



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

Web Enabled IoT in a Wireless Sensor Network used to Monitor and Control Reptile Terrariums

by
Austin Hooper

Supervisor: Gaynor Thomas
Second Marker:

Project submitted as part of the requirements for the
award of BSc (Hons) Computing (Software Engineering)

April 2020

Declaration

I, Austin Hooper, declare that I am the sole author of this Project; that all references cited have been consulted; that I have conducted all work of which this is a record, and that the finished work lies within the prescribed word limits.

This work has not previously been accepted as part of any other degree submission.

Form of Consent

I, Austin Hooper, hereby consent that my Project, submitted in candidature for the B.Sc. (Hons) Computing (Software Engineering) degree, if successful, may be made available for inter-library loan or photocopying (subject to the law of copyright), and that the title and abstract may be made available to outside organisations.

Abstract

Reptile care has evolved slowly, with relatively simple technology being used in the form of thermostats and timers for heating, lighting, or spraying. This project aimed to develop a web-enabled IoT system for automatically monitoring and controlling temperature, humidity and ultraviolet exposure in reptile terrariums, enabled through a wireless sensor network. Environmental control targets were $\pm 1^{\circ}\text{C}$ and $\pm 10\%$ relative humidity. An informal interview was conducted with local pet shop staff to gain industry context, along with a literature review to gain academic context. ESP8266 microcontroller boards along with DHT22 sensors were used to develop network nodes, which communicated back to a Raspberry Pi 3B+ acting as a database and web server. Arduino (C/C++), PHP, JavaScript, SQL, HTML and CSS were the primary technologies used to implement the system. Functional testing was undertaken to verify the system's ability to control its environment. The system was also evaluated against an existing product. User experience testing, including the System Usability Scale, was used to gauge the web application's usability with 15 participants. Results showed that in four 36 hour tests, temperature control with a 32°C target was effective, though did not meet the target tolerances. Temperature exited the target range by 0.4°C at worst. Despite this, the mean temperature over this time period was 31.95°C . Humidity control was also effective, though again exited the target threshold. Humidity control code was modified and tested to validate the impact of the modifications. Before modification, the mean humidity was 69.22%, with a range of 61-79%. After modification, the mean was 78.67%, with a range of 70-90%. Combination temperature and humidity control was unsuccessful, as the heating element overpowered the mister and the humidity did not increase to the target. The system performed favourably in comparison to an existing product, with a mean $\sim 1^{\circ}\text{C}$ closer to the target. Usability testing was successful – the application scored 78/100 in the System Usability Scale, against the average of 68, and was well received overall. Future work has been identified, primarily further networking features, improvements to the tolerances of the system, and applying the system to the care of other captive animals. The web application could be made to be more accessible, taking into consideration usability testing feedback. The system was successful overall in monitoring and controlling environmental variables, and the web application was found to be usable and well received.

Table of Contents

1. Introduction	1
1.1. Project Background and Overview	1
1.2. Aims and Objectives	4
1.3. Report Structure	5
2. Research	6
2.1. Literature Review	6
2.1.1. Smart Devices for Monitoring and Caring for Animals	6
2.1.1.1. Maia Zoo Terrarium Monitoring and Control	6
2.1.1.2. IBM and Marwell Zoo Energy Saving	8
2.1.1.3. WSN Honeybee Hive Monitoring	8
2.1.1.4. IoT Henhouse Remote Monitoring	11
2.1.1.5. WSN Fishpond Monitoring	12
2.1.1.6. Sea Water Monitoring Device for STEM Education	14
2.1.1.7. WSN Livestock Health Monitor Using Wireless Charging	14
2.1.2. Environment Monitoring and Control	15
2.1.2.1. SMS Enabled IoT Greenhouse Control	15
2.1.2.2. Automatic IoT Greenhouse Control	17
2.1.2.3. IoT Air and Water Quality Monitoring	18
2.1.2.4. IoT Crop Field Control Across Multiple Locations	18
2.2. Currently Available Commercial Systems	19
2.3. Informal Interview	23
2.4. Research Conclusion	23
3. Research and Development Methodologies	25
3.1. Research Methodology	25
3.1.1. Research Philosophy	25
3.1.2. Research Approach	25

3.1.3. Research Strategies	26
3.1.4. Data Collection and Analysis	26
3.2. Development Methodology.....	27
3.3. Project Risks	27
3.4. Hardware and Software Resources	30
3.5. Research and Development Methodologies Conclusion.....	31
4. System Design and Requirements	32
4.1. Requirements.....	34
4.2. Raspberry Pi Server.....	35
4.2.1. Database Design	36
4.2.2. Website Design.....	37
4.3. Node Devices.....	40
4.3.1. Node Hardware.....	41
4.3.2. Node Software	43
4.3.3. Node Device Case 3D Modelling	50
4.4. System Design and Requirements Conclusion	50
5. System Development.....	52
5.1. Web Server Development.....	52
5.1.1. Register and Log in.....	52
5.1.2. Sending Data to Nodes.....	53
5.1.3. Receiving Node Data.....	53
5.1.4. Dashboard	53
5.1.5. Logs.....	55
5.1.6. Progressive Web App Features	56
5.2. Node Device Development	59
5.2.1. Node Device Software	60
5.3. Troubleshooting	63

5.3.1. Web Application Graph.....	63
5.3.2. SSL and HTTPS	65
5.3.3. Sensor Readings	65
5.3.4. Node Power	65
5.4. System Development Conclusion	66
6. Testing.....	67
6.1. Functional Testing	67
6.1.1. Standalone Surface Temperature Control	68
6.1.2. Standalone Humidity Control	74
6.1.3. Combination Temperature and Humidity Control.....	78
6.4. Web Application User Experience and Accessibility Testing.....	80
6.4.1. User Experience Testing.....	81
6.4.2. HTML and CSS Validation	86
6.5. Testing Conclusion.....	86
7. Evaluation and Discussion.....	87
7.1. Discussion of Results.....	88
7.1.1. Functional Testing Results.....	88
7.1.2. Usability Testing Results.....	90
7.2. Recommendations and Suggestions.....	91
8. Conclusions and Future Work	93
8.1. Future Work	93
8.2. Conclusion	94
9. Reflection.....	96
References.....	98
Bibliography	104
Appendices	106
Appendix A – Project Timeline.....	106

Appendix B – Hardware and Software Resources.....	107
Appendix C – Page Wireframes	110
Appendix D – ESP8266 Array Handling and Sensor Reading Subprocesses	118
Appendix E – SSL Self-Signed Certificate Process	120
Appendix F – Hardware Development Photos.....	123
Appendix G – Sample User Experience Test and Questionnaire	127
Appendix H – Usability Feedback Changes.....	136

Table of Figures

Figure 1.1 - Heat lamp, normally fitted with infrared bulbs [7]	2
Figure 1.2 - Heating mat / pad [8].....	2
Figure 1.3 - Typical UVB bulb used for many reptiles [9]	3
Figure 2.1 - Terrarium control system by [12].....	7
Figure 2.2 - Hive monitoring website implemented by [14].....	9
Figure 2.3 - Henhouse web monitoring interface by [17].....	11
Figure 2.4 - MATLAB data distribution from [18]	13
Figure 2.5 - Greenhouse monitor and control system from [22]	15
Figure 2.6 – Air and water quality monitor system [25].....	18
Figure 2.7 - Microclimate MiniStat 100 thermostat [27]	19
Figure 2.8 - Microclimate EVO thermostat graph [30]	20
Figure 2.9 - Komodo Pulse 600W thermostat - personal unit.....	21
Figure 2.10 – ReptiRain sprayer - personal unit.....	22
Figure 3.1 - Risk matrix	29
Figure 4.1 – Network design	33
Figure 4.2 - Web server logical architecture.....	35
Figure 4.3 - Crow's foot relational database model	36
Figure 4.4 - Web application site's structure.....	37
Figure 4.5 - Home page logged in wireframe	39
Figure 4.6 - Dashboard individual view wireframe.....	39
Figure 4.7 - Node device logical architecture	40
Figure 4.8 - Node device wiring diagram.....	41
Figure 4.9 – Node device relay configuration	42
Figure 4.10 - ESP8266 main loop flowchart.....	43
Figure 4.11 – ESP8266 ReadServerData function flowchart.....	44
Figure 4.12 - ESP8266 ReadAndReact function flowchart pt. 1	45
Figure 4.13 - ESP8266 ReadAndReact function flowchart pt. 2	46
Figure 4.14 - ESP8266 ReadAndReact function flowchart pt. 3.....	47
Figure 4.15 - ESP8266 ReadAndReact function flowchart pt. 4	48
Figure 4.16 - ESP8266 SendDataToServer function flowchart	49
Figure 4.17 - Node device case render	50
Figure 4.18 - CAD drawing of final node case.....	51

Figure 5.1 – Web server read parameters for node devices code.....	53
Figure 5.2 – Website dashboard summary view	54
Figure 5.3 – Website dashboard individual view	54
Figure 5.4 - Example AJAX code for retrieving website log data.....	55
Figure 5.5 – Progressive web app install pop-up	56
Figure 5.6 - Progressive web app in Windows	57
Figure 5.7 - JavaScript notification code	58
Figure 5.8 - Notification popup	58
Figure 5.9 - Node device internals.....	59
Figure 5.10 - Node device in 3D printed case	60
Figure 5.11 - Node SendDataToServer code	61
Figure 5.12 – Node code for reading each parameter separated by comma	61
Figure 5.13 - Node ReadAndReact code	62
Figure 5.14 – Log graph with reading count limit.....	63
Figure 5.15 – Log graph with time range limit	64
Figure 5.16 – Troubleshooting: real-time clock voltage reading	66
Figure 6.1 - Testing setup	67
Figure 6.2 - Temperature test without $+1\frac{1}{3}^{\circ}\text{C}$ code offset.....	71
Figure 6.3 - Temperature test with $+1\frac{1}{3}^{\circ}\text{C}$ code offset.....	72
Figure 6.4 - Extended test runs with $+1\frac{1}{3}^{\circ}\text{C}$ code offset	73
Figure 6.5 - Heating lamp setup	74
Figure 6.6 - Initial standalone humidity control test runs	76
Figure 6.7 – Standalone humidity control test runs after code change.....	77
Figure 6.8 - Combined temperature and humidity control	79
Figure 6.9 – Overall SUS scores per participant	82
Figure 6.10 - SUS mean scores with standard deviations.....	83
Figure 6.11 - SUS question mean scores, negative vs positive responses	84

Table of Tables

Table 3.1 - Project risks, including risk exposure calculations.....	28
Table 3.2 - Hardware and Software resources	30
Table 6.1 - Temperature test runs (°C) without +1/3°C code offset.....	69
Table 6.2 - Temperature test runs (°C) with +1/3°C code offset.....	69
Table 6.3 – 36 hour temperature test runs (°C).....	69
Table 6.4 - Microclimate Ministat 100 test runs (°C).....	70
Table 6.5 – Initial standalone humidity control test runs (%)	75
Table 6.6 – Standalone humidity control test runs after code change (%)	75
Table 6.7 – Combined temperature and humidity control results	78
Table 7.1 – Comparison of Ministat 100 and project (32°C target).....	89

1. Introduction

1.1. Project Background and Overview

Reptiles are ectothermic, relying on the environment around them to regulate their body temperature, therefore, providing appropriate heating is essential. Excluding general requirements such as a spacious enclosure, food, and water, the primary requirements of reptiles are adequate temperature, humidity and lighting [1]–[3]. The scope of this project is limited to environmental conditions required specifically by lizards. The most common environmental requirements of lizards are:

- Ambient heating for basking (lying exposed to heat and light, usually from the sun or artificial heat sources), usually provided by a heating lamp (see Figure 1.1)
- Surface heat, commonly to heat their stomachs to help digest food, usually provided by a small heat mat / pad (see Figure 1.2)
- A specific humidity level dependent on the species
- Some require ultraviolet light exposure, usually provided by UV-B lamps for a set number of hours or time range per day (see Figure 1.3)

The primary requirement for captive reptiles is an environment with the appropriate temperature. Typical appropriate basking area temperatures for lizards are 28°C – 40°C, depending on the species. As discussed by C. Warwick et al. [4] and John Van Demark of Rhinebeck Animal Hospital [3], some of the effects of temperatures too low on reptiles are hypoactivity (inactivity), anorexia due to food processing difficulties, susceptibility to infections, and hypothermia. On the other hand, temperatures too high can result in burns, gasping and panting due to hyperthermia (overheating), and reluctance to eat, among other effects. Furthermore, C. Warwick et al. state that inadequate thermal conditions “may result in the exacerbation of acute stress as well as chronic debilitation,” further highlighting the importance of correct temperatures for reptiles.

Environment control is also extremely important for reptile egg incubation. Incubation temperature has been linked to hatchling behaviour and intelligence [5], [6].



Figure 1.1 - Heat lamp, normally fitted with infrared bulbs [7]



Figure 1.2 - Heating mat / pad [8]



Figure 1.3 - Typical UVB bulb used for many reptiles [9]

Currently, there are commercially available products which provide controllable heating or spraying based on environmental temperature or timer systems. However, these systems often lack monitoring features or more advanced control options. These systems are expensive per unit, making them unsuitable for controlling a large number of terrariums, such as in a pet shop or zoo setting – each terrarium would require an individual control system be purchased. Some of these products are discussed further in the Literature Review section.

To address these drawbacks, this project aims to develop a single system which monitors and controls the environmental conditions of multiple terrariums, using a wireless sensor network and a PHP web application.

This project will involve the development of a monitoring and control system for multiple reptile terrariums. A terrarium is a “container, often with a glass front, in which small reptiles or other small creatures are kept” [10]. Reptiles have specific environmental requirements which should be met by their terrarium enclosure in order for them to live healthily and happily. As discussed by UK Parliament [2], reptiles’ needs for specific environments, among other needs, are even covered by Section 9 of the Animal Welfare Act 2006 [11], stating, “For the purposes of this Act, an animal’s needs shall be taken to include—(a)**its need for a suitable environment...**”

This project will take input from a local pet shop in Swansea, and will use interviews to identify specific problems and approaches which can be taken to mitigate them.

From this background information, a problem statement can be developed:

Current single-terrarium control systems available on the market are not capable of advanced monitoring and control, and are often very expensive. This means many people caring for reptiles, particularly in pet shops, take a manual approach to temperature and humidity regulation, often with a detriment to reptiles' health. A system needs to be implemented to address this.

This project began in June, and will last until late April. Research is a continuous process, being undertaken throughout the entire project. The timeline is shown in detail, including design, implementation, and documentation time frames, in Appendix A.

1.2. Aims and Objectives

The project can be broken down into the following aim, and objectives to achieve this aim.

