School of Computer Science, University of Manchester

The Automated Greenhouse

Ryszard Rowinski
BSc (Hons) Computer Science WIE

Supervisor: Dr. Paul Nutter
May 2016

Abstract

The Automated Greenhouse

A greenhouse provides an environment to grow plants all year round, even on cold and cloudy days. However, extreme environmental factors inside the greenhouse such as high temperatures and a high humidity can negatively impact the plants. Consequently, controlling this environment is essential in order for the plants to grow strong and healthy.

The aim of this project was to design and build a greenhouse controller that can maintain the environment, by acting upon live sensor readings and be able to display the status of the system to the owner.

The project was split into two parts: programming a microcontroller to act as the central hub that manages the sensors and transducers; and creating a web application to allow the user to interact with the greenhouse controller.

This report goes thought the design and implementation of creating both of these parts as well as an evaluation of the final system including what could be done to improve it.

Author: Ryszard Rowinski

Supervisor: Dr. Paul Nutter

Date: May 2016

Acknowledgments

I would like to give special thanks to Paul Nutter for all his support and advice throughout the project.

I would also like to thank Daniel Underwood and Megan Hargreaves for sharing their ideas and continuous support.

Contents

List of Figures

List of Tables

1. Introduction

Greenhouses are a great way to make plants available all year round, however their effectiveness depends on the weather conditions which vary constantly. Although we are able to predict the weather to a high degree, the predictions are not always 100% accurate and so planning ahead would not help all of the time. Some of the problems that can occur are frost, condensation and overheating which can lead to the plants becoming damaged.

Automated systems that are used to control environments e.g. a fish tank that needs to maintain a specific water temperature, are able to cope with sudden, unpredicted changes. Being able to cope with the changing weather would reduce the number of damaged plants.

The purpose of this project is to address the fact that there are currently no automated systems specifically designed for controlling the everyday garden greenhouse.

This chapter highlights the aims and objectives of the project as well as what greenhouse automation systems exist in industry and what automation system are available in the home.

1.1. Home Automation

Home automation is becoming more and more prevalent and affordable. Some of the services currently available are: home theatre/entertainment, whole house audio control, lighting control, security and monitoring, thermostat control and more. Some examples of home automation systems in the market are: Hive [1], Control 4 Smart Home [2], Samsung SmartThings [3] and Apple HomeKit [4].

These systems are generally based around a hub which connects to specific compatible, controllable devices where users interface with it using a mobile device or a remote. Control is not limited to within the home either as most systems allow you to connect and change settings away from home.

1.2. Greenhouse Automation in Industry

Automated greenhouses do exist for large scale greenhouses. Companies such as GrowPonics [5] and Climate Control Systems Inc [6] provide services that maintain the environment as well as helping to reduce energy costs and waste. To do this, the systems need to monitor and manage a range of factors, such as: temperature, humidity, amount of light, wind speed and direction, CO2 levels, water flow, pH and the amount of fertiliser needed.

1.3. Aims and Objectives

The aim of this project is to create an automatic greenhouse controller that can monitor and control the temperature, humidity, light and soil moisture levels to user defined values. The controller would need some way for the readings to be displayed and for the user to update the settings.

There are four major objectives for the project:

1. Take Temperature, Humidity, Light and Soil Moisture readings
2. Display past and present sensor readings to the user
3. Be able to update the settings for multiple plants
4. Act upon sensor readings that deviate from the defined range

2. Background

This chapter talks about which environmental factors will be monitored by the automated system and discusses the choices of microcontrollers that were available to use.

2.1. Temperature

The temperature of the environment is extremely important for plants as it effects multiple growing factors: the rates of photosynthesis and respiration, germination, flowering and ultimately crop quality [44]. Extreme temperatures can negatively impact plant productivity [9] so maintaining the temperature in a greenhouse is crucial. Each plant also has its own specific temperature range so being able to adjust the settings in the greenhouse is equally as important.

2.2. Humidity

Humidity is the measure of how much moisture is present in the air. When plants transpire, water vapour along with molecules of gas are released from the statoma on the underside of the leaves, increasing the humidity [45].

A high humidity can be fatal to plants if it is not monitored, as a build-up of moisture on plants 'promotes the germination of fungal pathogen spores such as Botrytis and powdery mildew.' [10][1]. It is therefore important to make sure that air is circulating through the greenhouse to reduce the water vapour around the plants.

[1] Botrytis – '*Botrytis* at first appears as a white growth on the plant but very soon darkens to a gray color. Smoky-gray "dusty" spores form and are spread by the wind or in water. In greenhouses, any activity will result in a release of spores.' [11]

2.3. Light

Plants use the energy from light for photosynthesis and so without light, plants would not be able to grow. Different plant need different amounts of light but if a plant receives too little light then problems will start occurring, such as [46]:

- Stems will be leggy or stretched out
- Leaves turn yellow
- Leaves are too small
- Leave or stems are spindly
- Brown edges or tips on leaves
- Lower leaves dry up
- Variegated leaves lose their variegation

2.4. Soil Moisture

Water is another key element in photosynthesis. Plants absorb water through their roots where it then travels up the stem and into the leaves where photosynthesis occurs. It is therefore important to monitor the moisture levels in the soil, making sure that there is enough water for the plant but also taking care not to over saturate the soil, as this could cause the roots to rot.

2.5. The Choice of Microcontroller

The microcontroller is the key component of the project as it will control the sensors and transducers that monitor the environment. It will also need a way to connect to the internet and store sensor readings and plant settings.

Initial research into the possible devices needed for this project showed that the microcontroller would need around 7 analogue pins and 9 digital pins. This needed to be taken into account when selecting the board. However, these values were likely to change as the project progressed so the board needed to be able to cope with that. There were numerous microcontrollers available and Table 2.1 shows a comparison of five potential boards.

Boards / Features	Number of On board Analogue I/O Pins	Number of On board Digital I/O Pins	Internet Access (Ethernet)	Internet Access (Wi-Fi)	Can connect to monitor	Storage	Support
Arduino UNO [44]	6	14	With shield	With shield	No	Micro SD card slot with shield	High
Raspberry Pi 2 B [45]	0	40 (GPIO)	Integrated	WiPi	Integrated HDMI	Micro SD card slot	High
Beagle Bone Black [46]	7	65	Integrated	With dongle	Integrated micro HDMI	Micro SD card slot	Medium
Arduino YUN [12]	12	20	Integrated	Integrated	No	Micro SD card slot	High
Embedded Pi [47]	6	16 - 33	With shield	With shield	Yes – must connect to Pi first	Micro SD card slot with shield	Low

Table 2.1: Microcontroller Comparison

Out of these five boards the decision was made to use the Arduino Yun (Figure 2.1). The Yun was able to match up with the number of analogue and digital pins that the initial research found but there was also the ability expand if necessary. The Raspberry Pi 2 B and the Beagle Bone Black were also able to cope with the amount of pins needed, so these two were good contenders.

What helped make the final decision was the fact that the Arduino Yun had integrated Wi-Fi, which is important as the board is supposed to work out of the home and in the greenhouse. Furthermore there is a lot of support for the Arduino in general.

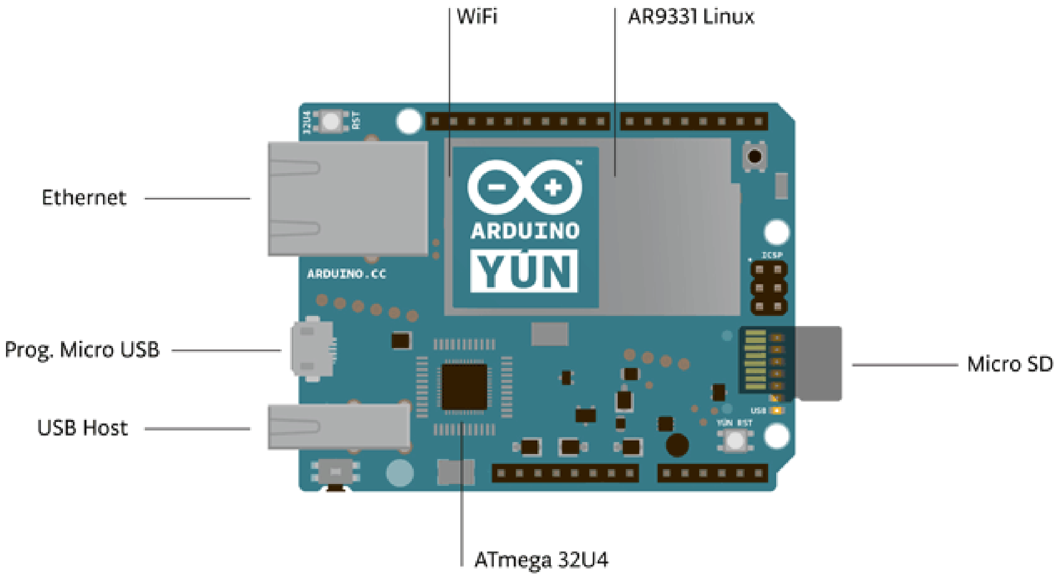


Figure 2.1: Arduino Yun

3. Design and Implementation

This chapter covers the design decisions and their implementation, including an overview of how the system links together and the setup needed before implementation could occur.

3.1. Dataflow

There are five main components for this project: a hub, sensors, transducers, LCD screen and a web application. As you can see from Figure 3.1, the Arduino Yun is responsible for everything and it is capable of doing this because this version of the Arduino has two processers on board: the ATmega32u4 [13] and the Atheros AR9331 [14].

