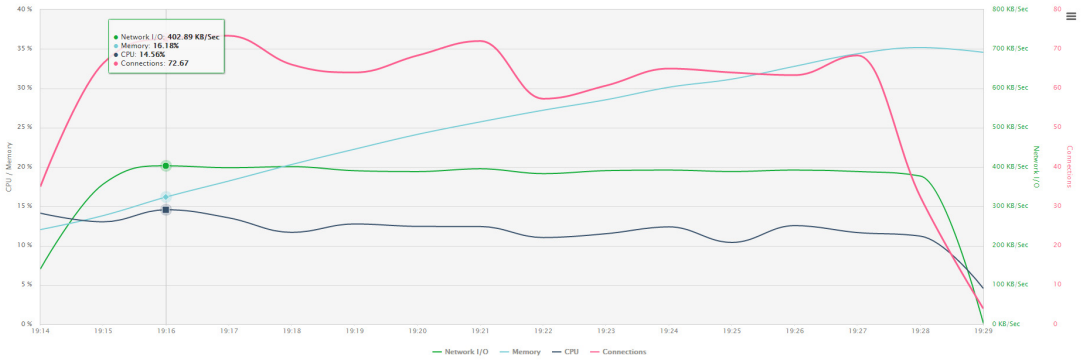


Figure 37.c Usage Statistics

7. Conclusion

7.1. Achievements

This project has been a great learning experience for me, both from the software and hardware point of view. While working on the project I've come over numerous problems and setbacks but managed to get past every single one of them. The biggest obstacle I found during the development process was to create a stable connection that will fail as rarely as possible.

7.1.1. Sensors

One of the most important part of the project are the sensors. When creating the project, I was able to understand how different types of sensors function such as the light sensors (photo resistor – changes its resistance based on light), temperature sensors (transistor that changes its properties based on the temperature) and so on. One last thing that I learned while working with different sensors was how different analog and digital sensors are and how different they work.

7.1.2. Communications Protocols and Methods

While creating the communication modules I learned a bit from each protocol and method such as network sockets such as the *"C10K"* problem (managing more than 10,000 connections at the same time); this, alongside other problems, came out when testing the server for a major number of simulated users at the same time. Other protocols used such as FTP, HTTP and SSH that I was already familiar with, I managed to deepen my knowledge on them.

Moving away from network communications and we are left with the serial communication of the Arduino with the Raspberry Pi and I2C (or I^2C) used for reading the digital sensors from the Raspberry Pi. By working with both of them, I managed to understand even better how serial communication works.

7.1.3. Threads and Tasks

While designing the desktop application, I run into blocks of code that will freeze the user interface or the whole program. Those blocks include waiting for a connection to be made, sending/receiving data to and from the server's socket. When writing the application in C#, threads and tasks have been the solution to overcome this issue, from simple tasks such as waiting for the user to login to more complex one such as periodically reading data from the socket.

7.2. Future work

While the project is a complex one, I still have a lot of ideas to improve it. All the circuits of the project are built using a breadboard and wires and is the first thing that needs to be modified. The breadboard and wires will be replaced with a custom-made PCB or soldered into a prototype board like in Figure 38.a and the display will use a board with a shift-register like the 74HC595 in order to use less wires as seen in Figure 38.b. The LCD comes on top of the board while the 5-pin connector (GND, +5v, 3 register pins) will connect to the PCB.

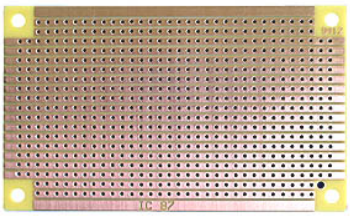


Figure 38.a Prototype Board

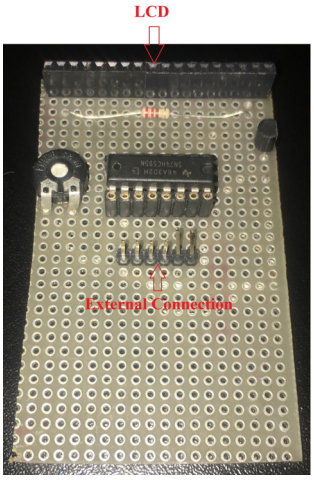


Figure 38.b LCD Multiplexing Board

After the circuit problem is resolved, I would implement the desktop application on the Mono platform in order for it to be run on the Linux OS that the servers run on or implement a similar way for the user to watch and control the box. The reason for doing this is to add an external touchscreen LCD such as in Figure 39. on the outside of the box or attached to the Raspberry itself.