Aim:

To design and develop a web-enabled system to automatically monitor and control temperature, humidity and ultraviolet exposure in multiple networked reptile terrariums.

Objectives:

- Conduct a literature review as secondary research
- Conduct informal interviews as primary research to determine specific problems with current solutions and requirements
- Design and develop an ESP8266 based monitoring and control system using sensors and actuators
- Design and develop a responsive and accessible PHP progressive web application to monitor and control the node hardware
- Test and evaluate the system against existing consumer systems

1.3. Report Structure

This report documents the research, planning, development, and testing of the proposed system. Chapter 2 reviews the current literature relevant to the topic and discusses areas which can be further developed in the area, along with discussing consumer products and an informal interview. Chapter 3 discusses and justifies the research and development methodologies to be used throughout the project.

Chapter 4 provides details of the system design, while Chapter 5 details implementation of the design. Chapter 6 details testing undertaken on the system and provides results. Chapter 7 evaluates the project overall and the efficacy of the developed system, discussing the results and providing suggestions for improvement. Chapter 8 concludes the report and discusses the potential for future work. Chapter 9 reflects on the project, followed by a references list, bibliography, and Appendices.

2. Research

2.1. Literature Review

This section details research consisting of existing products, an informal interview, and a literature review. The literature review research topic can be separated into two primary areas: IoT in animal and pet monitoring and care, and environment control and monitoring using IoT. The use of smart devices in “weather station” concepts are well-established and developed, though use of electronic systems in the care and monitoring of animals, specifically pets, is less developed.

2.1.1. Smart Devices for Monitoring and Caring for Animals

2.1.1.1. Maia Zoo Terrarium Monitoring and Control

The most directly comparable paper found in this area, by P. Santos, J. Alves, and P. Carneiro [12] investigates reptile terrarium control. The paper introduces its concept by discussing the importance of temperature regulation for reptiles due to their ectothermic nature; meaning they rely on external heat sources to regulate their own body temperature. This results in captive reptiles’ reliance on heat sources such as heat lamps or mats. The research is described as part of a project involving an undergraduate internship in partnership with Maia Zoo, which consists of developing a monitoring and control system for a reptile terrarium’s ambient temperature.

The paper describes the configuration of appropriate hardware and software for the project: An Arduino Uno microcontroller board which uses a custom calibrated NTC thermistor to determine current temperature, and uses relays controlling a fan and 230V lamp to cool or heat the enclosure when necessary. This is shown in Figure 2.1. Additionally, the Arduino Uno communicates data through USB to a computer which then stores the data in a database for viewing on a web server. The custom calibration of a thermistor is appropriate here due to the configuration and focus, though for the Major Project, a pre-calibrated sensor such as a DHT11 or DHT22 is more appropriate as a starting point, as calibrating a thermistor is outside the project scope.

There is a simple visual display implemented, using three LEDs to display the current temperature relative to the target temperature. This is a useful implementation that could be developed further to display more detailed information. The web interface provides basic monitoring features, and is secured using a username and password. Security is an essential component of a system which live animals are reliant on. It also provides alarm information (text on the page) for temperature peaks, an idea which would be useful to consider for the Major Project.

The paper is well illustrated, using diagrams to detail the system architecture and control. It focuses primarily on the monitoring of temperature rather than the control, and while it is assumed that such a system would improve temperature consistency, there is no discussion of the results of using the system – only the results before implementing it.

There is room for expansion of this system. As the paper discussed, humidity monitoring and control would also be useful for many reptile species. This would even allow the system to be expanded for amphibians, given their requirement of consistent, high humidity levels. Additionally, it should be noted that this system controls one terrarium – to control multiple, as may be the case in a zoo setting like the paper targets, would involve setting up this system for each terrarium. This could be expanded by incorporating a network or centralised logic to allow for the control of multiple environments by one system.



Figure 2.1 - Terrarium control system by [12]

2.1.1.2. IBM and Marwell Zoo Energy Saving

Similarly to the previous paper, Marwell Zoo, in collaboration with IBM, implemented a system using the Watson IoT Platform to provide many animals with more comfortable housing, while reducing heating energy costs [13]. The system was tested on Nyala enclosures, where infrared heating elements were only enabled when animals were detected inside the enclosure at night. Previously, the heaters had been toggled on a timer-based system, regardless of the occupancy of the enclosure being heated, thus wasting energy. The system gathered 4x4 grids of temperature readings per enclosure and used AI running in a cloud application to determine if any Nyalas were present. The system used many consumer components such as an ESP8266, Raspberry Pi, and USB battery banks.

2.1.1.3. WSN Honeybee Hive Monitoring

Another paper with a similar concept by E. Henry et al. [14] develops a system to monitor the temperature, humidity, noise volume and noise frequency of bee hives. The paper implements a wireless sensor network with each node belonging to a hive, which connects to a central server that gathers the data and displays it to a web front end. The paper also tests and discusses the effect of 2.4GHz Wi-Fi signals on bees.

The benefits proposed by the authors of such a system is that data gathering with the implemented system will not require opening the beehive, and thus not aggravating the colony. The authors also discuss the lack of quantitative data collected for the internal dynamics of beehives due to the limited nature of observation.

Each node device was enclosed in plastic containers to avoid moisture damage; however, the sensors were left exposed for accurate readings. The microcontroller board used was a Raspberry Pi 2B, with a DHT22 sensor used for temperature and humidity monitoring, and a PLMC15 omnidirectional microphone used for acoustic measurements. This model of Raspberry Pi is generally not necessary for this purpose, it is more powerful than this application requires, at the cost of a higher power draw and larger physical footprint. A potential drawback of using such a device for this application is that due to the power draw, it may introduce additional

heat into the hives (as waste energy from the processor), affecting results. If the microphone used was not interfaced over USB, a far smaller and lower power board could have been used such as a Raspberry Pi Zero W, or Arduino based boards. For the purpose of gathering in-hive data, the DHT22 is a suitable balance of performance and affordability, and would also be appropriate for the Major Project implementation. The node devices and the central server were both programmed using Python, with the server hosting a MongoDB database and a web server to visualise the data, as is seen in Figure 2.2. This architectural approach of storing data from many hives locally on an internet-connected server PC is discussed further by A. Kviesisa and A. Zacepinsa [15].

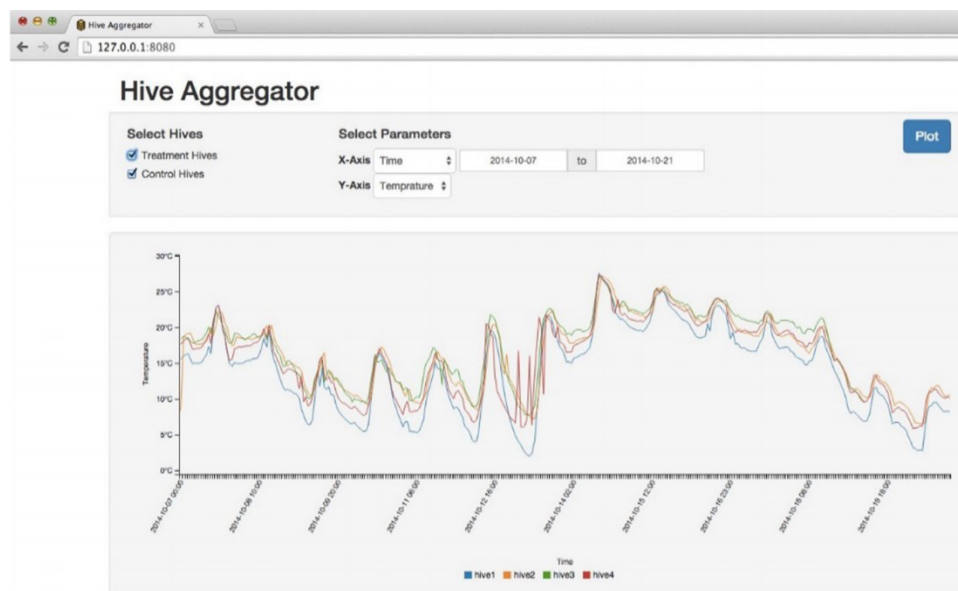


Figure 2.2 - Hive monitoring website implemented by [14]

As part of testing, temperature and humidity from the area (Montreal Island Weather Station, Dorval, Quebec) were both monitored and compared with in-hive readings. It was found that in-hive temperature correlated with external temperature; however, internal and external humidity levels did not correlate. The authors note that this reflects the bees' abilities to regulate hive internal humidity. This highlights the benefits of implementing a sensor network, as this data would be difficult to gather and analyse reliably without interfering with the hives. A similar approach could possibly be taken with the Major Project, to measure how well the proposed system regulates its temperature and humidity relative to the room it is in.

To test the impact of Wi-Fi on the bees, a control hive group was set up, connected to the server using ethernet and located outside of the Wi-Fi range (30 metres). The other group was within 1 metre of the Wi-Fi router. Wi-Fi exposure was not found to have a significant effect on temperature, humidity or noise levels of the hives, the authors noted the changes were within the standard deviation range. However, this testing only covered a limited number of variables with one specific insect, and the authors discuss the need for further testing and the possibility that wireless frequencies could impact bees at the physiological level, especially long term.

Sensor data revealed spikes in microphone readings at night, which is suspected to be evidence of animal disturbance. Real time detection of such problems would allow beekeepers to take immediate action against disturbances. This is significant, as it is stated that external animal disturbances accounted for 2.3% of beehive mortality in the United States in 2008 – 2009. The Major Project implementation could also be used to notify reptile owners of significant environmental changes to allow them to act quickly.

Data analysis also showed that in the control hive, environmental readings were very varied. This is explained to be due to a swarming event of that bee colony, which had been logged and detected unintentionally by the system. This concept has also been explored by A. Zacepins et al. [16] in a paper detailing monitoring of hive swarms using DS18S20 temperature sensors connected to Raspberry Pi microcontrollers. For ten hives, over approximately three months, measurements were taken every 60 seconds and saved to a MySQL database. Analysis showed that swarms could be easily detected based on temperature alone, as bees preparing to leave the hive resulted in increased hive temperatures of 1.5 – 3.5°C. This further highlights the potential benefits and usages of IoT devices to beekeepers.

2.1.1.4. IoT Henhouse Remote Monitoring

A similar system was developed by H. Li et al. for henhouse environment monitoring [17]. It monitored the ambient temperature, humidity, CO₂ and NH₃ and stored the data in a MySQL database, linked to a web server (seen in Figure 2.3) for remote access. A windows form interface for detailed setup and monitoring is also included. The site used AJAX functionality to achieve real-time updating graphs, an approach many other papers have omitted. This could be useful to implement in the Major Project as it will also have a monitoring element that may benefit from the real-time aspect. However, the user interface of this paper's work could be further developed, possibly by implementing a web front end for the current windows form, making it more accessible and allowing for styling and enhancement with CSS and JavaScript, as it had lots of information in one place and was difficult to read. An estimation algorithm was also developed to predict environment variables over time, which was proven to be very accurate. This may be more appropriate as future work for the Major Project.

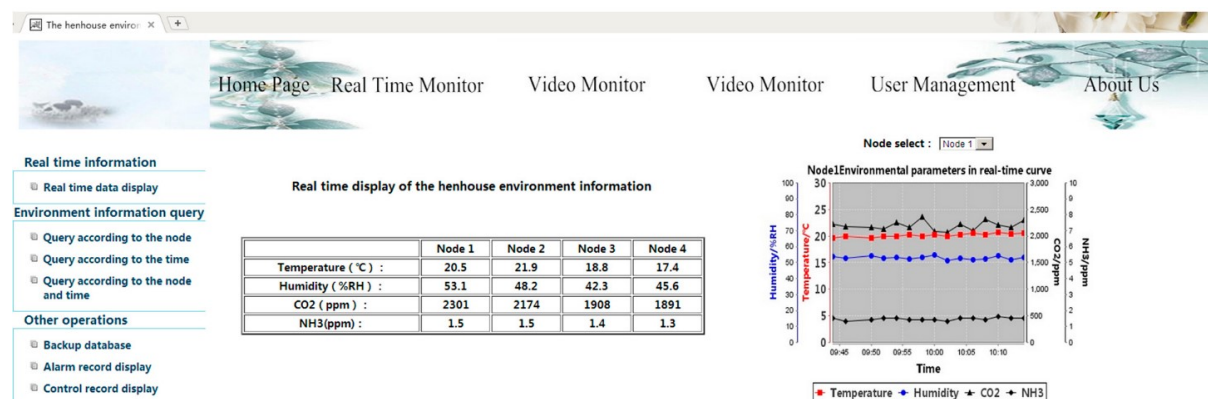


Figure 2.3 - Henhouse web monitoring interface by [17]

2.1.1.5. WSN Fishpond Monitoring

A further paper in this area by B. Shi et al. [18] details the development of a wireless sensor network (WSN) for monitoring water dissolved oxygen (DO) levels and temperature of a fishpond, for aquaculture. Aquaculture is farming or harvesting from water - this paper aims to aid in improving the health of livestock, and increasing yield. The authors implement a WSN using Texas Instruments CC2530 microcontrollers which gather data and transmit it back to a monitoring centre, consisting of a computer to monitor data, a web server for accessing data, and a database server to store sensor data. The paper also tests the effects of different power-saving methods and the effects of three different WSN topologies.

The website was written in C# using the Microsoft ASP.NET framework, communicating with a separate computer running a Microsoft SQL Server database. For medium to large scale applications such as the one proposed in this fishpond WSN paper, it may be beneficial to host a web server and database on different machines. However, for the Major Project, this can be achieved on a single machine.

The node devices were based on the Texas Instruments CC2530 microcontroller, using DS18B20 sensors for water temperature monitoring and DO-954A sensors for DO level monitoring. For the Major Project, it would be more suitable to use a more commercially available microcontroller such as an Arduino Nano or ESP8266 as there is more documentation and guidance for these products. The DS18B20 sensor is used appropriately here as it is a waterproof probe type sensor and is accurate enough for the purpose.

The nodes were deployed in waterproof buoys at different positions, making maintenance difficult due to the need to retrieve them. As a result, power saving is essential to reduce the need for battery changing or charging. The authors detail power saving measures taken, primarily using sleep states on the devices. An ESP8266 would be appropriate for this project as it allows for a similar configuration to that used in the paper, as it allows for “deep sleep” to conserve energy. Power usage was tested in active mode, and in two sleep modes. After 20 days of sensor node runtime, active mode resulted in 19% battery charge remaining, while the two sleep modes resulted in 43% and 58% battery charge remaining, respectively. The sleep modes were less effective on gateway nodes as they needed to be active for

more time to handle incoming data. However, this still shows a large improvement when using deep sleep on these devices and thus this approach should be taken when developing this Major Project system.