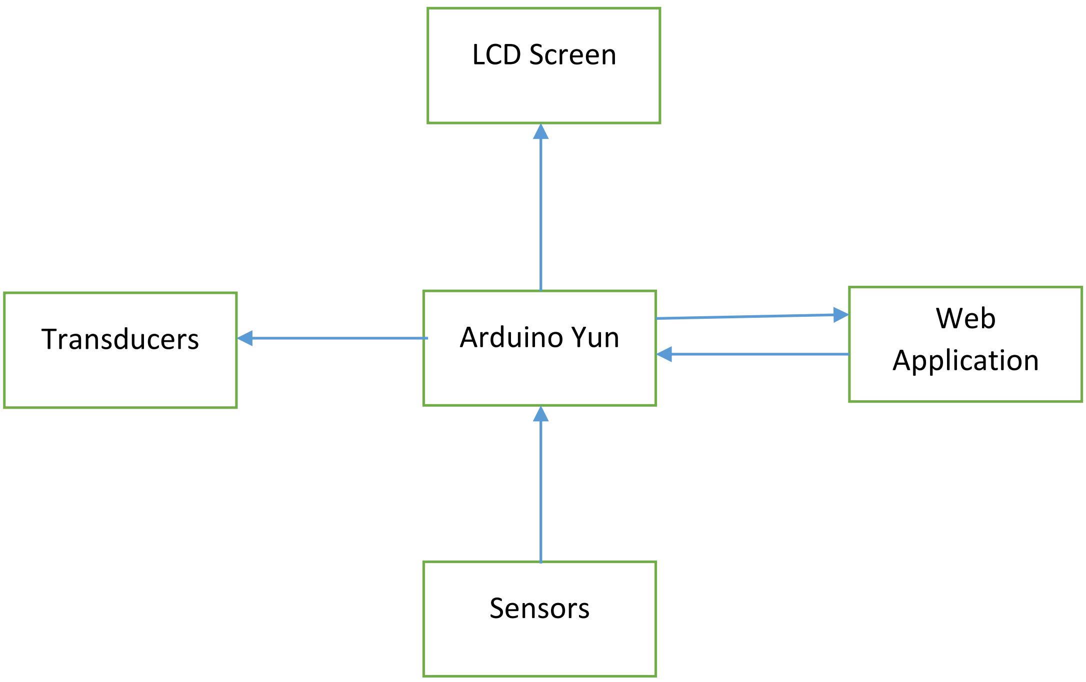


Figure 3.1: Data Flow

The ATmega is responsible for the Arduino environment i.e. controlling the analogue and digital pins, and the Atheros uses a Linux distribution, Linino [15], which allows the Arduino to host a webserver and makes it possible for a database to be maintained on board. The two processors are able to communicate via a Bridge library [17] (Figure 3.2) making it possible to compare current sensor readings against stored, user defined data.

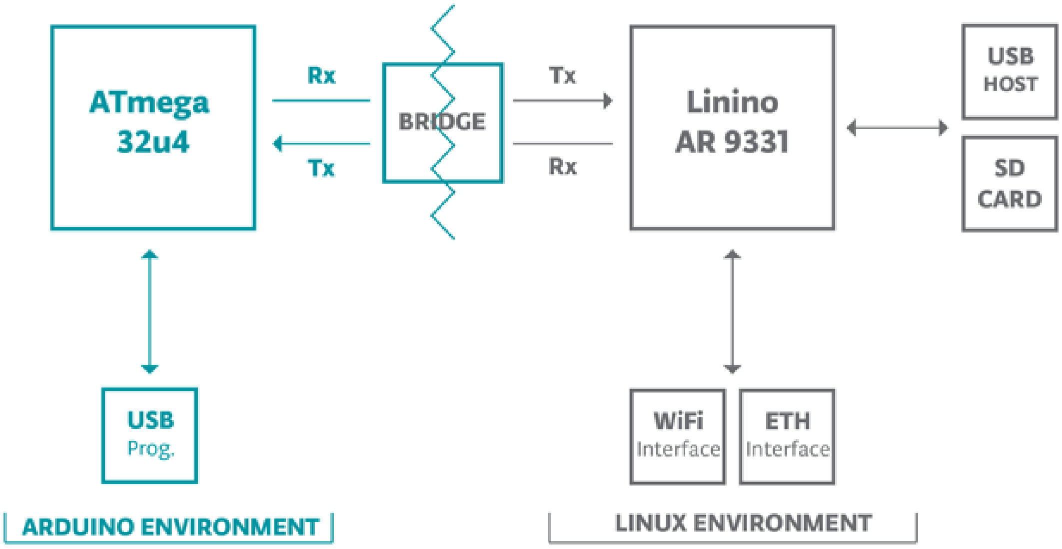


Figure 3.2: Arduino Bridge [12]

3.2. Setting Up

In order to start development on the greenhouse there were a few things that needed setting up first.

3.2.1. Environment

Working with an Arduino requires the use of the Arduino IDE [18] which allows you to write and upload your sketch to the board. For the Yun, the code can be uploaded via Wi-Fi so the next step was to connect the board to the PC.

The Yun has its own Wi-Fi available but once connected, the Yun can be configured to connect to any other Wi-Fi network. This can be achieved by navigating to the Arduino's local IP address 192.168.240.1 and logging in with the Arduino's password. In addition to configuring the Wi-Fi it is possible to update the OpenWrt [19] Linux distribution here.

Accessing the Linux file system was possible simply via SSH; either connecting to the board's current IP address.

If the Yun is connected to an external Wi-Fi network, such as a home network, then it is possible to connect to the website away from home, simply by connecting to the router's external IP.

3.2.2. Database

The Arduino Yun doesn't come with a database pre-installed so this was the first major task that needed to be completed, as it is crucial to be able to compare live readings against user-defined data for the system to be fully automated and to meet the objectives.

The first step in doing this was to expand the amount of storage space from the 16MB available: 9MB of which is already taken up by the operating system. This just ensures that there is enough space to save the readings. Completing this was quite straight forward as the Arduino is capable of utilizing a micro SD card and there is a sketch available to download from the Arduino website that formats the card into the new Linux file system [16].

Once the storage had been upgraded it was possible to install MySQL, PHP and a few necessary sub-modules using the inbuilt package manager OPKG [20].

3.2.3. Website Folder Structure

To have an accessible website two folders needed to be created in the SD root: one called 'arduino' with a subfolder called 'www'. The Yun then created a shortcut to this, called 'sd', within the main 'www' folder so accessing my website was now possible by navigating to: http://<current.ip.address>/sd e.g. 'http://192.168.240.1/sd'.

3.3. Arduino Sketch

The sketch is the main component of the automated greenhouse as this is what controls the sensors and transducers.

3.3.1. Design

The controller was required to monitor the temperature, humidity, light and soil moisture levels as these are four of the most important environmental factors during plant growth. The design step for this was to think about what can be done to control these factors. It was also necessary to think about how the web application would interact with the sketch so that it could display the current sensor values.

3.3.1.1. Dealing with Temperature

All plants have a temperature range that they grow best in, which is why we see different plants at different times of the year. An automated greenhouse should be able to maintain a specific temperature range for however long is necessary.

If the temperature exceeds the maximum allowed temperature, the greenhouse needs cooling down in some way. In industry this would usually be achieved by opening vents or turning on fans, letting air circulate through the greenhouse. When the temperature becomes too low, the greenhouse would need to be heated up and all vents and fans would be closed and powered down.

For the project, two CPU fans can be used represent an intake fan and an extractor fan and can be turned on when the temperature is too high. If the temperature becomes too low an LED will turn on as it is not feasible to connect a heater for safety reasons.

3.3.1.2. Dealing with Humidity

The most effective way to maintain a low humidity is to keep the greenhouse dry, which can be done by warming up plants and keeping air circulating through it. If the humidity is too low however, then moisture needs to be added to the air. This can be achieved by spraying water around the plants.

The CPU fans mentioned in 3.4.1.1 ‘Dealing with Temperature’ can be used to reduce the humidity if it becomes too high. If the humidity becomes too low another LED will be used to indicate it instead of spraying water, as using water around the board could damage it.

3.3.1.3. Dealing with Light

Lighting is generally only an issue at night but is easily rectified by simply turning on one or multiple lights, depending on the greenhouse size. It also needs remembering that ‘plants cannot get too much light, but they can get too much of the heat energy that comes with the light’ [21], so care needs to be taken with the type of bulbs being used and what the current temperature of the greenhouse is.

A larger white LED will turn on during periods of low light to represent this.

3.3.1.4. Dealing with Soil Moisture

Plants absorb water through their roots for use in photosynthesis. This, along with other factors, such as a high temperature, will dry out the soil. Maintaining the soil moisture content is therefore essential in order for plants to continue to grow healthily and can be achieved in different ways. One of which would be to sprinkle water on the soil after a certain time period.

There are soil moisture sensors available for the Arduino so monitoring the soil moisture content can be done using one of these. Indicating when the levels become too high or too low will again be done using LEDs. There are pumps available for the Arduino, and examples of this being done [22], but was no time to carry this out in this project.

3.3.1.5. Communicating with the Web Application

OpenWrt-Yun uses REST for its clients and servers and has three endpoints [40]:

- /arduino
- /data
- /mailbox

The endpoint needed for this project is '/arduino'. There is nothing pre-configured for this directory, so anything added to the URL after the endpoint is passed directly to the sketch to be parsed by an API.

The web application needs to be able to:

- Retrieve the current readings
- Inform the board when the current plant data changes
- Overwrite the light (LED representing the light)

3.3.2. Implementation

All the code for the Arduino sketch is written in the Arduino IDE and uses its own Arduino language [34]. The sketch itself is composed of two parts: the setup() and the loop(). The setup() function is run once when the board is reset or powered up but the loop() is repeated constantly and is where all the decision making happens.

3.3.2.1. Setup()

Code that only needed running once was written in setup(). These are the areas covered in the project:

1. Starting up the Bridge; to enable communication between the two processors.
2. Starting up the Console [35]. The Console allows the Arduino Yun to read data coming from the bridge and is a substitute for Serial [38]. These are only viewable on the PC and are mainly for debugging purposes.
3. Setting the pin mode. All pins being used should be run with pinMode(pin,mode) to indicate whether it will be used as an input or an output.
4. Starting the YunServer [37]. This allows for server based calls to be made to the Yun e.g. sending sensor readings to the web application.
5. Selecting the current plant settings form the database. This needs to be run once at the start so that the values can be used as the upper and lower bounds when monitoring the sensor readings. The next time this is run is when user changes which plant needs monitoring through the web application.

3.3.2.2. Loop()

All other functionality that does not fall under setup occurs here. To keep the code tidy, different procedures were created for each piece of functionality. These procedures were then called within the *loop* and can be split up into six areas:

1. Take all the sensor readings and convert the values
2. Average out the readings using a moving window approach.
3. Compare the average readings against the plant settings.
4. Save the readings if the time limit has passed.
5. Print the readings and settings to the console.
6. Deal with API calls.