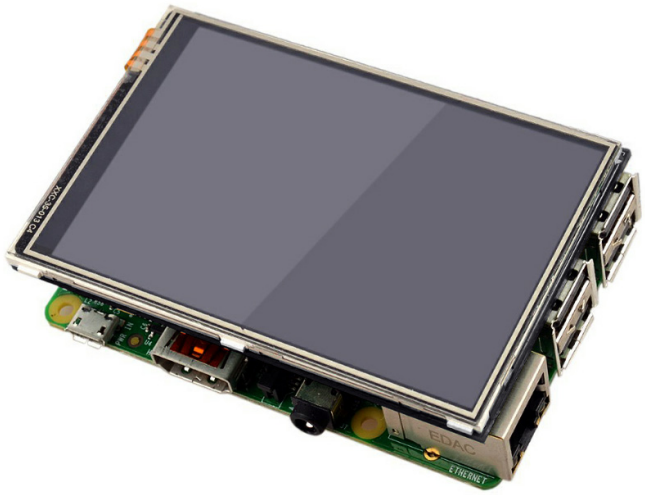


Figure 39. Raspberry Pi touchscreen LCD

The next update the system receives is to make all the features from the website and the application common between them. Functions from the website such as editing a user, starting the LCD or viewing the file charts will be added to the application while features from the application such as the graph and relay controls would be added to the website.

One thing that can be added is a webcam placed inside the box for monitoring various things, like the progress of plants or presence of bugs and rodents.

In the near future, the following specific updates are planned:

7.2.1. Website Specific Updates

- Data logs graph.
- Possibility to control the actuators.

7.2.2. Application Specific Updates

- Possibility to edit any user from the application (available for administrators only).
- LCD Control (available for administrators only).
- Charts for displaying file status.
- Password recovery.
- Responsive graph including zooming, auto-updating and panning.

These are all specific to the application the user is using, but I also plan to add general functionalities that will work on all devices such as personal information modification and sensors schedule.

All those mentioned before are planned to be done in the near future, but, I already started in adding new things to the system such as a scheduled program based on the calendar. This is being implemented, for now, in the desktop application and will look like in Figure 40. where the user will be able to choose between the automatic control that is currently deployed on the server, the manual mode using the overdrive function or via a calendar schedule. The user will be able to set the date and time at which the actuators will be activated.

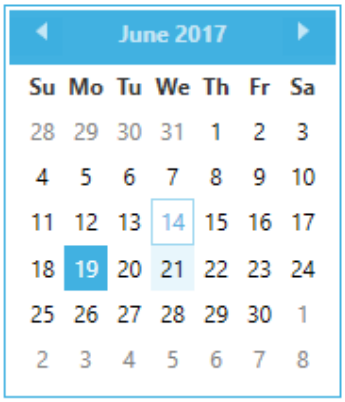


Figure 40. Calendar View

Another quality-of-life improvement over the current version would be to implement sensors for the actuators in order to make sure they work. An example would be a flow meter for the water pump; the water container would run out of water and will notify the user. As of now, this will only be possible via email or through the web/desktop application if they are open. I plan on porting the application to the mobile side, on iOS and Android and the user will receive a mobile notification from the installed application. After building the mobile app, notifications will become much easier.

References

[1] - Aziz, Izzatdin Abdul, Mohd Hilmi Hasan, Mohd Jimmy Ismail, Mazlina Mehat, and Nazleeni Samiha Haron. "Remote monitoring in agricultural greenhouse using wireless sensor and short message service (SMS)." International Journal of Engineering & Technology IJET 9, no. 9 (2009): 1-12.

[2] - Ahonen, Teemu, Reino Virrankoski, and Mohammed Elmusrati. "Greenhouse monitoring with wireless sensor network." In Mechtronic and Embedded Systems and Applications, 2008. MESA 2008. IEEE/ASME International Conference on, pp. 403-408. IEEE, 2008.

[3] - Call, Matthew, John A. Morrison, Lan D. Phan, and Mei-Man L. Syu. "System and method for improving a data redundancy scheme in a solid state subsystem with additional metadata." U.S. Patent 8,700,951, issued April 15, 2014.

[4] - Alexandre Denault, Analog Music in a Digital World, 2014

[5] - Doknić, Vesna. "Internet of Things Greenhouse Monitoring and Automation System." (2014)