The paper also discusses testing packet loss in star, mesh, and tree WSN topologies. When data was sent at 2 second intervals, the star topology performed the worst, losing 33.3% of packets sent. The authors suggest this may be because some nodes were too far away from the central sink node. After star was the mesh topology, with a packet loss rate of only 4.1%, and the tree topology performed the best at 3.1% packet loss. As this Major Project implementation will involve many nodes in near proximity, it is likely that the star topology will perform well, as there are less hops from sensor node to the sink node than in other topologies. A star topology would also be significantly simpler to implement. However, other topologies will be considered as they may be appropriate to allow the system to expand in future work.

The paper discusses the observations made from using the system to monitor DO and temperature over the span of approximately 2 months. Data from one day was used to demonstrate the results. The authors highlight that at 11:00 AM, both the DO concentration and temperature of the water were higher than at 11:00 PM on the same day. This data highlights the uses of such a system to gather data which would otherwise not be available. Below in Figure 2.4 is some sample data for DO concentration produced by the system:

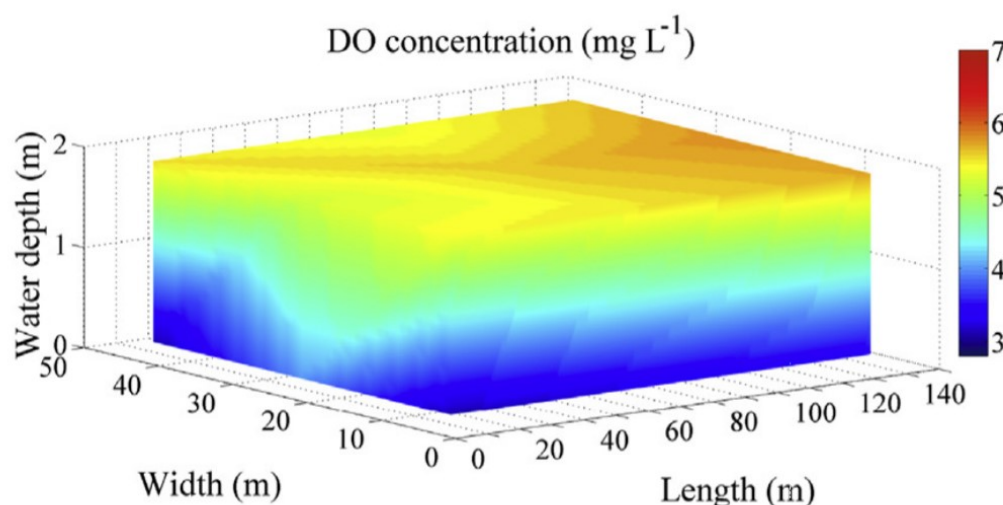


Figure 2.4 - MATLAB data distribution from [18]

2.1.1.6. Sea Water Monitoring Device for STEM Education

A similar device was prototyped by C. Tziortzioti et al. [19], sensing water temperature, DO and pH levels, among other values, of sea water near beaches. The data was intended to be used in STEM teaching, with an Arduino logging sensor data to an SD card. No network connection meant more portability, but also lower power consumption. However, for a larger-scale system, network capability, for example using a SIM card, would allow for networking of many devices as nodes. The authors encountered accuracy problems with some sensors, and some were not appropriate for use in water, so could have been replaced with different models. Despite this, the device was successful in acquiring data that could be used in an educational context.

2.1.1.7. WSN Livestock Health Monitor Using Wireless Charging

L. T. Beng et al. [20] also discussed power usage concerns in WSNs, especially the difficulty of replacing power sources, and proposed a custom wireless power transfer system enabled WSN for monitoring livestock health. This was undertaken as standards for consumer wireless charging are only capable of extremely short distance power transfer, such a maximum distance of 2.5mm for the Qi standard [21]. This paper also used deep sleep modes on node devices, however, proposed the node devices be wirelessly charged each day when the livestock returns to the ranch. The system monitored temperature sensors that were implanted under livestock's skin and communicated over NFC to a base station if temperatures were above a threshold. The benefit of this approach is eliminating the need to change batteries for charging.

2.1.2. Environment Monitoring and Control

2.1.2.1. SMS Enabled IoT Greenhouse Control

A paper by P. V. Vimal and K. S. Shivaprakasha [22] details the development of a system to monitor and control soil moisture, light levels, temperature, humidity and soil pH levels of a greenhouse. The system is connected to a GSM enabled circuit board with a SIM card to facilitate SMS message sending and receiving with other mobile devices.

The system is based on the Arduino Uno microcontroller, enclosed in a water-tight box. This is suitable in this case as it has an RJ45 port for ethernet communication, and can output 5V meaning it is able to power all of the actuators connected to it, without needing external power. This approach will not be suitable for the Major Project as higher voltages may be required and there is also a need for wireless communication. The prototype can be seen in Figure 2.5 below:



Figure 2.5 - Greenhouse monitor and control system from [22]

Parameters are provided to the system via an android mobile application, which are then saved to a database. This is an appropriate method, though a web application would be more suitable as it would allow for access on any device capable of running a web browser, and not only mobile devices. The application is also stated to have monitoring functionality, though this is not shown in the paper.

Environment control is achieved as follows: soil pH levels by spraying alkaline or acidic fluids when necessary, soil moisture by powering a pump, temperature using a cooling fan, and light level through powering artificial lighting elements. An element of this setup which could be improved is the temperature regulation, as this system can only cool the greenhouse and not heat it up. A heating element such as a ceramic or infrared lamp could be incorporated to allow the system to be used in colder climates, or to grow plants which require warmer environments.

Temperature and humidity monitoring are achieved using the DHT11 sensor. While appropriate for experimental and development purposes, the DHT11 sensor only has a temperature accuracy of $\pm 2^{\circ}\text{C}$ [23], This means that a temperature reading of 25°C could be anywhere between 23°C and 27°C in reality. It is however acknowledged that this paper is more intended as a proof of concept. For caring for live animals, however, a more accurate sensor would be beneficial. The pH, soil moisture and light-dependent resistor (LDR) sensors are described, but their specific models are not provided; so it is impossible to know if they were reliable or accurate sensors. An LDR sensor could possibly be incorporated into the Major Project in order to align ultraviolet exposure times with daylight times, though this may be more appropriate for future work.

The primary flaw with the implemented control system is the need for some level of human intervention. When a sensor reads a value outside of the parameters (for example, the temperature reading is higher than expected), an SMS alert is sent to the specified user, who can then control the system by texting key-words such as "LIGHTON". For crops with very specific requirements, or for live animals, immediate action would be required - waiting for user input would negate the benefits of such a system. This also adds an unnecessary step, though it does provide the user with the ability to make the decisions.

2.1.2.2. Automatic IoT Greenhouse Control

S. N. Nnadi and F. E. Idachaba implemented a similar system [24] which further explores the idea. The paper highlights the significance of more advanced agriculture due to growing food demand, along with the benefits of requiring little human intervention and the ability to grow crops regardless of the season. This system automates the control of temperature, humidity, light intensity, and soil moisture. It discusses previous developments in the area of greenhouse automation, making appropriate suggestions for further development and improvement.

The primary improvement over the previously discussed paper's concept is that this system is fully automated, with no requirement for human intervention to control relays and thus control environmental conditions. This system uses a similar microcontroller board: the Arduino Mega, and again uses a DHT11 sensor. The paper discusses the requirement for the system to be cost effective to be worth operating, so the lower-cost sensors are justified here – though not ideal due to accuracy. Similarly to the previously discussed reptile terrarium control system, this system also uses the following languages and formats to achieve its functionality: Arduino (C / C++), PHP, JSON, XML and CSV. The system also allows for sensor data to be viewed via the ThingSpeak IoT analytics platform. This data is then read by an Android application to allow for easier remote monitoring, and allows the user to override the automatic settings and power or un-power any actuator.

Similarly to previous papers, the control elements used in this system include a heating bulb, artificial lighting, a water pump, and exhaust fan. This system has an appropriate information display on-site, implemented using a 20x4 character LCD display. This, along with the communication and data handling methods, would be useful to carry forward in the development of a new reptile care system.

The paper discusses tests conducted growing beans, stating that those planted with no monitoring and control had a survival rate of 5 out of 10, while those under the care of the system had a survival rate of 9 in 10. Though this is a significant improvement, it a sample size of only 10, so would not be appropriate to demonstrate the efficacy of the system.

2.1.2.3. IoT Air and Water Quality Monitoring

A further benefit to the environment which IoT can provide was discussed by S. Anupamashree et al. [25], who developed an air and water quality monitoring system which logs data to the Blynk IoT platform, and sends SMS messages when safe thresholds are exceeded. The system was implemented using an Arduino ATmega328P, and NodeMCU used to facilitate wireless communication. This prototype can be seen below in Figure 2.6. The system monitors SO₂, NO₂, CO₂ levels of air, and temperature, transparency, and pH in water near industrial areas.

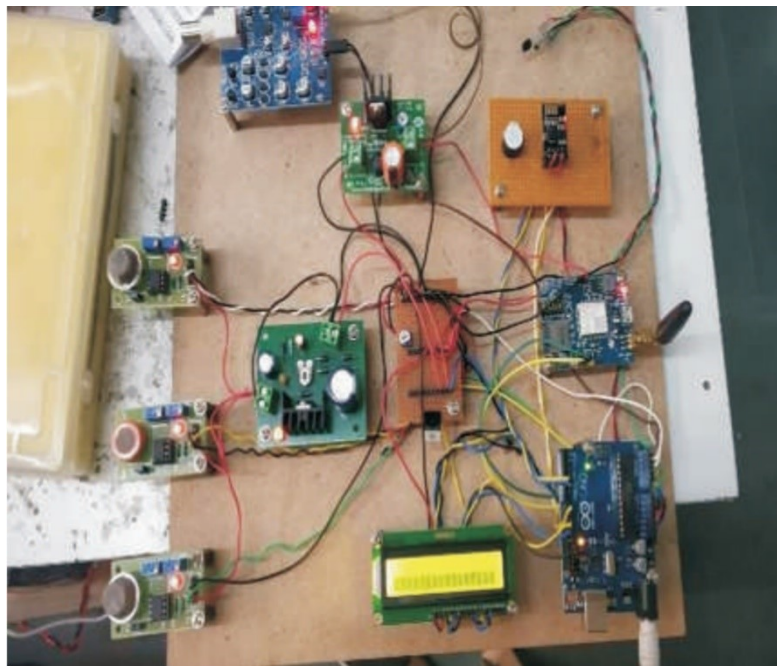


Figure 2.6 – Air and water quality monitor system [25]

2.1.2.4. IoT Crop Field Control Across Multiple Locations

Another paper in this area by J. Muangprathub [26] explored a more general system architecture which could be applied to a variety of agriculture scenarios. The developed system used a NodeMCUs (ESP8266) and DHT22 sensors to monitor temperature and humidity and control relays in crop fields across three villages, growing different plants. The primary purpose in this instance was to water plants by powering solenoid valves. A simple web application was hosted directly on the NodeMCU and made accessible over the internet. The system was successful, reportedly maintaining appropriate humidity levels while reducing costs.

2.2. Currently Available Commercial Systems

This subsection details some currently available products for controlling reptile terrarium temperature and humidity. Each of the thermostats discussed do not include the heating elements themselves, these are purchased separately and controlled via a mains adapter on the thermostat.

Microclimate MiniStat 100 [27] and Pulse B2 HT [28]

One of the most popular reptile thermostat brands is Microclimate. The MiniStat 100 features a maximum power output of 100W for heating elements. It is an on/off toggle system. One wire is used for mains power, one for the sensor, and one for the heating element. It has a temperature range from 19°C – 35°C, a range which would be acceptable for most pet reptiles, some exceptions being Bearded Dragons, Uromastyx, and the Yemen Chameleon. The unit has a heater status LED which lights green when power is being provided to the heating element. The pricing listed on the manufacturer's website is £33.99. There is a MiniStat 300 model available which has a maximum power output of 300W, which is otherwise identical to the 100 model, with a price of £42.99 [29].

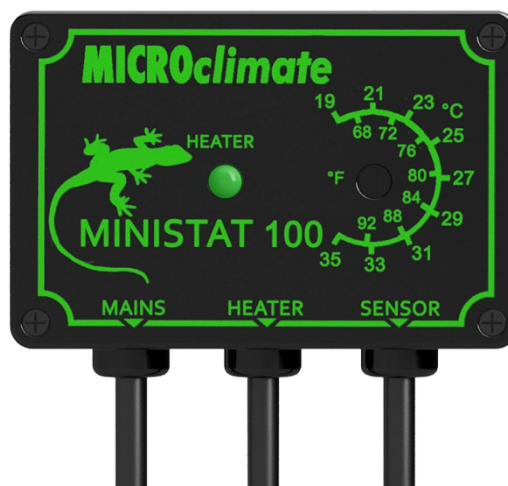


Figure 2.7 - Microclimate MiniStat 100 thermostat [27]

The Pulse B2 HT is a similar product, with two primary differences. Firstly, it has a higher temperature range, from 29°C – 45°C. This allows for care of reptiles requiring hotter temperatures, but is also not suitable for those requiring cooler

temperatures, as it has a minimum temperature of 29°C. The second and more important difference is the dimming functionality. As its name implies, the Pulse B2 HT uses pulse width modulation (PWM) to dim heating elements, allowing for finer control. The drawback of this product, along with many dimming thermostats, is its price. This product's Manufacturer Suggested Retail Price is £58.49. As a single unit, this may be acceptable, though purchasing a large quantity for a pet shop or zoo setting may not be viable.

Microclimate EVO [30]

Compared to the previous Microclimate models, the Evo (shown below in Figure 2.8) has a more advanced feature set, providing:

- A touchscreen
- Maximum load of 1,200W across two heating element channels
- Dimming or on/off control
- 24 hour temperature history graph

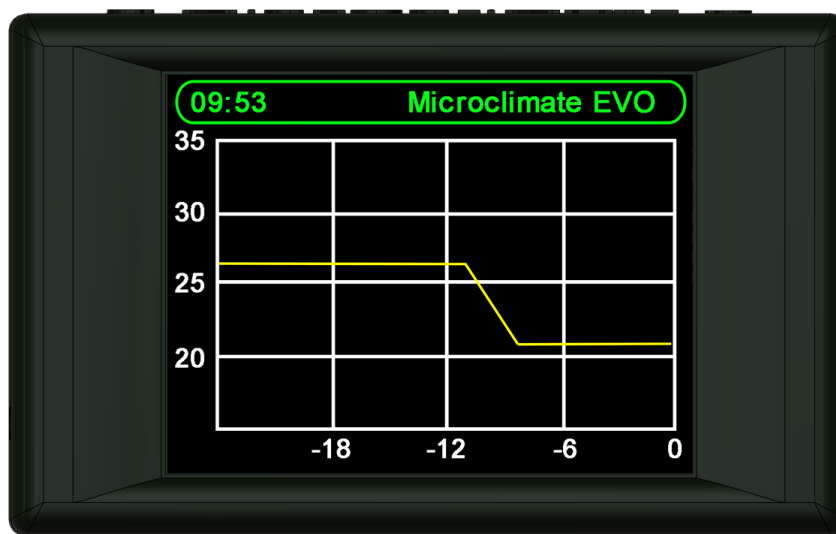


Figure 2.8 - Microclimate EVO thermostat graph [30]

This model is priced at around £120, with an Evo Lite model priced at £94.99 [31]. The advantages of these models are advanced monitoring and control of two heating elements from one device, though at these prices would be unsuitable for use in large quantities.