3.3.2.2.1. Sensor Readings

Temperature and Humidity

The temperature was monitored in degrees C and two different sensors were available: the TMP36 [8] (Figure 3.3), and the RHT03 [7], also known as DHT22 (Figure 3.4).

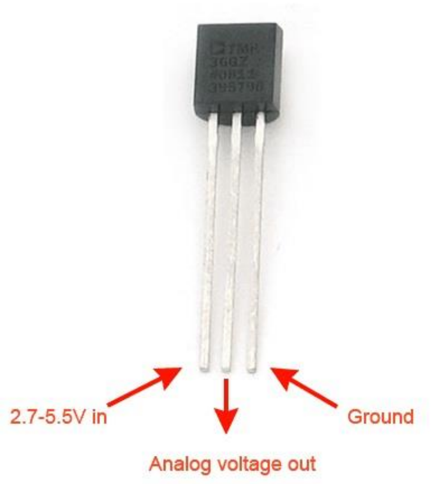


Figure 3.3: TMP36

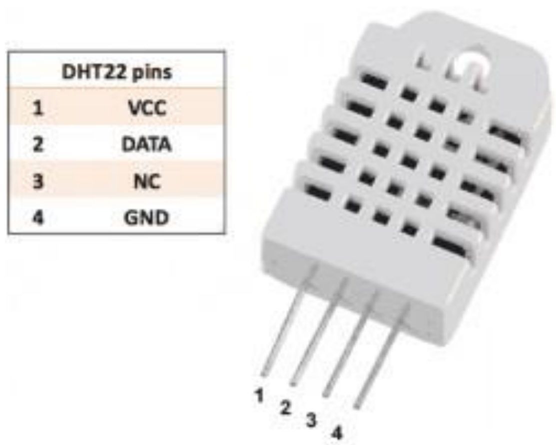


Figure 3.4: RHT03 / DHT22

The TMP36 is an analogue device that produces a voltage output that is linearly proportional to the temperature in Celsius, as seen in Figure 3.7. To find sensor's temperature reading we first need to calculate the voltage output by converting the analogue value given. The analogue values range from 0 – 1023 and are retrieved by using the `analogRead()` function.

```
int sensorValue = analogRead(A0);
float voltage = (sensorValue) * 5.0 / 1024.0;
```

Figure 3.5: Converting analogue reading into a voltage

In Figure 3.5, *A0* represents the first of the twelve[2] analogue pins available on the Yun. In the second step the sensor value is multiplied by *5.0* as this is the voltage being supplied to the sensor from the board. This could instead be *3.3*.

The final step is to convert the voltage into the temperature. Shown in Figure 3.6.

```
float temperature = (voltage - 0.5) * 100.0;
```

Figure 3.6: Converting Voltage to Degrees C

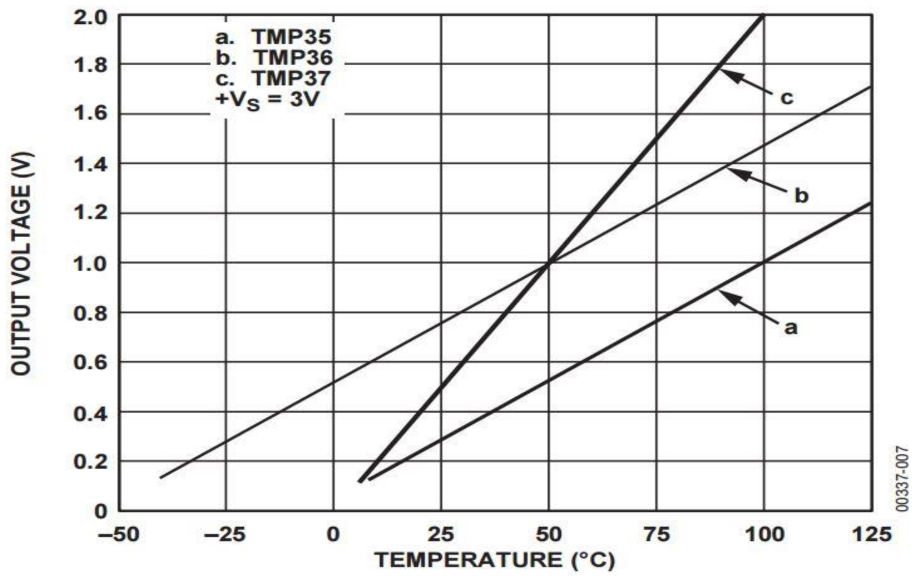


Figure 6. Output Voltage vs. Temperature

Figure 3.7: Output Voltage vs. Temperature for the TMP36 [12]

[2] A0 – A5 are in the same position as on the standard Arduino UNO, but A6 – A11 are actually the digital pins 4, 6, 8, 9, 10, and 12. All these pins can be used as digital pins.

The RHT03 on the other hand is a digital sensor which returns both temperature, in degrees C, and humidity readings to the board without the need for any conversions. Although it does require a copy of the DHT library [23] to use it.

In the end, the RHT03 was used as the temperature sensor as it gave more consistent readings and it was also the sensor being used to measuring the humidity.

Light

Measuring the light levels was carried out using an LDR [24] (Figure 3.9). This is not particularly good for accurate measurements but it is enough to indicate when it becomes dark. Like the TMP36 this is an analogue device so the output needs to be converted into something more understandable, and that was lux.

Lux [25] is one of the few ways to measure light. It is the unit of illumination equal to one lumen per square meter and can be calculated from the output voltage as shown in Figure 3.8.

```
lux = 500.0 / (10.0 * ((5.0 - voltage) / voltage));
```

Figure 3.8: Converting Voltage to Lux

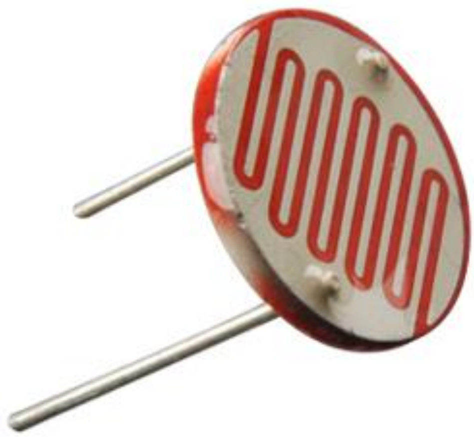


Figure 3.9: LDR

Soil Moisture

The HL–69 Soil Moisture sensor [26] was used for the soil moisture measurements (Figure 3.11). This was another analogue sensor so the values returned were between 0 – 1023. When the soil was dry this sensor gave a high output value and when the soil was wet it gave a low output value. This needed to be converted into a percentage as this is more intuitive to understand. The calculation is shown in Figure 3.10.

```
100 - (100.00 / 1024.00) * currentSoilMoistureValue
```

Figure 3.10: Calculating Soil Moisture Percentage

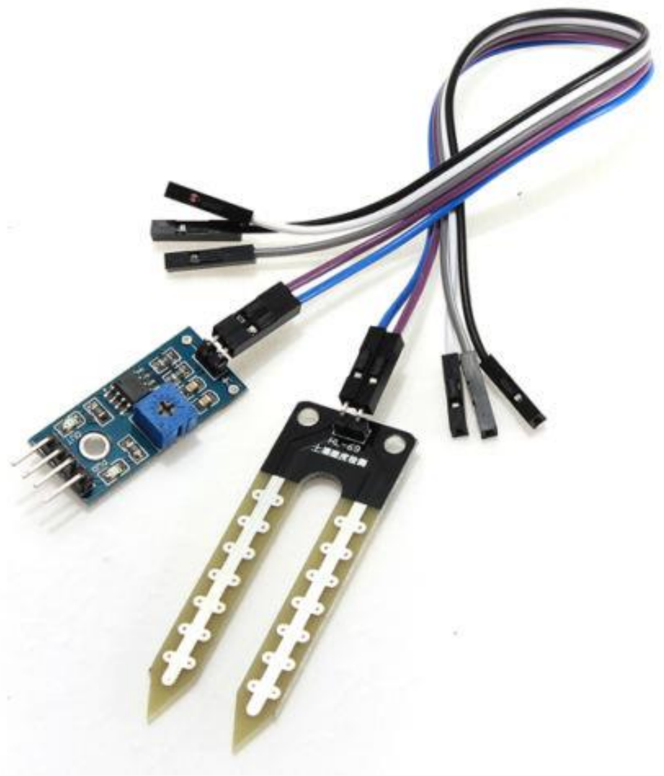


Figure 3.11: HL-69 Soil Moisture Sensor

3.3.2.2.2. Moving Window

To make the sensor readings more reliable it would be better to use an average across multiple readings instead of sticking with the raw value. One method for this is a moving window.

A moving window has a defined size and follows the latest value. In the project the moving window size was 30, which meant the previous 30 readings were stored and averaged. When the new sensor readings were made, they replaced the oldest value in the window.

Another approach was to take five readings for each reading every loop and find the average of them. This however was too slow as the RHT03 requires a 2 second delay before a request can be made again. This meant there was an extra 8 second delay before anything else could be executed. This in turn impacted the web application’s calls to the API as they were happening every 10 seconds and so started to build up.

3.3.2.2.3. Transducers

Taking sensor readings is the first step for an automated system, the second step is using these values to control the environment.

Intake and Extractor fans

Two CPU fans were connected to reduce the temperature and humidity (Figure 3.13). When either factor reaches their maximum allowed value the fans will turn on until the readings have reduced back to the acceptable range. Whether the fan is used to take in or extract air simply depends on which direction it is facing, there is no difference in the way it is controlled. The intake fan is used to bring in the cooler, fresh air whereas the extractor fan takes out the hot, humid air. These would be placed at opposite ends in an actual greenhouse so that the air can flow all the way through it. Turning the fan on is achievable by using the `analogWrite()` function, shown in Figure 3.12 .

```
void toggleIntakeFan(){
  // Compare max temperature and humidity readings
  if (currentTemperature >= plantSettings[1] ||
      currentHumidity >= plantSettings[3]) {
    analogWrite(INTAKEFAN, 255);
    Console.println("Intake fan ON");
  }
  else {
    analogWrite(INTAKEFAN, 0);
    Console.println("Intake fan OFF");
  }
}
```

Figure 3.12: toggleIntakeFan() function. Shows an example usage of the analogWrite() function.

Figure 3.13: 3 pin CPU fan. One used for intake and another for extraction

The fans used in the project have three pins: GND (Ground), V_{in} (Voltage In) and Fan Tach (or RPM output), but to make the fans spin it is only necessary to connect the GND and V_{in} pins. Unlike the other components used, the fans need an input voltage of 12V to run. The Arduino however can only provide a maximum of 5V so a transistor was needed to up the voltage.

A transistor acts like an electronic switch; they turn on and off using a small voltage and once it is on it will allow you to power bigger circuits. There are two types of transistors: NPN and PNP, each having three legs: Collector, Base and Emitter, sometimes referred to as Gate (Base), Drain (Collector) and Source (Emitter). The collector collects the current being controlled, the emitter releases the current and the base controls the collector and the emitter.

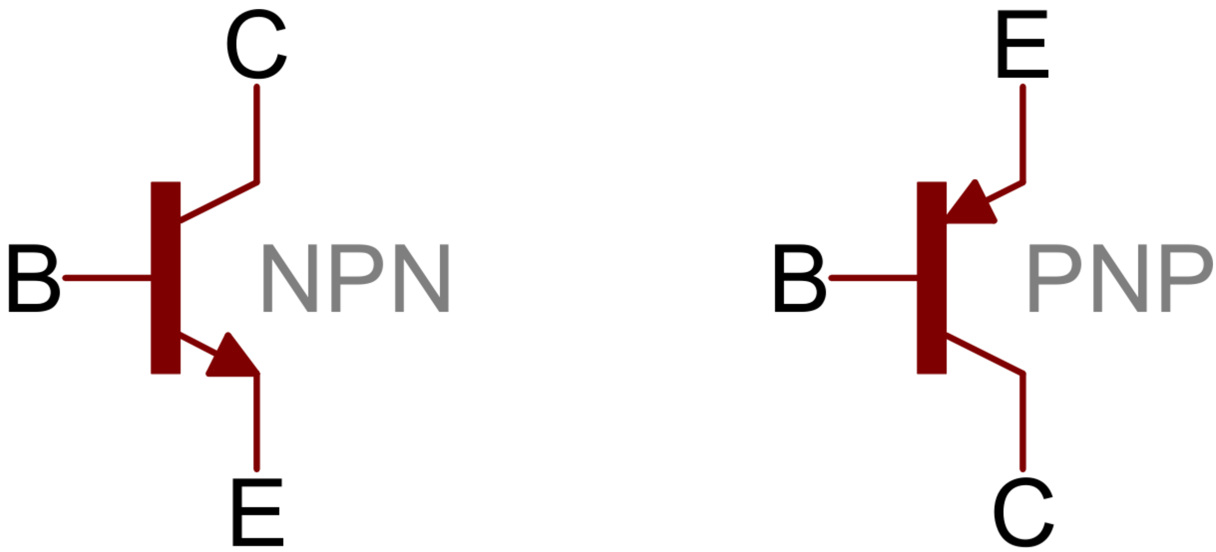


Figure 3.14: NPN vs. PNP circuit symbols [29]

The difference between NPN and PNP transistors is the direction that the current is intended to flow [27]. For an NPN transistor current flows out from the base and in to the collector[3] but with a PNP transistor the current flows out of the collector and into the base (Figure 3.14). There are various models of transistors but the one used in the project was the P2N2222A [28], an NPN transistor.

Another necessary component was a Diode [29]. The diode prevents a reverse current from flowing through the Arduino, which could potentially damage it. A reverse current could occur because of the momentum of the fan as it slows down, or because the fan could be turned [30].

[3] A mnemonic for NPN is 'Not Pointing iN'.

LEDs

Five LEDs of three different colours were used in this project:

1. White [31]: to simulate a light turning on when the LDR reading became too low
2. Red [32]: to indicate a low temperature, as a heater could not be used
3. Red: to indicate a low humidity, instead of a sprinkler system
4. Red: to indicate a low soil moisture level, instead of a sprinkler system
5. Green [33]: to indicate a high soil moisture level / to show when enough water has been added

The LEDs have two legs (see Figure 3.16), the positive anode and the negative cathode. The anode is longer in length and needs connecting to either a digital or analogue pin. The cathode is connected to GND. Whenever the boundary is passed, a signal is sent to the anode pin using `digitalWrite()`, turning on the LED (see Figure 3.15).

```
// Check min temperature
if (currentTemperature < plantSettings[0]) {
  digitalWrite(MIN_TEMP_LED, HIGH);
}
else {
  digitalWrite(MIN_TEMP_LED, LOW);
}
```

Figure 3.15: Example of digitalWrite() usage.

Figure 3.16: Example of one of the five LEDs used

3.3.2.2.4. Saving the Readings

One of the objectives included being able to see past sensor readings so in order to do this the sensor values needed to be saved to the database. Saving the readings every loop cycle would have been very inefficient as this would fill up the database very quickly, but it would also have a negative impact on the performance of the web application. This is because it takes time to save the data, so any API calls would have to wait to be dealt with.

Readings needed to be saved at least every hour so that an hourly graph could be plotted on the web application. However, to make the graph more reliable it would be better to plot an average, so readings were saved every 15 minutes (see Figure 3.17).

```
if((long)(millis() - rolltime) >= 0) {
   saveReadings();
   rolltime += FIFTEENMINS;
  }
```

Figure 3.17: If test to see if 15 minutes have passed

The `millis()` function returns the number of millisecond since the sketch started running and `FIFTEENMINS` was a constant created to represent 15 minutes in milliseconds.

```
void saveReadings(){
  Process saveReadings;
  String saveCommand = "python
   /mnt/sda1/arduino/www/Python/saveReadings.py " +
   (String)currentTemperature + " " +
   (String)currentHumidity + " " +
   (String)currentLux + " " +
   (String)currentSoilMoisturePercentage;
  saveReadings.runShellCommand(saveCommand);
}
```

Figure 3.18: saveReadings() function

To actually save the readings requires running a shell command, which is possible because of the Bridge library. The library includes a `Process` class that contains a function called: `runShellCommand()`. This takes a string as its argument, so a command to run a python script was created which takes all the sensor readings as parameters and saves them to the database (see Figure 3.18).

3.3.2.2.5. Outputting Sensor Readings and Plant Settings

To keep track of the sensor readings, they were all outputted to the Console every loop cycle. As well as this, a comment was outputted giving the state of the white LED and the fans. An example of this is shown in Figure 3.19.

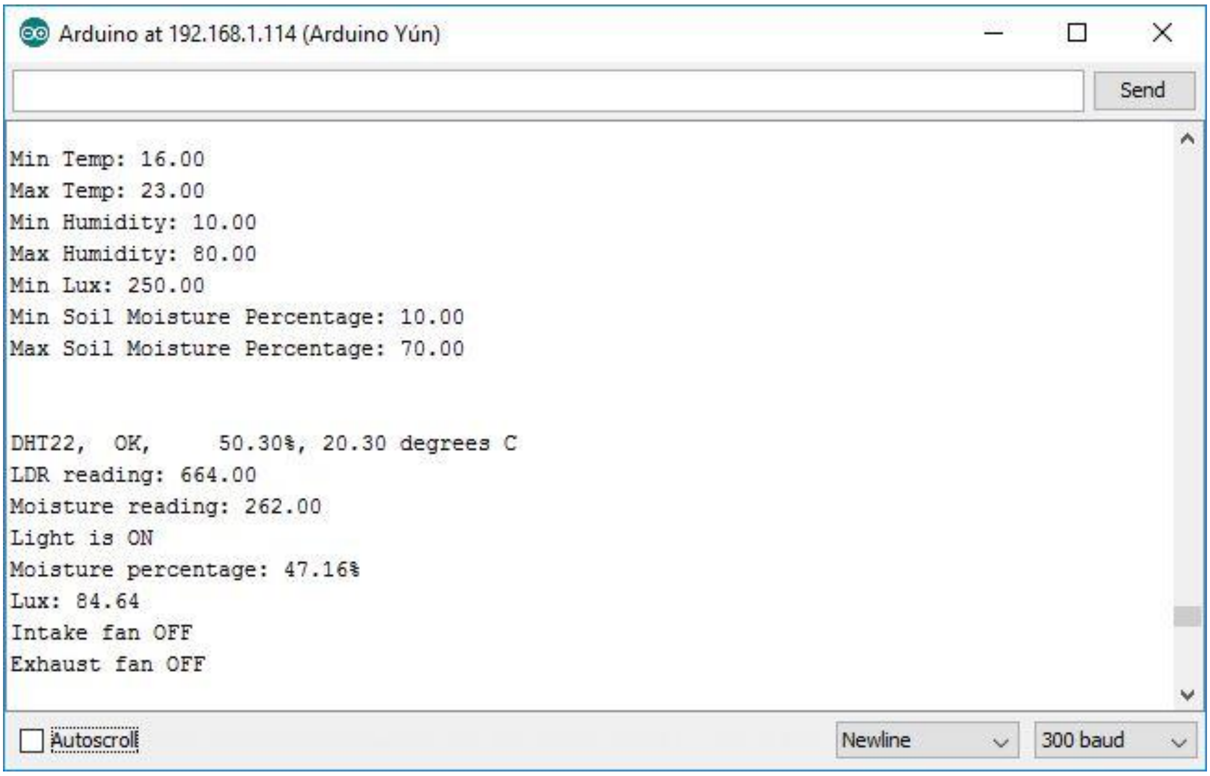


Figure 3.19: Console output

One of the requirements was to display the sensor readings to the user. The Console only works through the IDE so one way to do this was to use a 16x2 LCD screen [39] (Figure 3.20) that was connected to the Arduino. The screen uses a potentiometer [40] to adjust the contrast to make the text readable. The potentiometer allowed the resistance to be adjusted, making the screen become lighter or darker depending on which way it was turned.

Figure 3.20: LCD displaying all four sensor readings

3.3.2.2.6. API

The API works by firstly listening out for a client on the server. If a client is connected, the command that was sent is parsed to see what was asked.