Komodo On/Off [32] and Pulse [33] Thermostats

The Komodo thermostats have similar functions to previously discussed models. Komodo sell both on/off and dimming thermostats, up to 600W power delivery, and with a claimed temperature range of no more than $\pm 1^{\circ}\text{C}$ from their set target. The least expensive model is the on/off 100W model, priced at £29.99. The disadvantage of on/off thermostats is a higher temperature range due to the lack of fine control. The Pulse model provides dimming, with the higher power, 600W model (shown below in Figure 2.9) being priced at £49.99. As these thermostats are similar to many others on the market, their advantage is that they are comparatively inexpensive.

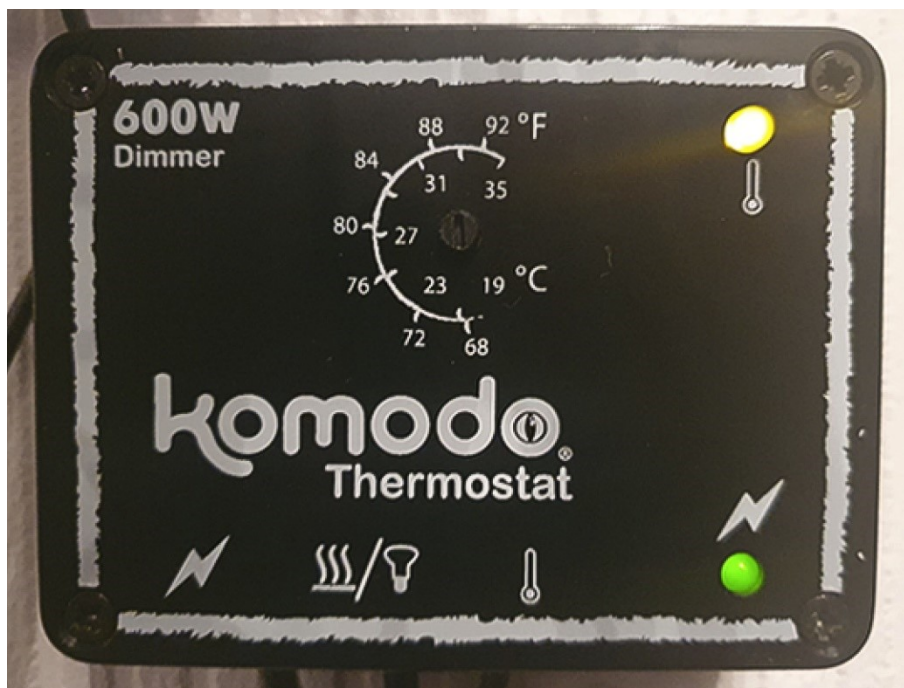


Figure 2.9 - Komodo Pulse 600W thermostat - personal unit

ZooMed ReptiRain [34]

The ReptiRain (Figure 2.10) is one of the most popular misting systems for reptiles. Its features include spray interval settings of every ½ hour, 3, 6, or 12 hours, and has spray durations of 15, 30, 45 or 60 seconds. It includes sprayer nozzles and tubes, and a mains power adapter.



Figure 2.10 – ReptiRain sprayer - personal unit

The main disadvantage of this unit is that it does not sense the current humidity, and sprays only based on a timer. This means that even if the enclosure is already humid enough, it will continue to spray, regardless of exceeding the threshold. The unit is priced at approximately £75 - £85, which would be far too expensive to use in large quantities.

Other manufacturers with similar products include Habistat, ExoTerra, and MistKing.

2.3. Informal Interview

An informal interview was conducted with a pet shop in Swansea in order to determine difficulties they encounter and gain more context. Below the primary findings of this interview are detailed:

- Although automated consumer products are available, they are too expensive to purchase for an entire shop. Although the shop sold many of these units, they did not use them for their own reptiles and instead took a manual approach.
- Manual care involved periodic spraying to control humidity, ultraviolet lamps on wall socket timers, and manual control of heating lamps which were constantly on throughout the day. Ambient temperature was mostly controlled by swapping out heat bulbs for ones with varying wattages throughout the year, to reflect the change in ambient temperature
- The staff noted that this has no issues with most species, but some are more sensitive to temperature and so need more regular checking
- They also discussed the impact this has on the shop as a whole – it takes away from time which could be used elsewhere. This is especially true as this pet shop was run by a small team of only a few people

2.4. Research Conclusion

This chapter has reviewed and critically analysed papers relevant to this project, existing commercial systems, and detailed an informal interview. The findings of the work in this section will influence the design of the Major Project implementation.

Based on the findings of this section, the following areas will be explored further:

- Power saving strategies for microcontroller devices
- WSN topologies
- Microcontroller, sensor, and actuator considerations – Arduino or similar microcontroller boards are most suitable due to their low power consumption, cost, and form factor. Consideration of hardware devices will be essential for reducing the cost of the system – essential for a pet shop environment where the device may be used in large quantities.

- Software and programming language considerations. PHP is a suitable candidate for a web server interfacing with a MySQL database.

The following section discusses the research and development methodologies to be used in this project.

3. Research and Development Methodologies

This section details the research and development methodologies with justifications. The research methodology subsection discusses the research philosophy and approach, research choices and time horizons, and data collection and analysis. The development methodology subsection discusses the development approach and methodology chosen, along with hardware and software choices.

3.1. Research Methodology

3.1.1. Research Philosophy

This project will take a mixed-methods, primarily positivistic approach, combined with elements of phenomenological approaches. A positivistic approach is the most beneficial as the project involves a significant development and testing aspect, which will rely on gathering quantitative, precise data samples. However, qualitative data will be gathered and analysed in order to aid development and evaluation, so a phenomenological aspect is present.

Phenomenology is based on perceptions and ideas, allowing for data to be generalised between settings, and producing data which is valid as opposed to reliable [35].

Positivism as a primary philosophy focuses on observations and quantitative data, providing low validity but high reliability [35]. A positivistic approach uses artificial environments, which will be useful for testing, in combination with the phenomenological approach.

3.1.2. Research Approach

The project will take a deductive approach. Deductive research emphasises scientific principles, linking theories and data, collecting quantitative data, and a highly structural approach, among other benefits. This is beneficial for the project as it will require quantitative data as part of testing, with the aim of proving a theory using data. Deductive research also allows for generalisations based on data, which is

important as it allows for interpretation of data which can be applied to multiple scenarios.

3.1.3. Research Strategies

This project uses a variety of research strategies.

Unstructured interviews will be carried out with staff at a local pet shop, one before and one after the development. These will be used to guide the development and evaluate the efficacy of the system. Interviews also provide some context for the project to aid in the requirements development.

A literature review has been conducted, investigating current IoT developments. This included a range of academic, governmental, and industrial sources.

Software development will be used to create the proposed system. This will be influenced by the above mentioned research in both the development, and evaluation components.

3.1.4. Data Collection and Analysis

This project will take a cross-sectional approach as it focuses more on multiple data points from one point in time. This applies to both the quantitative and qualitative data gathered.

The quantitative data will be primarily numerical and continuous data – temperature and humidity over time from functional testing. The qualitative data will be in the form of interviews. Data will also be gathered in user experience testing, to benchmark the usability of the web application. These will involve System Usability Scale questions and tasks which allow for observation of the participants. Observation was chosen over video recording or screenshots as they would provide large quantities of data with little value, whereas observation with notes and a questionnaire provided data which was simpler to analyse.

3.2. Development Methodology

The development methodology chosen for this project is Feature-Driven Development (FDD). FDD is process-oriented, and client-centric [36], [37]. As is typical of agile methodologies, it has an incremental approach. This is particularly important given the continuous learning and improvement during Higher Education, allowing for iterations as knowledge grows.

The five core phases of FDD [36], [38] are listed below:

- Develop and overall model
- Build a feature list
- Plan by feature
- Design by feature
- Build by feature

As this project is separated into hardware development and website development, being able to plan, design and build by feature is beneficial, as it allows for smaller steps to be made towards a larger goal. FDD is typically used in software teams [39], however, there is no reason it could not be used by individuals, especially given its highly structured nature. It considers features from the view point of the user or client, not from their implementation [40], which helps develop useful features.

Individual developer approaches to Behavioural or Test Driven Development were also considered, however, FDD was chosen due to its highly sectioned, compartmentalised approach.

3.3. Project Risks

This project has risks associated with it, both those applicable to academic projects generally, and to this specific project. The primary risks are detailed on the next page in Table 3.1, along with risk exposure values.

Table 3.1 - Project risks, including risk exposure calculations

Risk		Severity (0 - 4)	Probability (0 - 1)	Risk Exposure Value (0 - 4)	Risk Prevention and Monitoring	If the risk occurs
R1	Missing supervisor meetings (e.g. absence of student or supervisor)	2	0.2	0.4	Keep in regular contact with supervisor, keep log of project meetings	Review with supervisor, move to remote meetings if necessary
R2	Funding difficulties regarding acquiring hardware for the deliverable	3	0.4	1.2	Discuss with supervisor in advance	Move to simulation based project or modify project to require less expensive hardware if necessary
R3	Prolonged power outage at home, preventing working on a computer or developing the project deliverable	4	0.1	0.4	N/A	Continue with project on University campus, e.g. computer labs or open access areas
R4	Hardware components' delivery delayed or hardware faulty	3	0.2	0.6	Order from reliable suppliers	Return faulty hardware and notify supervisor of delays caused
R5	Student illness or absence from University	3	0.5	1.5	Update supervisor and other University staff when necessary	Review with supervisor, apply for Extenuating Circumstances if necessary
R6	University closure (e.g. adverse weather)	2	0.1	0.2	Monitor University communications to keep up to date	Move to remote meetings if necessary
R7	Loss of data (e.g. drive failure, accidental deletion)	4	0.1	0.4	Make regular backups of work both locally and in cloud storage	Return to most recent backup and accept the data loss. Allocate additional time to the project to catch up
R8	Sections of project overrun original timeframe, not finishing in time or the deliverable not functioning	3	0.5	1.5	Monitor progress continually and review against aims and objectives	Review with supervisor if appropriate, allocate additional time to project to catch up
R9	Requirements inflation - as the project progresses, more requirements emerge, threatening project completion	2	0.6	1.2	Plan thoroughly at beginning of project. Monitor progress continually and review against aims and objectives	If additional requirements outside of scope or time available, discuss with supervisor

The primary measures for preventing, monitoring, and dealing with risks are to continually review progress, communicate regularly with the project supervisor about any changes or difficulties, and to plan ahead of time. The risks are displayed below in Figure 3.1 in a risk matrix:

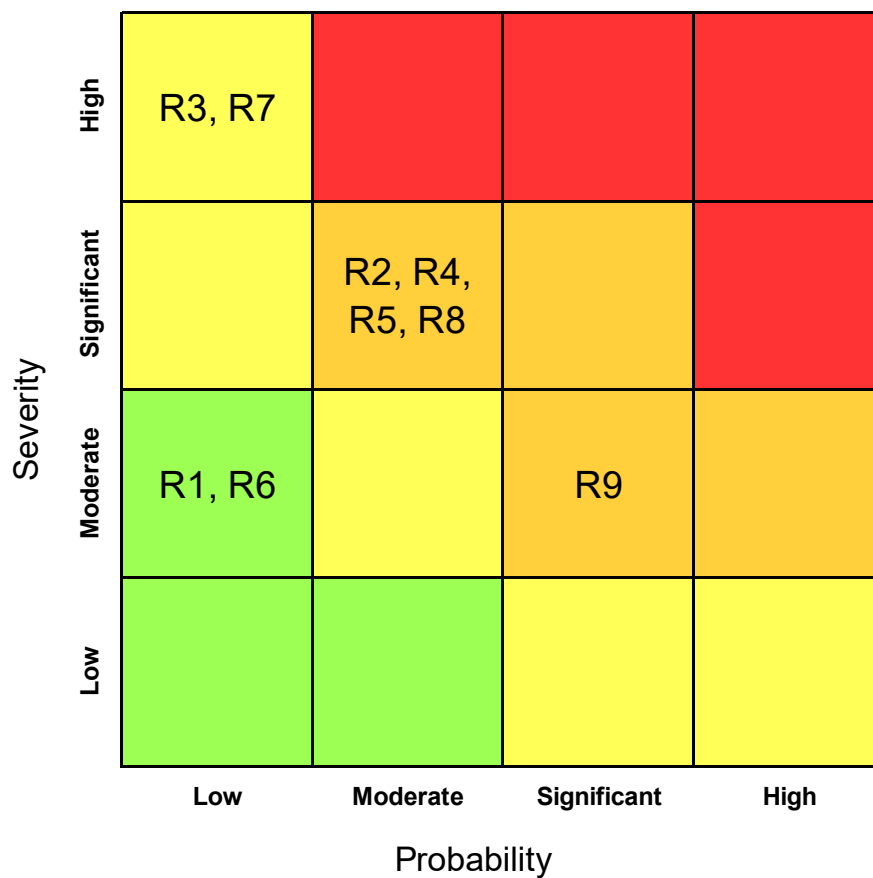


Figure 3.1 - Risk matrix

3.4. Hardware and Software Resources

This project will require a range of hardware and software resources. These are detailed and justified in full in Appendix B, and summarised in Table 3.2 below:

Table 3.2 - Hardware and Software resources

Hardware	Software / Languages
A Raspberry Pi 3B, for use as a server	The Arduino programming language for programming node devices
One Wemos D1 Mini development board per node device – total of 3 nodes for testing	PHP for back-end web development, due to its range of included libraries and functionality
Two DHT22 (AM2302) per node device, for sensing temperature and humidity	JavaScript for providing AJAX functionality
One DS3231 Real-time clock for reading time, per node	SVG for vector graphics in the web application
Four relays per node	HTML and CSS for the web front-end
Actuators – per node: • Heating lamp • Heat pad • Sprayer • Ultraviolet lamp	SQL for interfacing with a MySQL database

Node devices will be based on the Wemos D1 Mini [41], and programmed using the Arduino language – which is “merely a set of C/C++ functions” [42]. The Arduino code will use two custom libraries [43], [44] to read sensors, along with standard Arduino libraries such as SPI, Wire and math. The DHT22 sensor was chosen due to its relatively high accuracy [45] and low price.