The web application's home page uses the API the most as it displays the current readings, requesting for the new values every 10 seconds. Initially each reading was requested individually (Figure 3.21) but this was a slow process as only one command could be dealt with during each cycle of the loop, so instead all the readings are returned from one command (Figure 3.22).

```
if (command == "DHTTemp") {
  client.print(currentTemperature);
}

else if (command == "DHTHumidity") {
  client.print(currentHumidity);
}

else if (command == "lux") {
  client.print(currentLux);
}

else if (command == "moisture") {
  client.print(currentSoilMoisturePercentage);
}

else if (command == "isLightOn") {
  client.println(checkLight());
}
```

Figure 3.21: Return readings using individual API calls

```
if (command == "getReadings") {
  client.println(currentTemperature);
  client.println(currentHumidity);
  client.println(currentLux);
  client.println(currentSoilMoisturePercentage);
  client.println(checkLight());
}
```

Figure 3.22: Return readings using one API call

3.4. Web Application

Even though the greenhouse is automated, the user needs a way to monitor the sensor readings and set the maximum and minimum values allowed for each environmental factor. Most home automation systems have an app that allows the user to do just this.

3.4.1. Design

The web application was split into two parts: a UI and a database behind it.

3.4.1.1. Database

The database needed the ability to record each of the sensor readings with the time they were taken. It also needed the ability to store plant settings; these being:

- Plant name (to aid the user)
- Minimum temperature
- Maximum temperature
- Minimum humidity
- Maximum humidity
- Minimum lux
- Minimum soil moisture
- Maximum soil moisture
- Is this the active plant?

3.4.1.2. UI

The application had to be able to display the past and present sensor readings; allow the user to pick which plant settings the greenhouse should adhere to; and be able to add, update and delete plant settings. The application also needs to be usable on a variety devices: PC, mobile, tablet etc.

3.4.2. Implementation

The implementation for the web application was carried out at the same time as the implementation of the Arduino sketch because they both depend on each other. The web applications needs to know the past and present sensor readings and the sketch needs to know which settings were active or if the user requested to turn the light off.

3.4.2.1. Database

The database consisted of just five tables: temperature, humidity, light, soil_moisture and plant_data. These could hold the data mentioned in 3.4.1.1. To retrieve and save the data the web application used AJAX to call a PHP page which executed the relevant SQL code. If there was a request for data then a JSON object was then returned containing the result.

3.4.2.2. UI

The UI consisted of six pages: the dashboard, a page for each reading and one for editing the plant settings.

Home

The home page displays all four of the averaged sensor readings at the top of the page. They each have a coloured background which informs the user whereabouts in the given range the current reading lies. If the background is blue, the reading is too low; if the background is green, the reading is acceptable; but if the background is red, then the reading is too high. The range for each factor is also shown just below readings (see Figure 3.23).

Below the current readings is a graph displaying the average hourly readings for each of the sensor reading (see figure 3.23). All the data is plotted on one graph but each factor can be toggled on or off. There is also the option to change the date to see the graph for a previous day. The graph was produced using Chart.js [48] for its simple yet stylish appearance, its responsiveness, its support for multiple browsers and its ability to work on a variety of devices. A few other possible options were: chartist [49], highcharts [50] and zingChart [51].

Underneath this is the override section. This currently only overrides the light for 60 loop cycles (Figure 3.24)

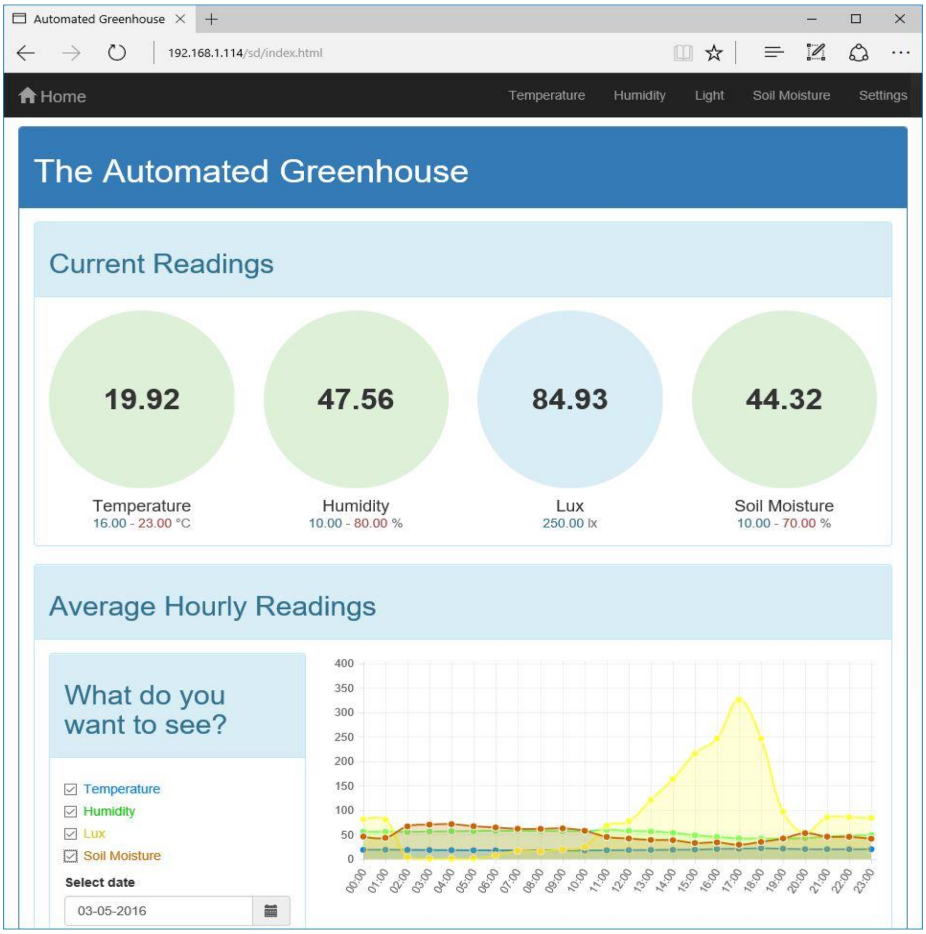


Figure 3.23: Web Application home page showing current readings and a graph of the average hourly readings for a chosen day

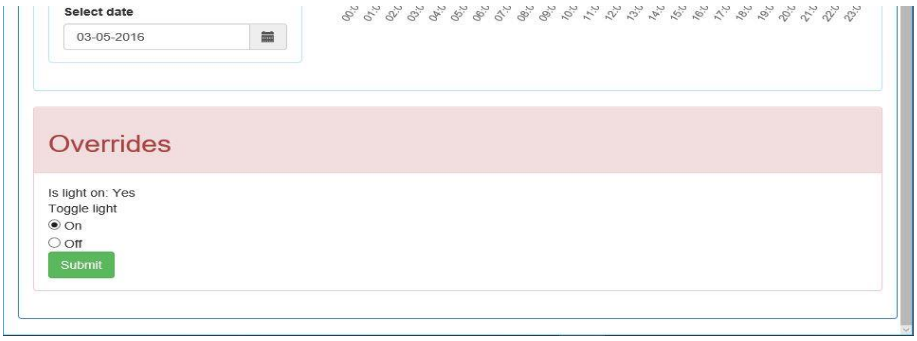


Figure 3.24: Web Application Overrides section on home page

Detailed results sections

There are four other areas to the application, one for each environmental factor. These pages all display an average hourly graph for the selected day like on the home page but just for the relevant factor. Furthermore there is a table displaying all of the saved readings for that day. Figure 3.25 shows the Temperature page as an example.

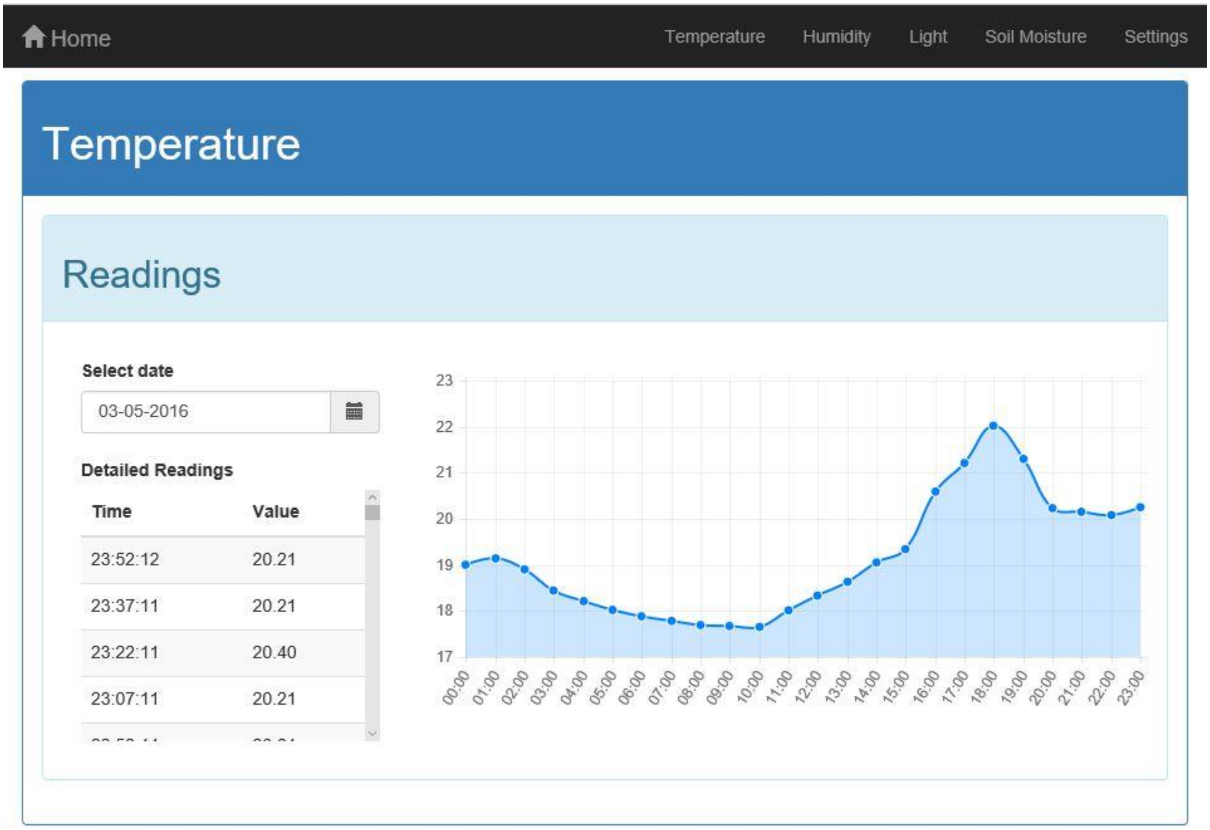


Figure 3.25: Screenshot of the Temperature page

Settings

The final page is the settings page. This is where the user can add plant information and choose which settings should be active and used in the greenhouse. The page was split into three sections: 'Current plant', 'All plant data' and 'Edit plant data' (see Figure 3.26 and Figure 3.27).