Web development will use PHP and JavaScript, using jQuery [46] for AJAX functionality. JSON will be used as it is easily parsed by JavaScript [47], and MySQL will be used as it is well integrated into PHP, for example using the PDO class [48].

3.5. Research and Development Methodologies Conclusion

This section has detailed research and development methodologies and approaches used by the project, explained the project's risks, and listed resources required for the project. The next section details the proposed system's design for both hardware and software components.

4. System Design and Requirements

This section details the design for both hardware and software components of the deliverable.

The system will consist of a WSN connecting many sensor nodes to a central server.

Each node device will be responsible for monitoring and controlling the conditions for a single reptile terrarium. Each node device will send and receive data in communication with a central server which hosts the web site and database.

The node devices will request their environmental parameters from the server, providing the server with an ID. The server then searches its database for the given ID and communicates the parameters of this ID back to the node device. The nodes then control actuators when their sensor readings exit the parameter ranges. Node devices will communicate their most recent sensor readings and actuator states back to the server to be inserted into the database.

Ideally, the network would be implemented using Zigbee or Bluetooth. However, Zigbee connectivity would be expensive and difficult to implement, and Bluetooth cannot support a large number of devices connected at once.

The network uses a star topology, though if expanded, a tree or mesh topology may be beneficial. On the next page in Figure 4.1 is a visualisation of the network, created in Cisco Packet Tracer:

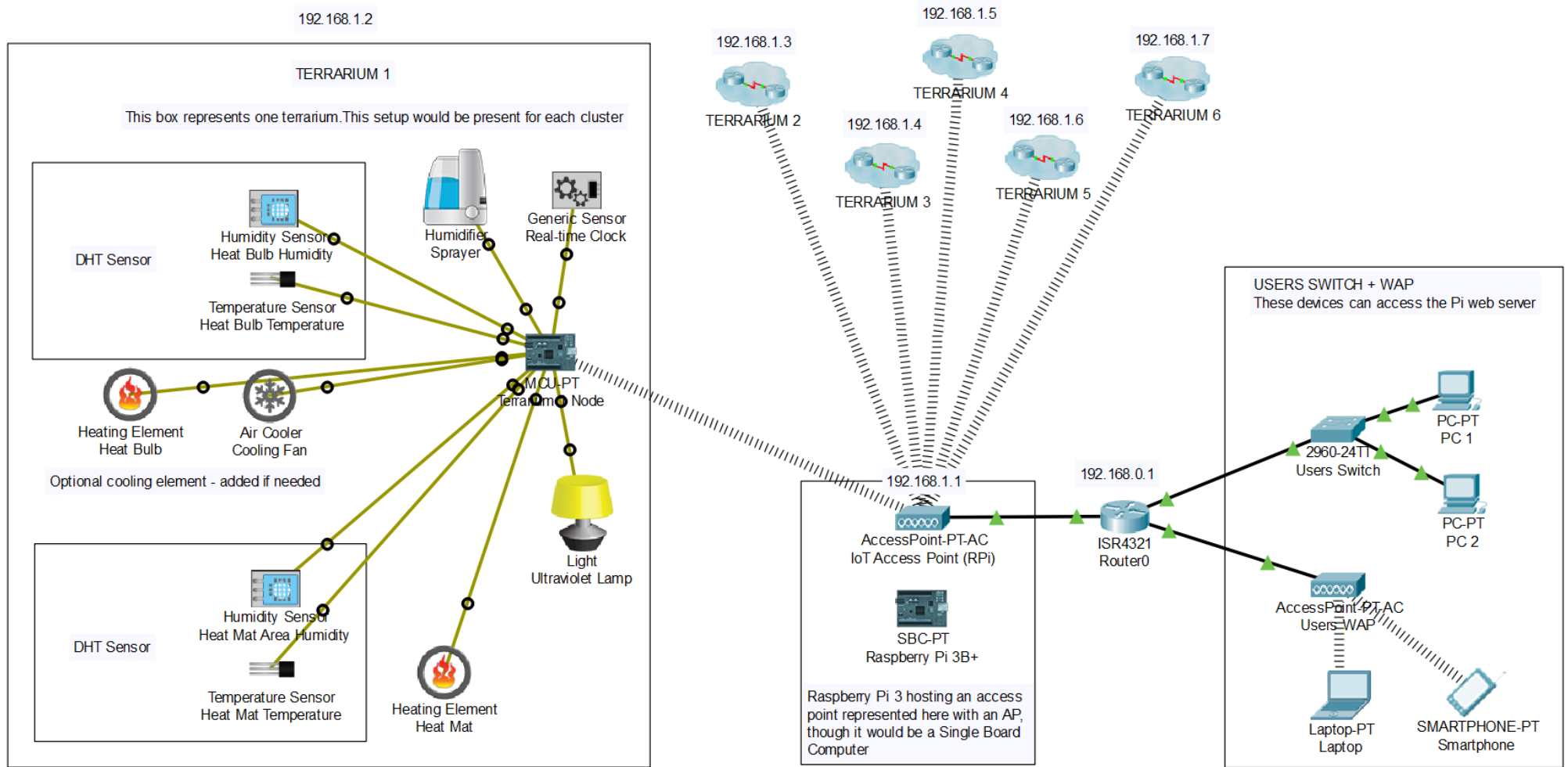


Figure 4.1 – Network design

4.1. Requirements

Based on the research conducted, primarily the interview, the below requirements have been developed:

- Environment monitoring functionality – temperature and humidity
 - Wireless communication between nodes and server
 - Data display on a web application
 - Push notifications for alerts

- Environment control functionality – temperature, humidity, and lighting
 - Relay control to power or un-power devices
 - Aspiring goals: $\pm 1^{\circ}\text{C}$ temperature target, $\pm 10\%$ humidity
 - Heat mat control
 - Heat lamp control
 - Sprayer control
 - Ultraviolet light control

- Most importantly, the system must be automatic and must function unsupervised

4.2. Raspberry Pi Server

The Raspberry Pi 3B+ server achieves three pieces of functionality:

- A PHP and JavaScript web application to allow users to interact with the system, and for the node devices to communicate to the server over HTTP requests, with dedicated PHP pages for handling them.
- A MySQL database server to store and organise the data
- It acts as an access point for the node devices to connect to

A hierarchical visualisation of the server can be seen in Figure 4.2 below:

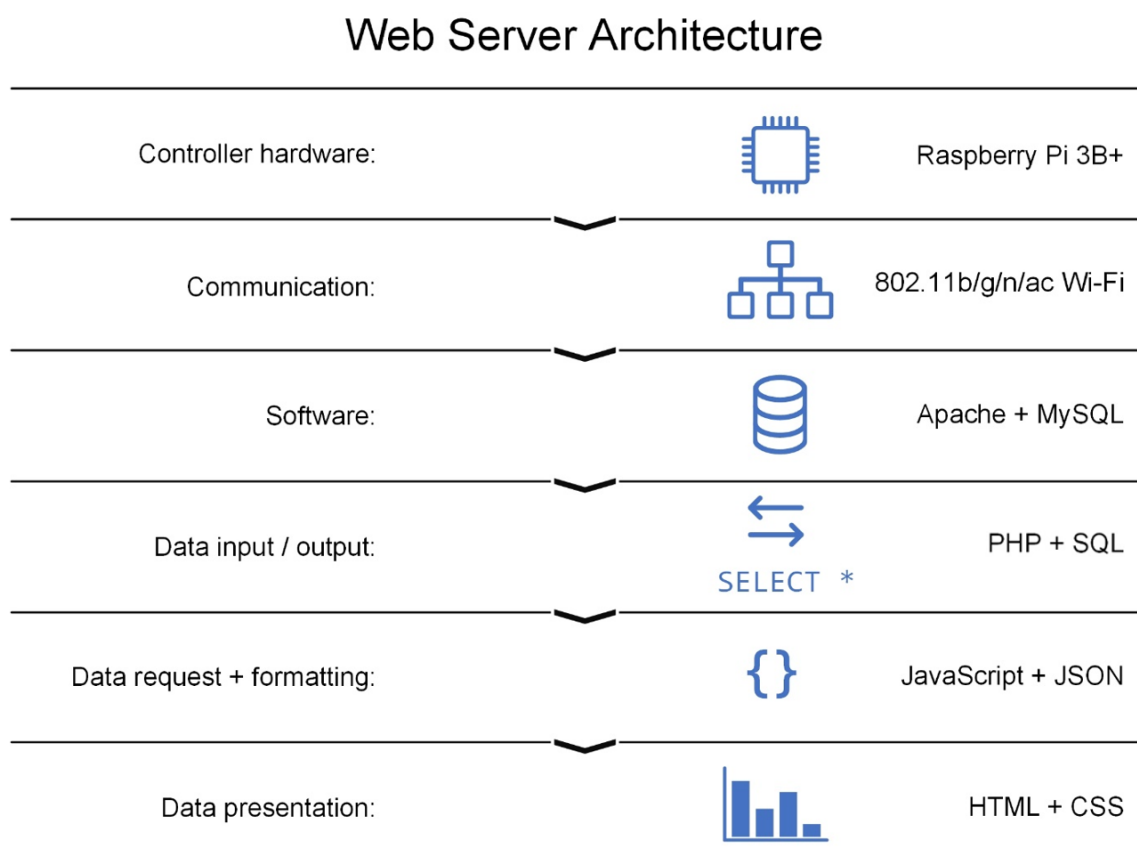


Figure 4.2 - Web server logical architecture

4.2.1.Database Design

A relational database was designed to store data for reptile terrariums and the animals living within them, data logs, and web application users. The following relational database rules have been applied during design:

- Each pet species record may be present in zero or more terrariums, but each terrarium must have zero or one pet species associated with it
- Each terrarium may have zero or more logs, but each log record must be linked to one and only one terrarium

This relational model can be seen below in Figure 4.3:

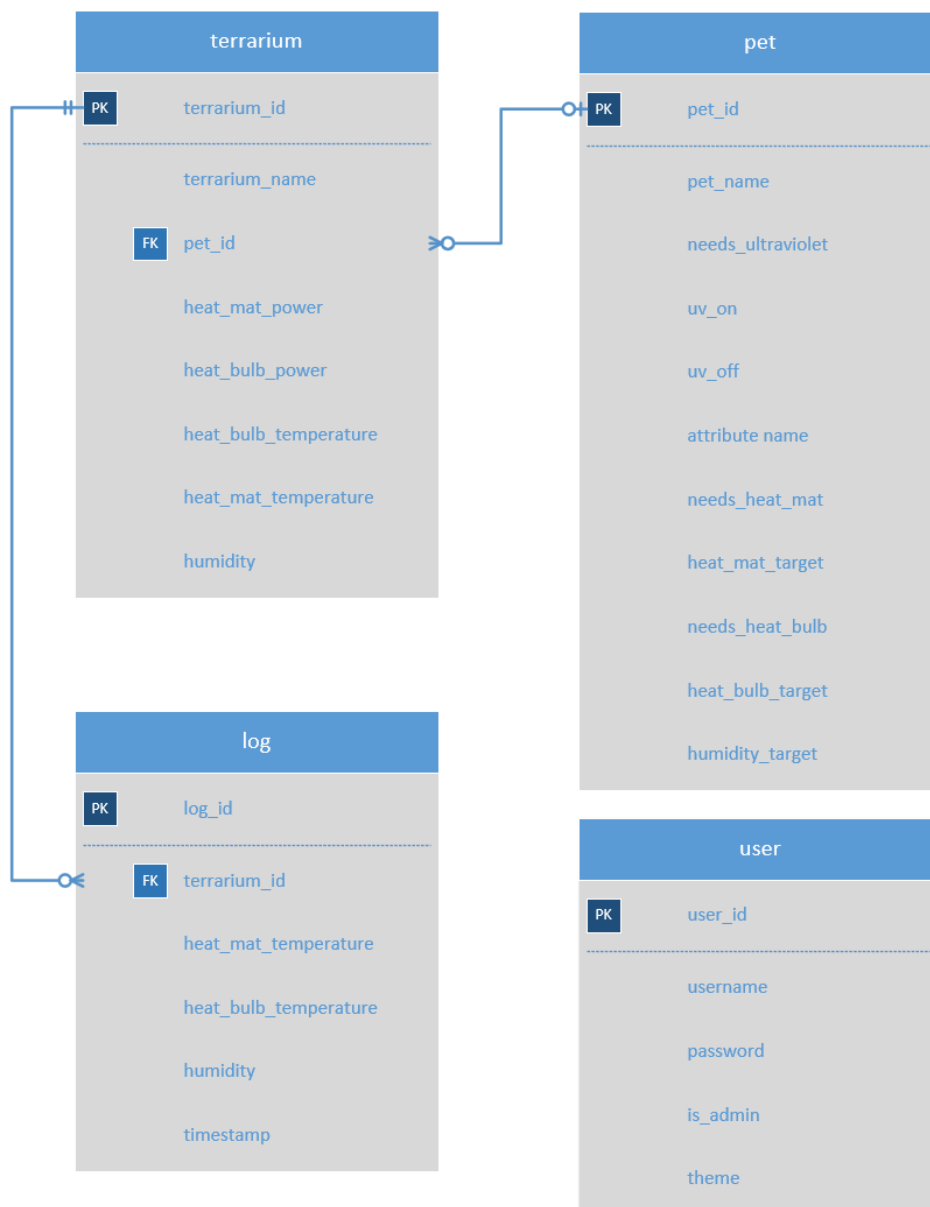


Figure 4.3 - Crow's foot relational database model

4.2.2. Website Design

The site structure consists of two parts: the user-accessible site, and the pages which node devices interact with. This structure is detailed below in Figure 4.4:

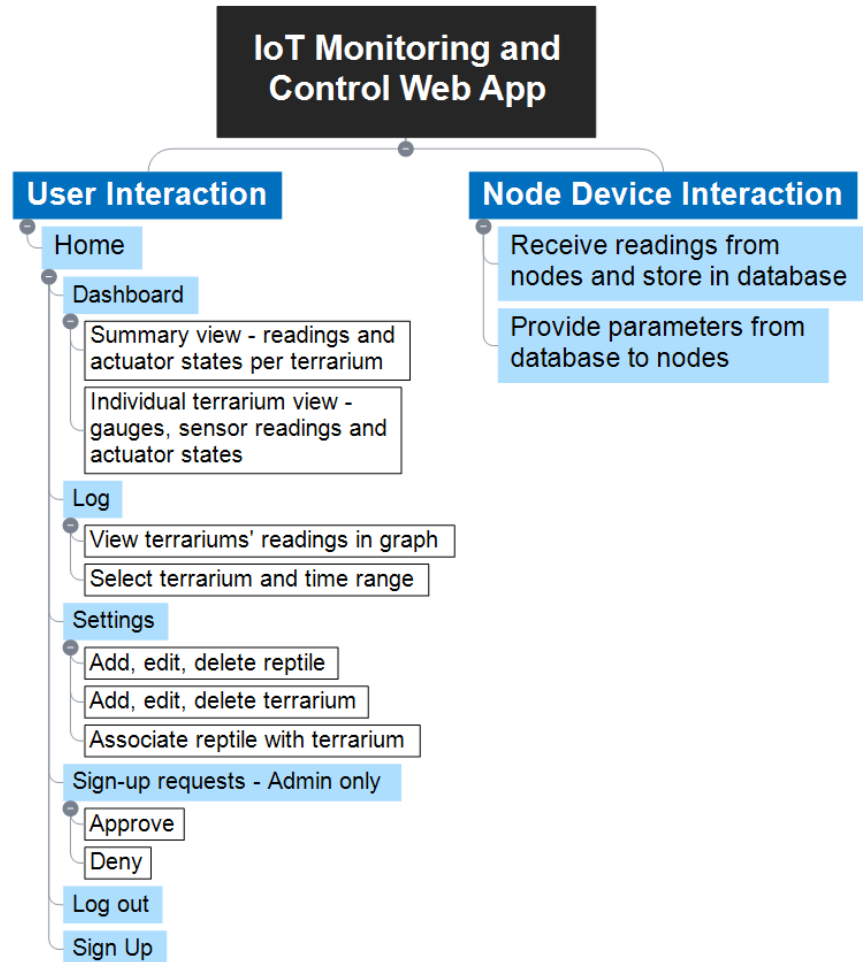


Figure 4.4 - Web application site's structure

The functionality of each page is as follows:

Home

If not logged in, this page will allow the user to log in, and provides a link to sign up. If logged in, it provides links to other pages on the site as icons (Figure 4.5).

Sign up

This page will allow the user to enter a username, email, and password. It verifies that the username is not already in use and that the details meet complexity requirements.

Dashboard

This page has two modes: summary view and individual terrarium view. Summary view will provide details of all terrariums, while individual provides more information about a single terrarium. The user will be able to change between modes by using a dropdown input or clicking a terrarium to view more details. The individual dashboard wireframe is included in Figure 4.6.

Log

This page will display a graph to the user of sensor readings over time for a terrarium, providing an option to download the data as a CSV file. The terrarium is chosen using a dropdown element.

Settings

This page would allow the user to perform CRUD (create, read, update, delete) database operations on reptiles and terrariums. It also allows for linking a reptile to a terrarium. This page is accessible to administrators only.

Sign up Requests

This page would allow for administrators to approve or deny sign up requests. Approving a sign up request allows a user to then log in to their account.

Below are two example wireframes plans for the website. The remainder of the initial wireframes, including mobile designs, are included in Appendix C. It should be noted that the design of the site has changed since these wireframes were created.

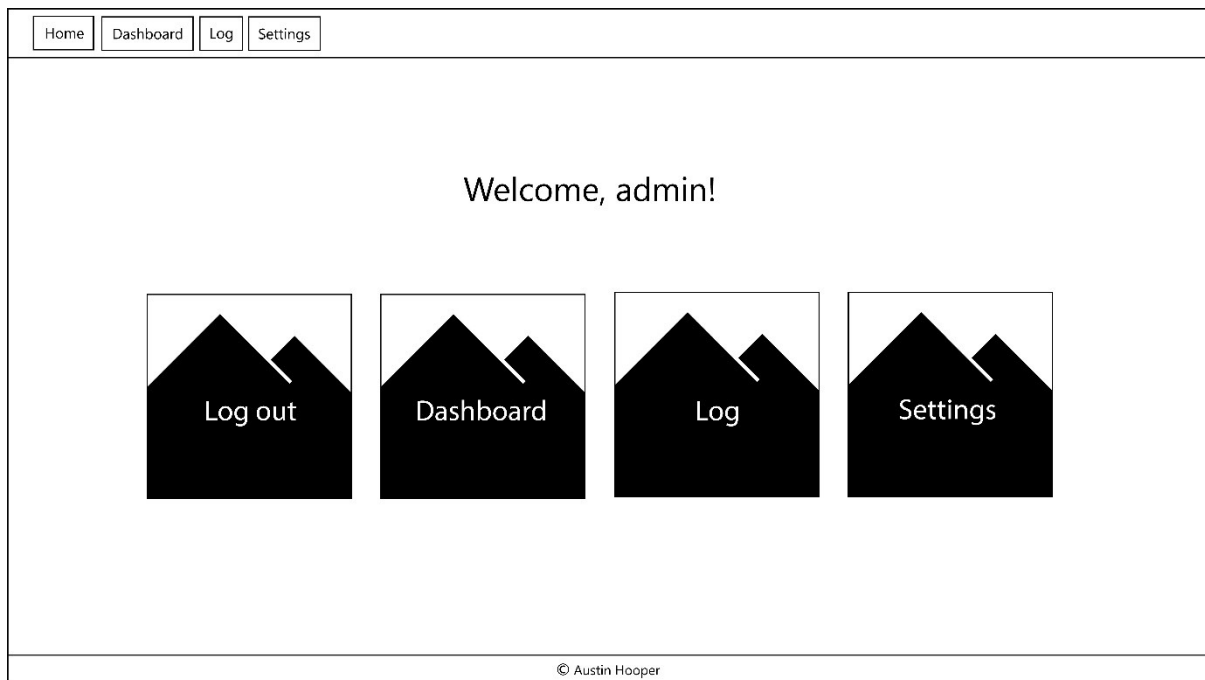


Figure 4.5 - Home page logged in wireframe

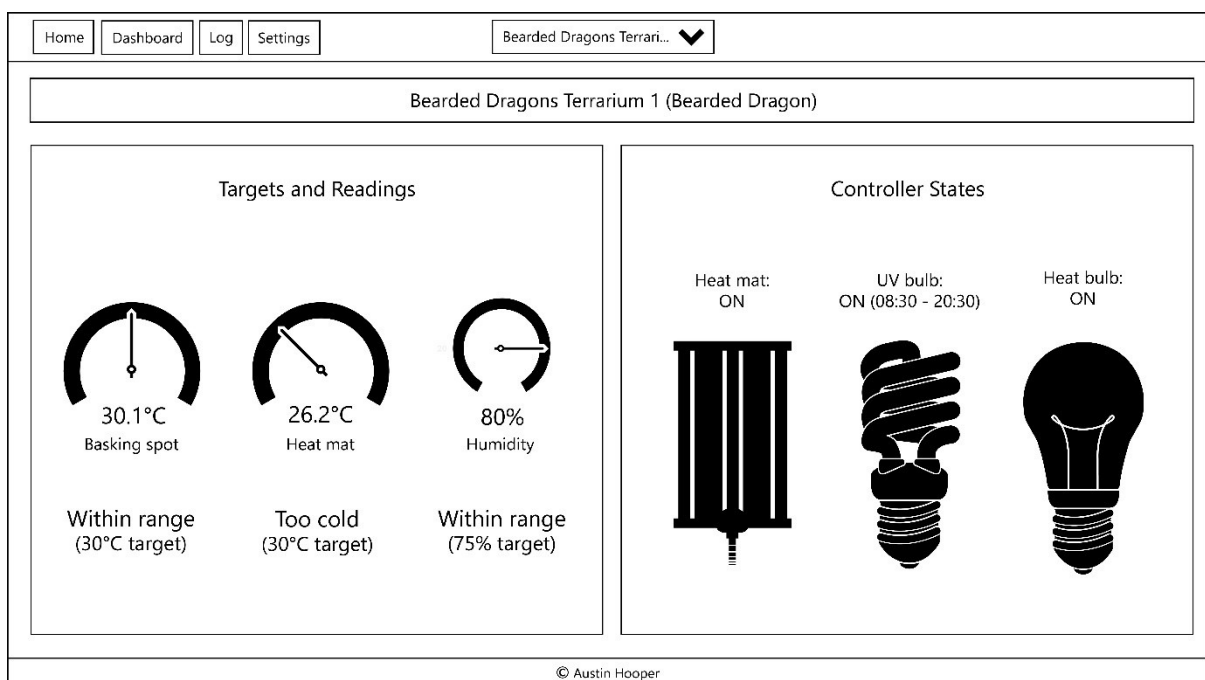


Figure 4.6 - Dashboard individual view wireframe

4.3. Node Devices

The node devices are based on the Wemos D1 Mini – an ESP8266 based microcontroller board. Sensors and actuators are controlled or monitored through General Purpose Input/Output (GPIO) pins, which are programmed in the Arduino language (C and C++ based). The nodes communicate with the central server over Wi-Fi to send and receive data.

A hierarchical visualisation of this can be seen in Figure 4.7 below:

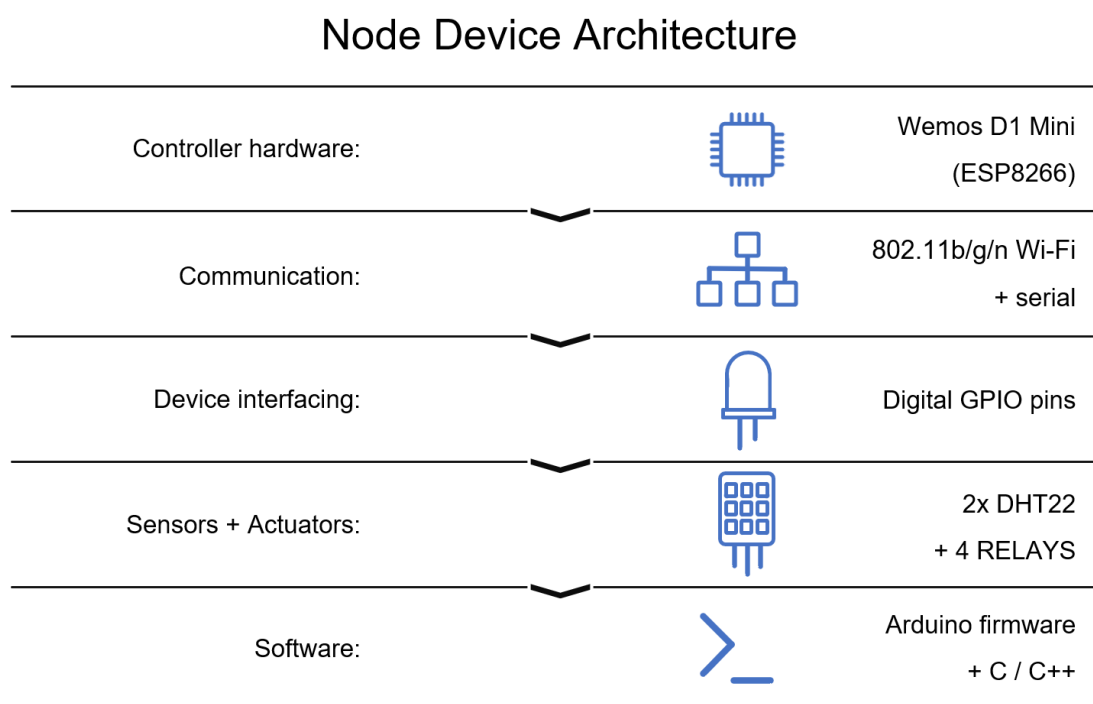


Figure 4.7 - Node device logical architecture

4.3.1. Node Hardware

The node devices' hardware and wiring consist of two temperature and humidity sensors, a real-time clock, and a quad relay board, connected to the microcontroller board. The relays will control a heating lamp, ultraviolet light, a pump acting as a mister, and a heating pad.

The real-time clock has separate power delivery from a USB battery pack as the D1 Mini could not provide enough power for all devices. The real-time clock was also grounded to the D1 Mini as this was required for data communication. This is discussed further in section 5.3.4. The only wiring change since the initial design was the additional power and grounding to the RTC.

See in Figure 4.8 the final wiring diagram:

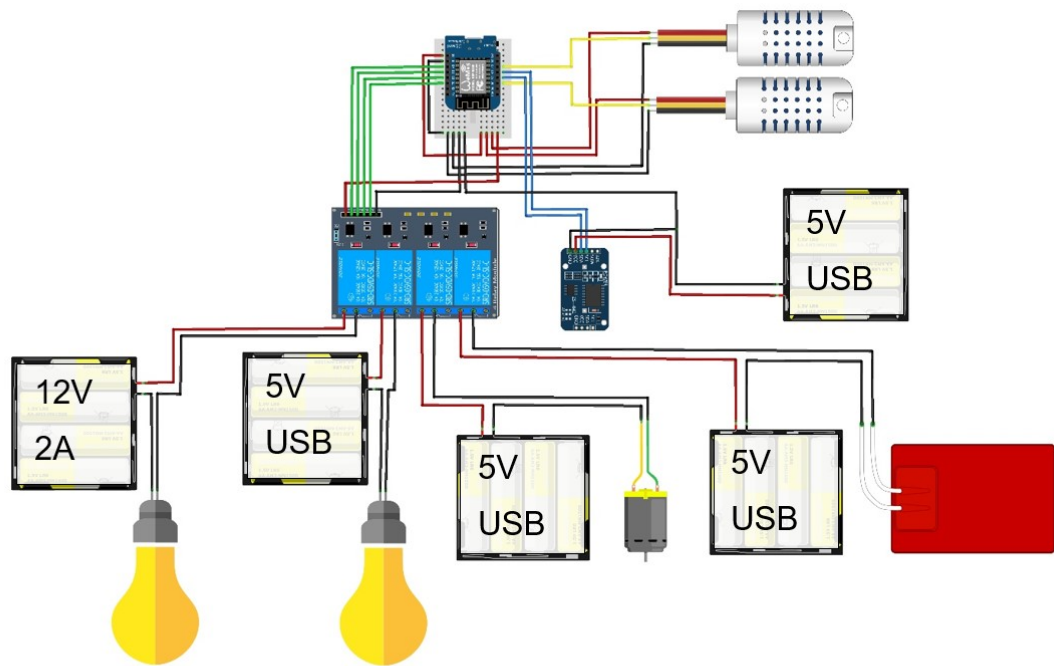


Figure 4.8 - Node device wiring diagram

The four actuators (heating bulb, LED, heating pad and sprayer) are each connected to their respective relays using 12V DC jack plugs, allowing for modularity. Figure 4.9 illustrates the power flow and relay system:

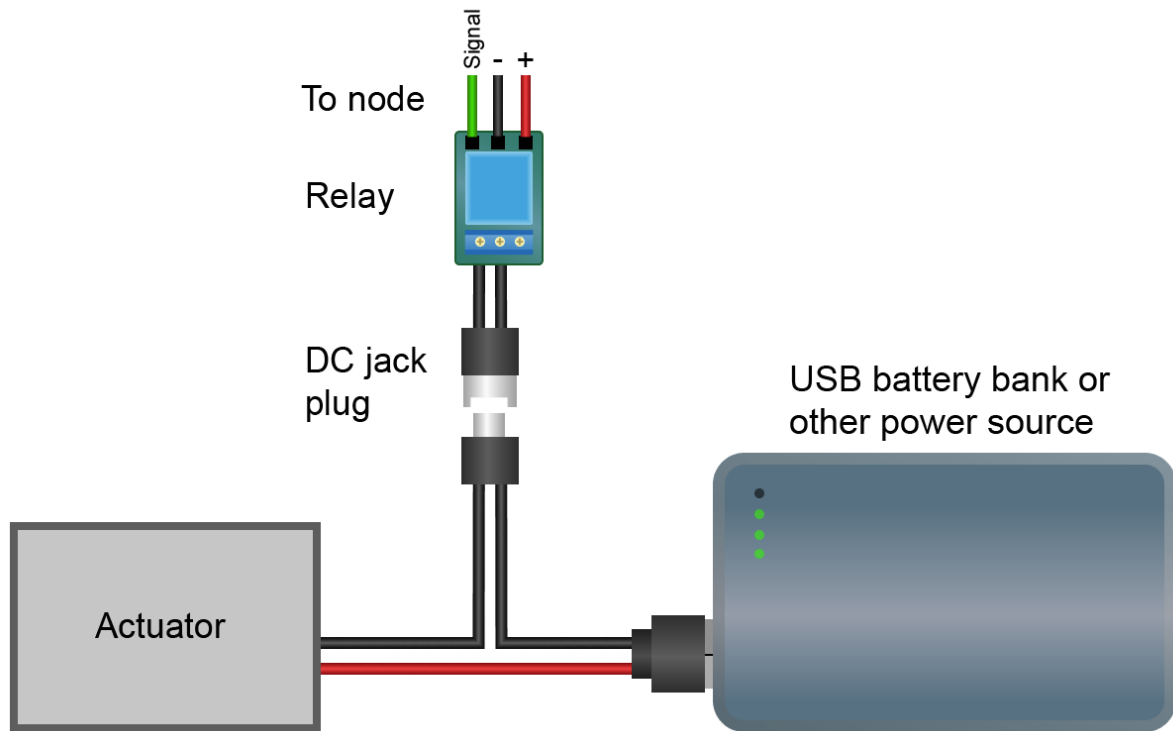


Figure 4.9 – Node device relay configuration

4.3.2. Node Software

This subsection shows detailed flowcharts for the node device (D1 Mini) code.

The node device code primarily uses global variables, some being passed to functions to avoid modifying the original values. The main program flow of the ESP8266 is primarily used to call other functions, as follows in Figure 4.10:

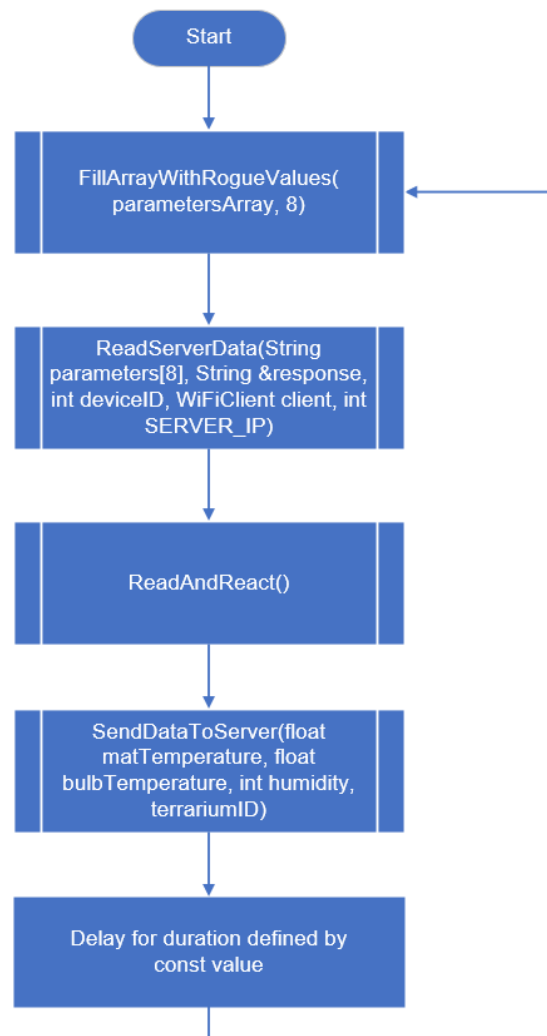


Figure 4.10 - ESP8266 main loop flowchart

This flowchart has no end as it is intended to loop infinitely. The Arduino code replaces the typical `int main() { }` code from C/C++ with a `loop() { }` function.

The `FillArrayWithRogueValues()` subprocess is called to allow the code to later check if server data was read successfully. This subprocess is included in Appendix D.

The first subprocess call within the main loop is to read the data from the server's HTTP response. This function flows as follows:

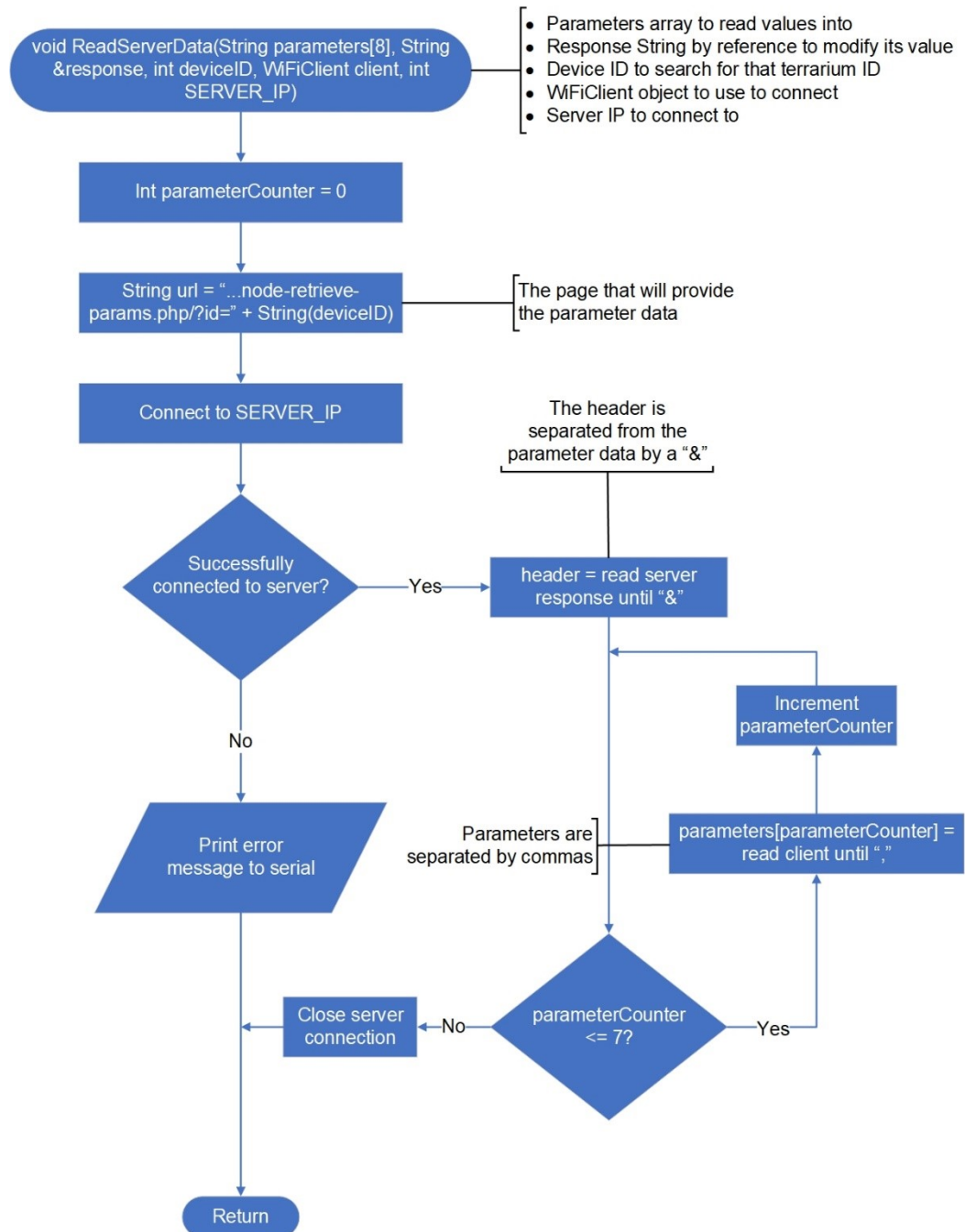


Figure 4.11 – ESP8266 ReadServerData function flowchart

This code connects to a server and adds the terrarium ID to the GET parameters of the URL. The server then responds with data, which is added to an array, looping through each parameter, and exploding the response string by commas. For

example, the server might respond with '1,2,3,4'. Commas split data items up in a way that is easy to detect.

The next function to be called is the ReadAndReact function. It reads sensors, compares their readings to target readings from the server, and powers or un-powers relays as appropriate. This function is the most significant portion of code.

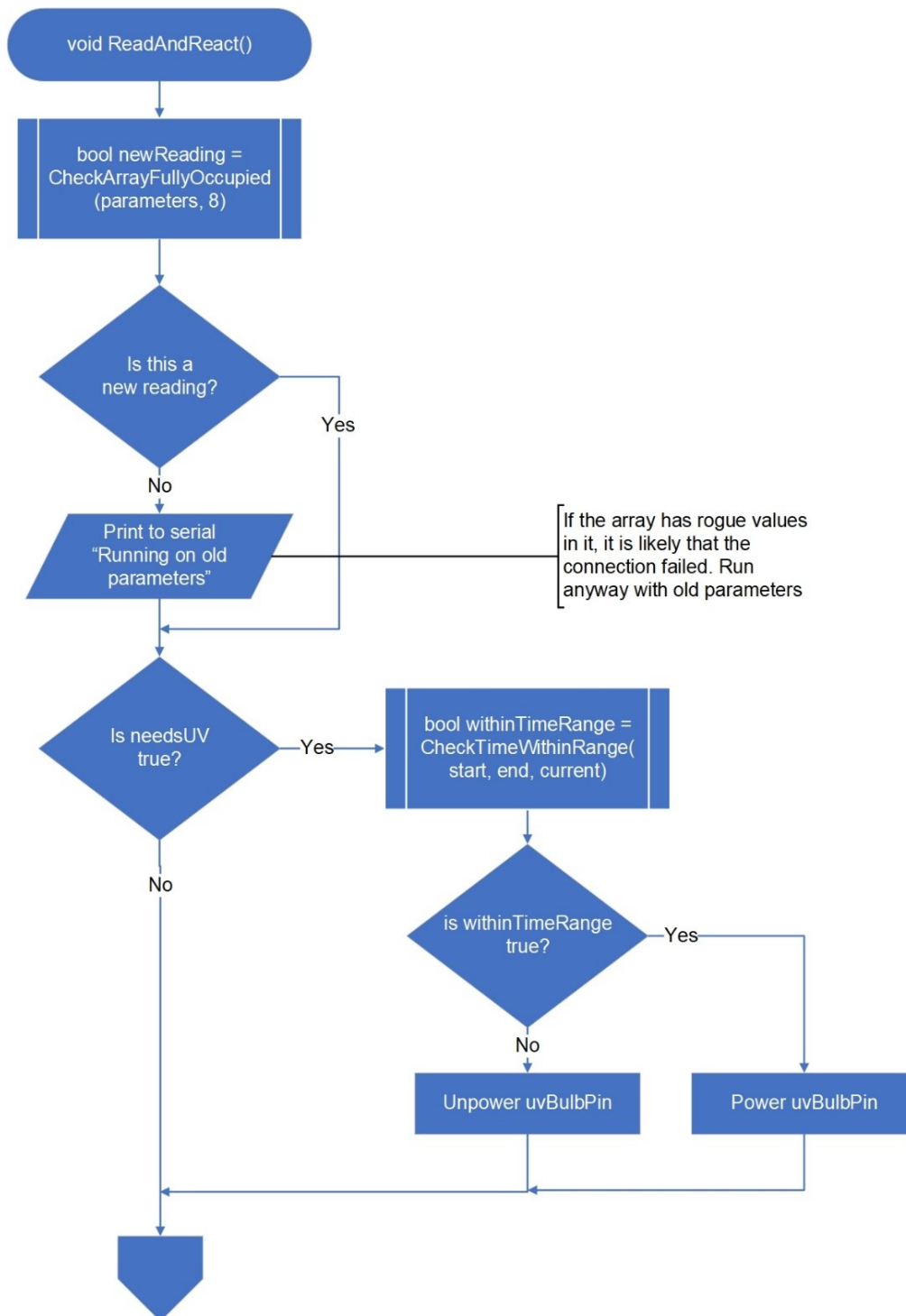


Figure 4.12 - ESP8266 ReadAndReact function flowchart pt. 1

The next part of this subprocess checks if heating elements are required, and if so, reads their associated temperature sensors, powering heaters based on these readings. This is shown below in Figure 4.13. The **ReadSensor** function is detailed in Appendix D.