The ‘Current plant’ section displays the settings for the currently active plant; the ‘All plant data’ section display all the saved plant settings that the user has entered; and the ‘Edit plant data’ section allows the user to either add settings for a new plant or edit the settings of an existing plant. The system does not allow for multiple settings to have the same name.

Styling

To make the application portable to different devices, the styling was done using Bootstrap [41]. The application makes use of the ‘Panel’ component to group similar information together and uses the default Bootstrap colour scheme throughout.

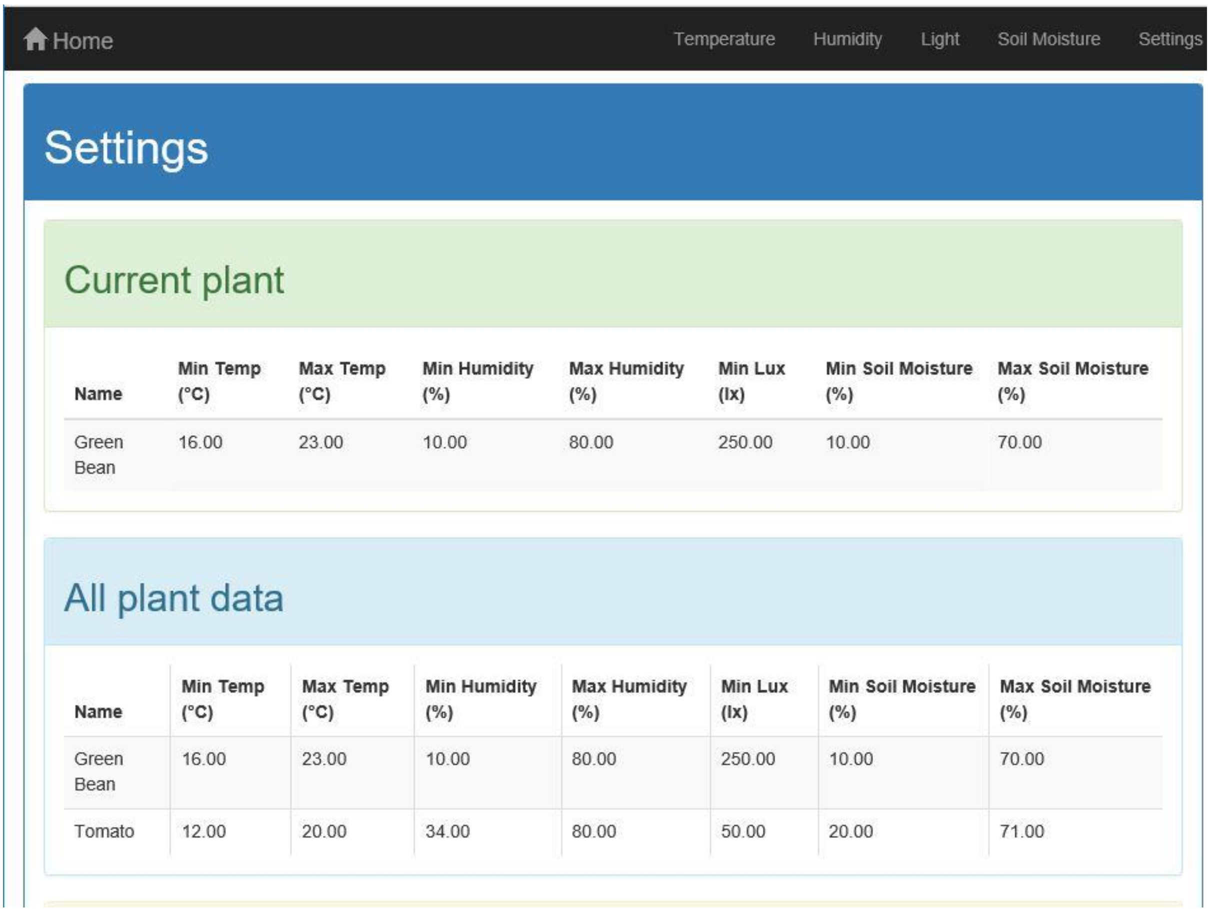


Figure 3.26: Screenshot of Settings page, showingthe Current Plant and All Plant Data sections

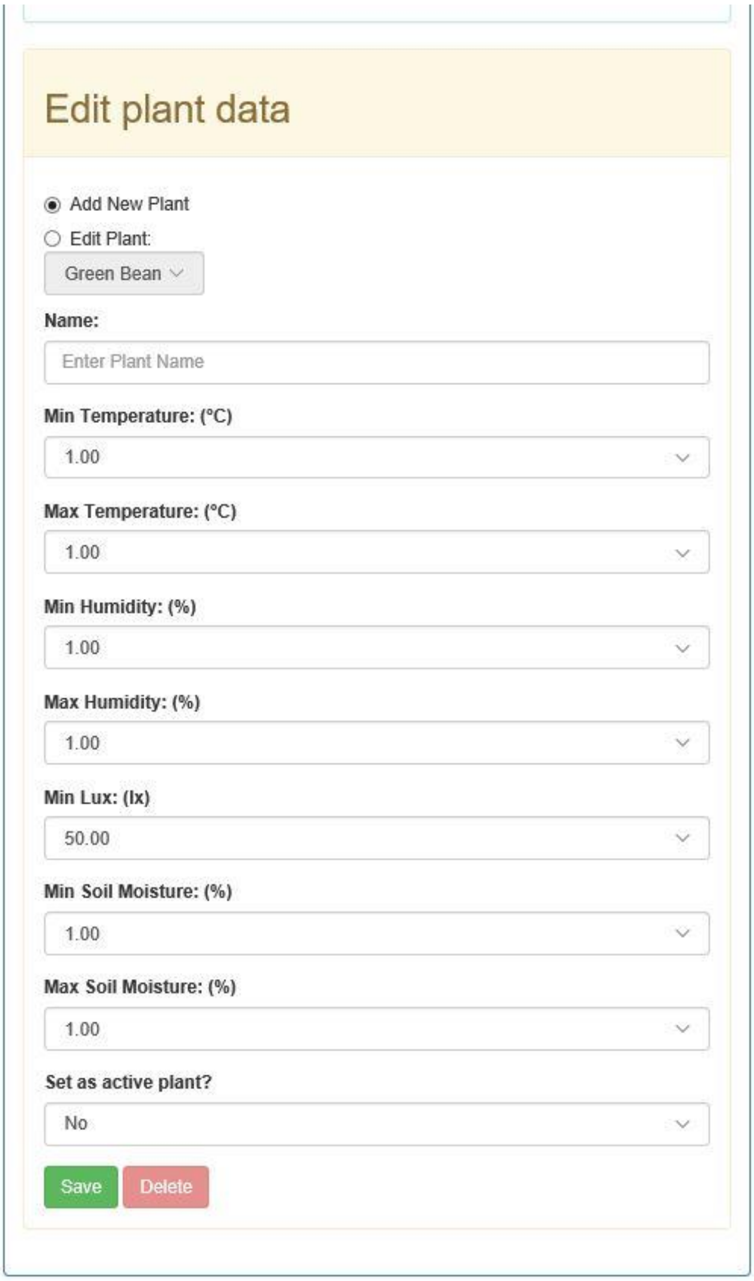


Figure 3.27: Screenshot of the Settings page showing the 'Edit plant data' section

4. Testing and Evaluation

This chapter discusses the techniques used to test and evaluate the system.

4.1. Testing the Hardware

There were two areas to look at when testing the hardware. The first area dealt with the sensor readings: do the sensors actually return a value and did these values change correctly when the environment changed. The second area dealt with the transducers; did the fans and lights turn on or off when the readings were out of the acceptable ranges.

4.1.1. Sensors

The first area was simple to test. After the sensors had been connected to the Arduino, code was written to read the output from the pin that was used then the result was outputted to the Console.

Testing for temperature and humidity changes was completed by breathing hot air around the sensor. When this was done, both readings rose suddenly then gradually began to fall as expected.

The LDR was tested by shining a light onto it. As the light came closer to the LDR the readings increased and then reduced once taken away as expected.

For the moisture sensor, when water was added to the soil the percentage rose then slowly decreased as the day went on.

4.1.2. Transducers

Testing the transducers started off by sending a high signal to the device to make sure they worked. After that, hardcoded values were used as the environmental boundaries. At this stage the sensor tests were repeated to see if the transducers were working based on the environment changing. The hardcoded values were also changed to test the boundaries. The fans and LEDs turned on as expected.

Once this was completed and the database was set up with a record in the plant_data table, the values were retrieved using a Python script and testing was repeated.

4.2. Testing the Web Application

Testing the web application was more complex than testing the sketch as there is a lot more going on. Each subsection gives a list of all the tests that were carried out in order for the pages to be successful.

4.2.1. Home Page

Test	**Expected result**	**Pass?**
Did the API call return the sensor readings?	All current sensor readings should be returned as well as indicating if the main LED is on or off.	Yes
Did the circle behind the sensor readings have the correct colour?	If the reading was below the minimum the colour should be blue. If the reading was in range the colour should be green. If the reading was above the maximum the colour should be red.	Yes
Did the graph display the readings for the correct date?	The graph should display the readings for the chosen day.	Yes
Did the override work for the main LED?	The main LED should turn on or off for approximately 60 seconds after sending the corresponding command.	Yes

Table 4.1: Web application Home page tests

4.2.2. Settings Page

Test	Expected result	Pass?
Was it possible to add plant data?	The page should allow you to create new plant settings.	Yes
Was it possible to edit the plant data?	The page should allow you to edit existing plant settings.	Yes
Was it possible to delete the plant data?	The page should allow you to delete existing plant settings.	Yes
Was it possible to have multiple plant settings with the same name?	It should NOT be possible to create multiple plant settings with the same name. The user should be notified if this happens.	Yes
Was it possible to have a minimum setting higher than the corresponding maximum setting?	It should NOT be possible to have a minimum setting higher than the corresponding maximum. The user should be notified if this happens.	Yes
Was it possible to change the active plant?	The settings shown in the 'Current plant' panel should be updated to the newly active plant. Also the Arduino should now be using the new settings.	Yes
Were all the plant settings that had been added visible?	All added plant settings should be visible in the 'All plant data' table.	Yes
Were the correct values saved when saving plant data?	The values stored when adding or editing plant data should be the same as the values entered. This can be checked in the 'All plant data' table.	Yes

Table 4.2: Web application Settings page tests

4.2.3. Sensor Readings Pages

Test	Expected result	Pass?
Did the graph display the readings for the correct date?	The graph should display the readings for the chosen day.	Yes
Did the table display the readings for the correct date?	The table should show all readings for the chosen day	Yes

Table 4.3: Web application Sensor reading pages tests

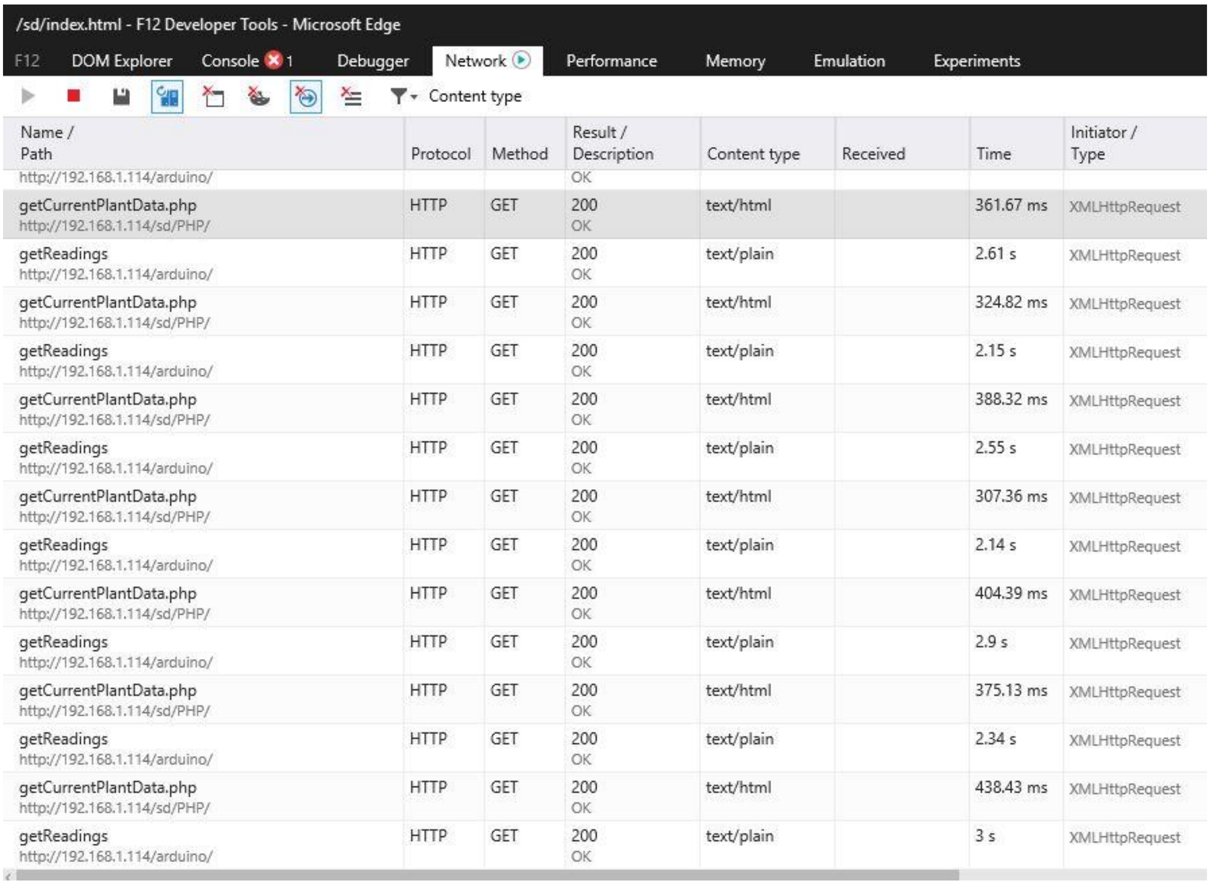
4.3. Agile practice

Throughout the project a physical task board was used to keep track of what work still needed completing. This involved splitting the project into small tasks with a difficulty from 1 – 5, and each week a number of tasks would be selected to be completed. At the end of the week the total difficulty for the week was found from the tasks that had been completed. Tasks for the next week were then chosen and had to equal or better the total difficulty from the previous week. The difficulty for each uncompleted task was also updated at this point if necessary. Selecting the tasks in this way ensured that enough work was being selected for the week.

The task board was a great way to keep track of what had been achieved and still needed to be completed.

4.4. Performance

The performance was tested by utilizing the Developer tools function in the browsers. The main performance was on the home page as this needs to be able to show up to date sensor readings. A timeout out of 10 seconds was selected before requesting new data. It only took around 2 – 3 second to retrieve the readings (Figure 4.1) but the added time accounts for any delays that might occur. If there were any delays then this also prevents a backlog of requests from building up.



/sd/index.html - F12 Developer Tools - Microsoft Edge

F12 | DOM Explorer | Console 1 | Debugger | Network | Performance | Memory | Emulation | Experiments

Content type

Name / Path	Protocol	Method	Result / Description	Content type	Received	Time	Initiator / Type
http://192.168.1.114/arduino/			OK				
getCurrentPlantData.php http://192.168.1.114/sd/PHP/	HTTP	GET	200 OK	text/html		361.67 ms	XMLHttpRequest
getReadings http://192.168.1.114/arduino/	HTTP	GET	200 OK	text/plain		2.61 s	XMLHttpRequest
getCurrentPlantData.php http://192.168.1.114/sd/PHP/	HTTP	GET	200 OK	text/html		324.82 ms	XMLHttpRequest
getReadings http://192.168.1.114/arduino/	HTTP	GET	200 OK	text/plain		2.15 s	XMLHttpRequest
getCurrentPlantData.php http://192.168.1.114/sd/PHP/	HTTP	GET	200 OK	text/html		388.32 ms	XMLHttpRequest
getReadings http://192.168.1.114/arduino/	HTTP	GET	200 OK	text/plain		2.55 s	XMLHttpRequest
getCurrentPlantData.php http://192.168.1.114/sd/PHP/	HTTP	GET	200 OK	text/html		307.36 ms	XMLHttpRequest
getReadings http://192.168.1.114/arduino/	HTTP	GET	200 OK	text/plain		2.14 s	XMLHttpRequest
getCurrentPlantData.php http://192.168.1.114/sd/PHP/	HTTP	GET	200 OK	text/html		404.39 ms	XMLHttpRequest
getReadings http://192.168.1.114/arduino/	HTTP	GET	200 OK	text/plain		2.9 s	XMLHttpRequest
getCurrentPlantData.php http://192.168.1.114/sd/PHP/	HTTP	GET	200 OK	text/html		375.13 ms	XMLHttpRequest
getReadings http://192.168.1.114/arduino/	HTTP	GET	200 OK	text/plain		2.34 s	XMLHttpRequest
getCurrentPlantData.php http://192.168.1.114/sd/PHP/	HTTP	GET	200 OK	text/html		438.43 ms	XMLHttpRequest
getReadings http://192.168.1.114/arduino/	HTTP	GET	200 OK	text/plain		3 s	XMLHttpRequest

Figure 4.1: Network traffic on the web application's home page

4.5. Accuracy

The temperature and humidity readings were compared to second device, a Sensirion SHTC1 [55] (Figure 4.2). The two devices gave very similar results with the temperature generally having a difference of 0.2 degrees and the humidity generally having a difference of 3%.

There were no comparison devices available to see if the lux or soil moisture values were accurate.

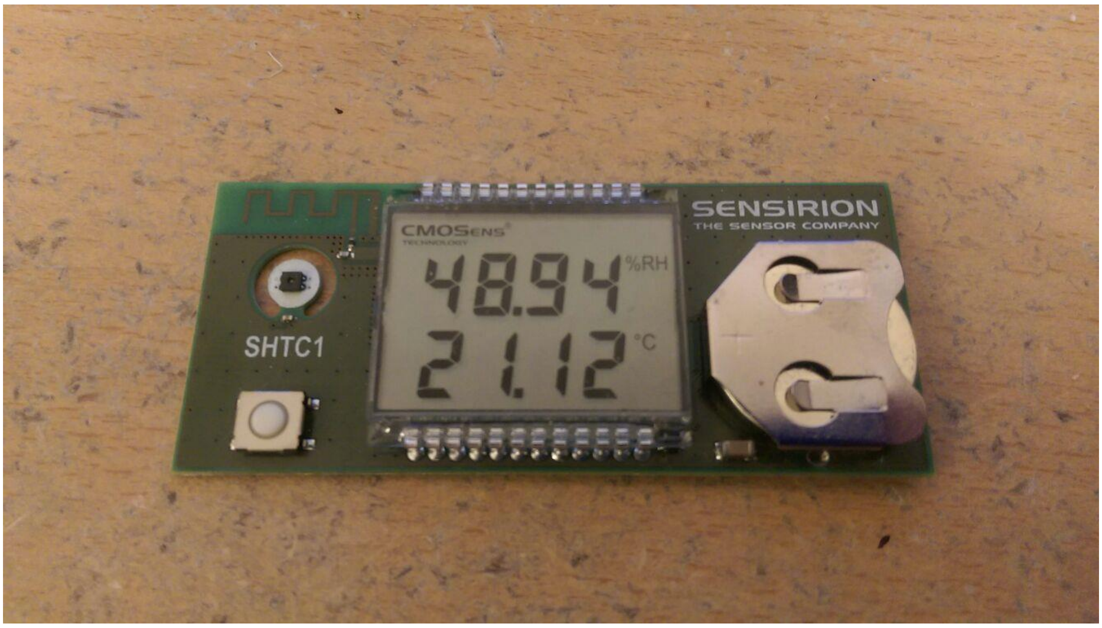


Figure 4.2: Sensirion SHTC1 - Temperature and humidity sensor used for comparison

5. Conclusion

This chapter reviews project as a whole, discussing what was achieved, what was learnt and what could be done to improve it.

5.1. Reviewing the Project Objectives

There were four objectives set at the start of the project:

1. Take Temperature, Humidity, Light and Soil Moisture readings
2. Display past and present sensor readings to the user
3. Be able to update the settings for multiple plants
4. Act upon sensor readings that deviate from the defined range

The first and second objectives have certainly been achieved as readings are being displayed in three different ways: via Console, on an LCD screen and on a webpage.

The third objective has also been completed as the web application allows the user to add multiple plant settings to the database and these can be changed at any time. This can be seen in Figures 3.26 and 3.27.

The fourth and final objective has mostly been achieved. An event did occur whenever any reading deviated from the defined range, but there was no heater or sprinkler to help maintain the environment.

5.2. Personal Development

A number of new skills have been learnt during this project through the use of new technologies and methodologies: the Arduino, Bootstrap and the agile practice of a physical task board. The Arduino was focused around hardware knowledge, bootstrap helped with page styling and using a task board helped with the planning and organisation of the project.

5.3. Future Work

There were a number of improvement ideas that came up during the project implementation and research:

1. Adding a water pump to maintain the water levels in the soil. There are examples of this being carried out already [22] so this would be feasible
2. Controlling a heater to maintain the temperature and humidity. There are also some example projects for this such as: 'Using PID on an Arduino to control an electric heater' [52]
3. To make the sensor readings even more reliable, multiple sensors could be used instead of just one
4. More plant settings could be added in, such as specific night time settings and specified times for devices to turn on. This is because certain plants rely on specific night time regimes. This can easily be done as the Arduino can run a 'date' command using the bridge to check the time
5. More override functionality could be added. For example, turning fans on and off or even overriding the new sprinklers and heaters. This has been done with an LED already so shouldn't be too much trouble
6. The overrides could also use a time limit. This too should be achievable as there is already a 15 minute time limit for taking sensor readings
7. The time limit for taking the readings could also be made variable by the user
8. The greenhouse could also send out alerts such as emails or social media messages to the user to give regular reading updates or say when the heater has come on etc. This too is known to be achievable [53][54]

This is not a finite list of improvements as there could be many more changes made. The Arduino Yun does however only have a limited amount of space for the sketch so if a lot more functionality was to be added then it might be necessary to also look at using multiple boards to control everything.

6. References

1. Hive. URL: https://www.hivehome.com/ (Visited on: 06/11/2015)
2. Control 4. URL: http://www.control4.com/ (Visited on: 06/11/2015)
3. Samsung smart things. URL: https://www.smartthings.com/uk/ (Visited on: 06/11/2015)
4. Apple homekit. URL: http://www.apple.com/uk/ios/homekit/ (Visited on: 06/11/2015)
5. GrowPonics. URL: http://www.growponics.co.uk/ (Visited on: 07/04/2016)
6. Climate Control Systems INC. URL: http://www.climatecontrol.com/ (Visited on:)
7. RHT03 Data Sheet http://cdn.sparkfun.com/datasheets/Sensors/Weather/RHT03.pdf
8. TMP36 Datasheet http://ctms.engin.umich.edu/CTMS/Content/Activities/TMP35_36_37.pdf
9. Temperature extremes
   Weather and Climate Extremes 10 (2015) 4–10: Temperature extremes: Effect on plant growth and development
10. Reducing Humidity in the Greenhouse. URL: https://ag.umass.edu/fact-sheets/reducing-humidity-in-greenhouse (Visited on: 17/04/2016)
11. Botrytis. URL: http://extension.psu.edu/pests/plant-diseases/all-fact-sheets/botrytis-or-gray-mold (Visited on: 17/04/2016)
12. Arduino Yun Description. URL: https://www.arduino.cc/en/Main/ArduinoBoardYun (Visited on: 13/10/2015)
13. ATmega32u4 Datasheet http://www.atmel.com/Images/Atmel-7766-8-bit-AVR-ATmega16U4-32U4_Datasheet.pdf
14. LININO AR9331 Datasheet https://www.openhacks.com/uploadsproductos/ar9331_datasheet.pdf
15. Linino http://www.linino.org/ (Visited on: 13/03/2016)
16. Expand Yun Disk space. URL: https://www.arduino.cc/en/Tutorial/ExpandingYunDiskSpace (Visited on: 16/10/2015)
17. Bridge Library. URL: https://www.arduino.cc/en/Reference/YunBridgeLibrary (Visited on: 16/10/2015)

18. Arduino IDE. URL: https://www.arduino.cc/en/Main/Software (Visited on:16/10/2016)
19. Open Wrt. URL: https://openwrt.org/ (Visited on: 17/10/2015)
20. Yun Package Manager. URL: https://www.arduino.cc/en/Tutorial/YunPackageManager (Visited on: 17/10/2015)
21. Cropking effect of light on plants. URL: https://www.cropking.com/blog/light-greenhouse-how-much-enough (Visited on: 29/04/2016)
22. Arduino Sprinkler system. URL: http://www.instructables.com/id/Arduino-Automatic-Watering-System-For-Plants/?ALLSTEPS (Visited on: 13/10/2015)
23. DHT library. URL: http://playground.arduino.cc/Main/DHTLib (Visited on: 30/04/2016)
24. LDR datasheet http://www.gotronic.fr/pj-1284.pdf
25. Measuring light. URL: http://www.mediacollege.com/lighting/measurement/ (Visited on: 30/04/2016)
26. Soil Moisture sensor. URL: https://solderingsunday.com/shop/arduino/hl-69-soil-hygrometer-moisture-sensor/ (Visited on: 10/03/2016)
27. Transistors. URL: https://learn.sparkfun.com/tutorials/transistors (Visited on: 29/04/2016)
28. P2N2222A transistor datasheet http://www.onsemi.com/pub_link/Collateral/P2N2222A-D.PDF
29. Diode datasheet http://www.vishay.com/docs/88503/1n4001.pdf
30. Arduino for Dummies: John Nussey, 2013
31. White LED http://cdn.sparkfun.com/datasheets/Components/General/YSL-R1047WD-F10.pdf
32. Red LED http://www.arduino.cc/documents/datasheets/LEDR-L-7113ID-12V%28Ver1287713938.7%29.pdf
33. Green LED http://www.arduino.cc/documents/datasheets/LEDG-L-7113GT%28Ver1286952261.13%29.pdf
34. Arduino language. URL: https://www.arduino.cc/en/Reference/HomePage (Visited on: 07/10/2015)
35. Arduino Yun Console. URL: https://www.arduino.cc/en/Tutorial/ConsoleRead (Visited on: 13/03/2016)
36. Arduino Serial. URL: https://www.arduino.cc/en/Reference/Serial (Visited on: 13/03/2016)

37. YunServer. URL: https://www.arduino.cc/en/Reference/YunServerConstructor (Visited on: 13/03/2016)
38. Getting started with the Arduino Yun. URL: https://www.arduino.cc/en/Guide/ArduinoYun (Visited on: 28/04/2016)
39. LCD datasheet http://www.farnell.com/datasheets/31493.pdf
40. Potentiometer datasheet http://www.digchip.com/datasheets/download_datasheet.php?id=1470160&part-number=WIW1028-R-103
41. Bootstrap. URL: http://getbootstrap.com/ (Visited on: 24/02/2016)
42. Effect of temperature on plants. URL: http://www.ext.colostate.edu/mg/Gardennotes/143.html#temp (Visited on: 17/04/2016)
43. Humidity. URL: http://www.jasons-indoor-guide-to-organic-and-hydroponics-gardening.com/plant-growth-and-humidity.html (Visited on: 17/04/2016)
44. Arduino Uno. URL: https://www.arduino.cc/en/main/arduinoBoardUno (Visited on: 13/10/2015)
45. Raspberry Pi 2 Model B. URL: https://www.raspberrypi.org/products/raspberry-pi-2-model-b/ (Visited on: 13/10/2015)
46. Beagle Bone Black. URL: https://beagleboard.org/black (Visited on: 13/10/2015)
47. Embedded Pi. URL: https://www.element14.com/community/docs/DOC-53007/l/embedded-pi-a-triple-play-platform-for-raspberry-pi-arduino-and-32-bit-embedded-arm (Visited on: 13/10/2015)
48. Chart.js. URL: http://www.chartjs.org/ (Visited on: 10/03/2016)
49. Chartist.js. URL: http://www.chartjs.org/ (Visited on: 10/03/2016)
50. Highcharts js. URL: http://www.highcharts.com/ (Visited on: 10/03/2016)
51. ZingChart. URL: http://www.zingchart.com/ (Visited on: 10/03/2016)
52. Using PID to control electric heater. URL: https://georgedewar.wordpress.com/2015/07/27/using-pid-on-an-arduino-to-control-an-electric-heater/ (Visited on: 02/05/2016)
53. 1Sheeld Social media shields. URL: http://1sheeld.com/tutorials/social-media-shields/ (Visited on: 02/05/2016)

54. Twitter and the Arduino Yun. URL: http://www.instructables.com/id/twitter-and-the-Arduino-Y%C3%BAn/?ALLSTEPS (Visited on: 02/05/2016)
55. Sensirion SHTC1 Temperature and Humidity sensor. URL: https://www.sensirion.com/fileadmin/user_upload/customers/sensirion/Dokumente/Humidity_Sensors/Sensirion_Humidity_Sensors_SHTC1_Datasheet_V4.pdf