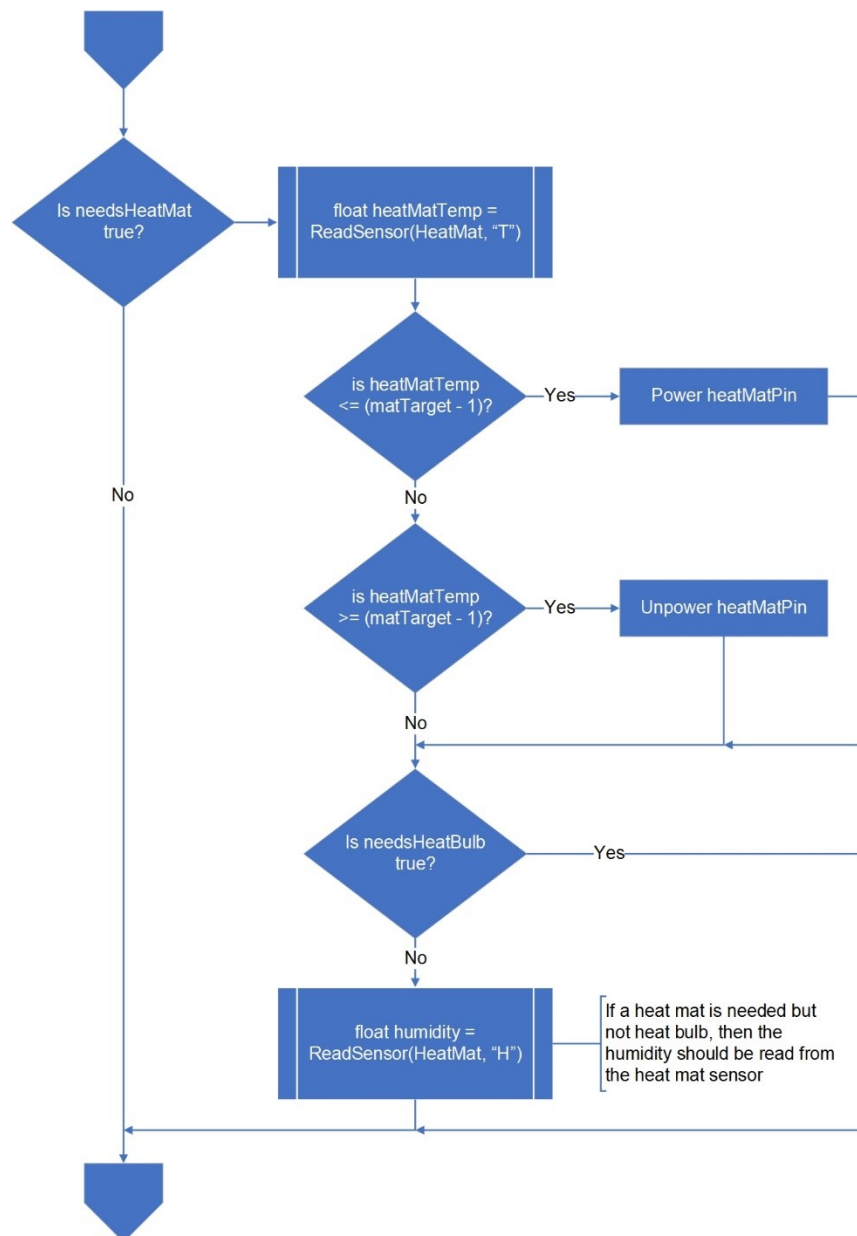


Figure 4.13 - ESP8266 ReadAndReact function flowchart pt. 2

The process is similar for handling the heating bulb for ambient temperature, reading server data and comparing it to sensor readings:

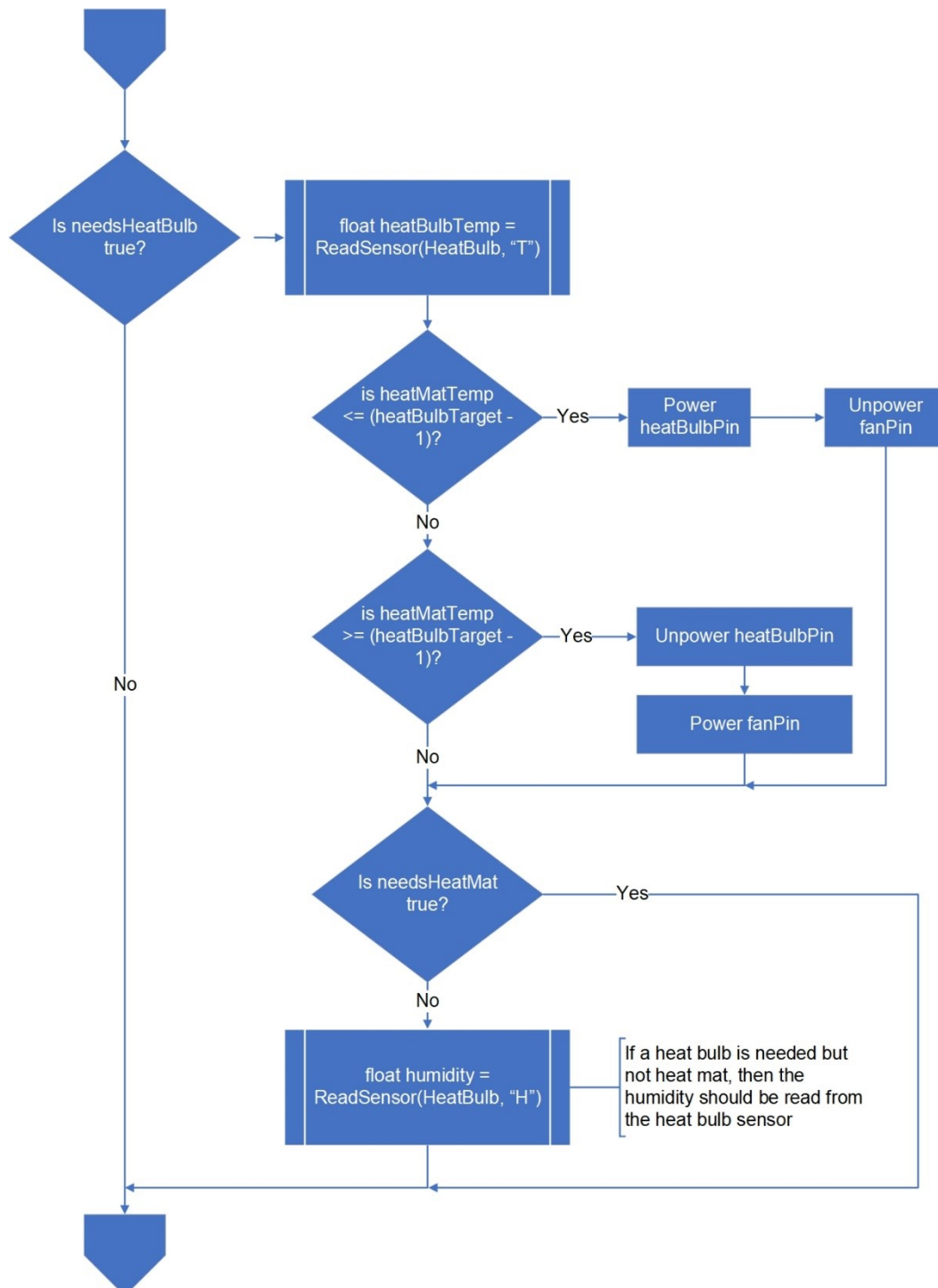


Figure 4.14 - ESP8266 ReadAndReact function flowchart pt. 3

Finally, this subprocess compares humidity readings to the target. If both heaters are required, then an average humidity is obtained from both sensors, otherwise a single reading is taken:

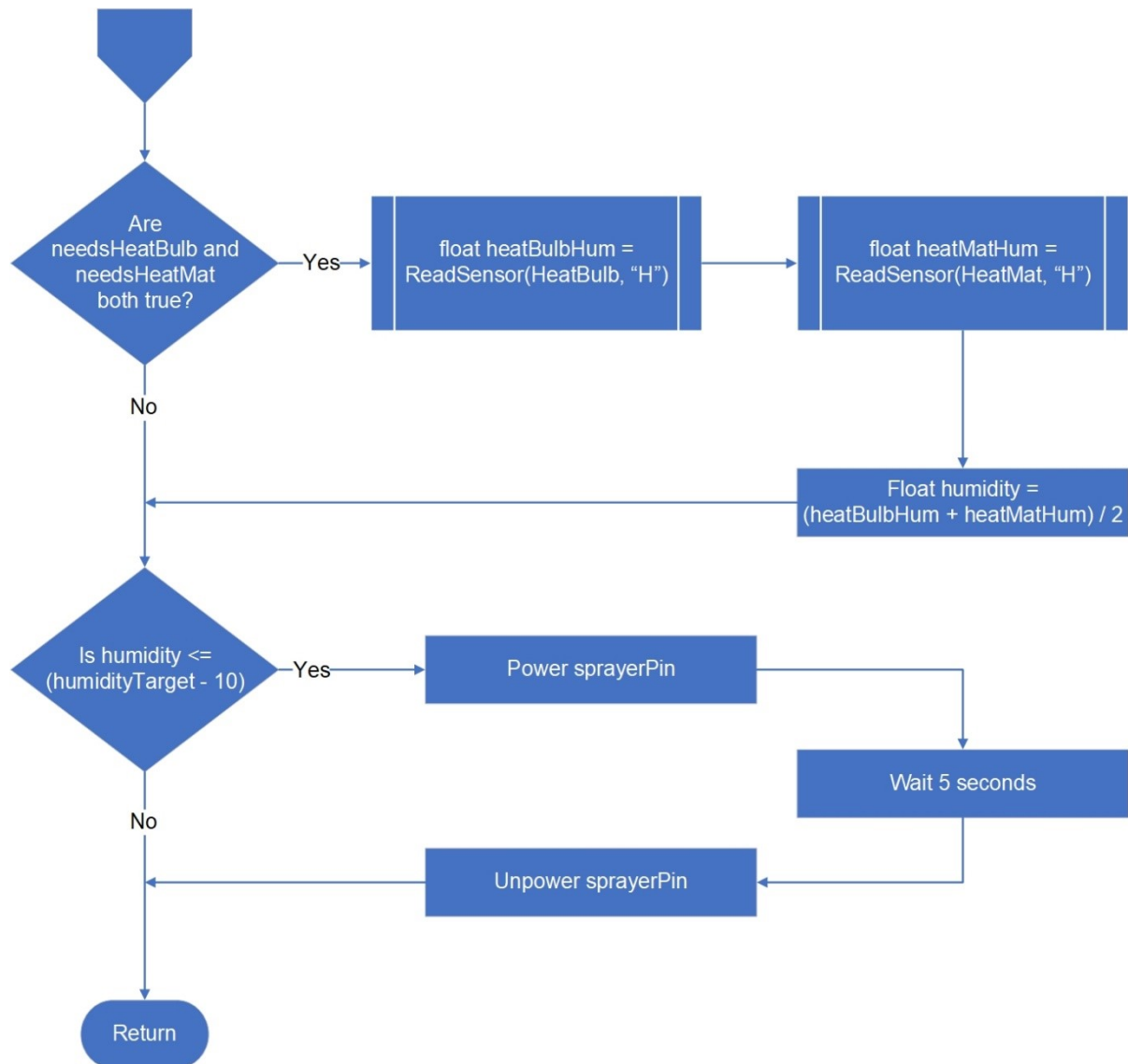


Figure 4.15 - ESP8266 ReadAndReact function flowchart pt. 4

The final subprocess in the main loop sends sensor readings to the web server. This function provides the server's data reading URL, to which GET parameters are appended. These are then read by the server using PHP.

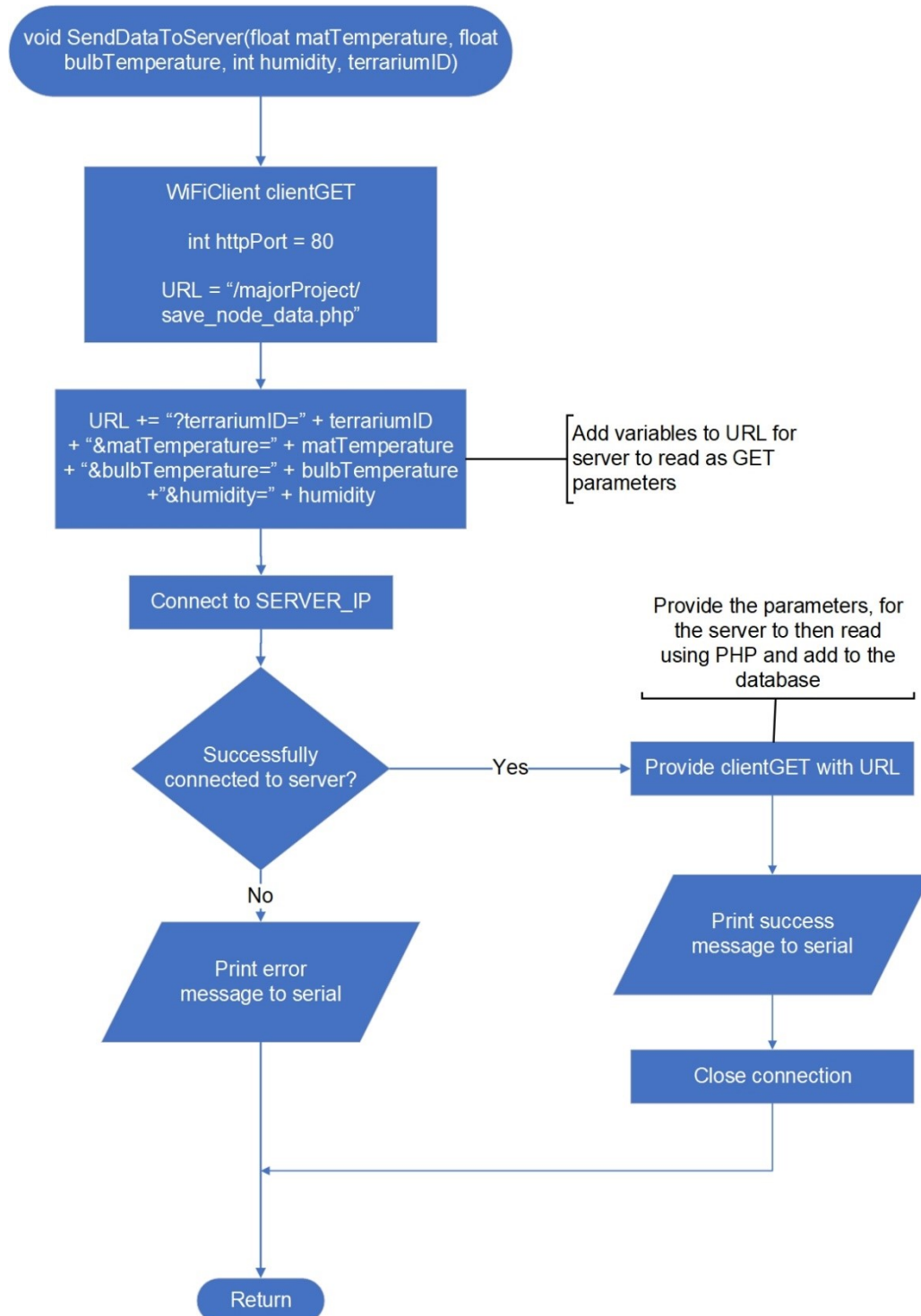


Figure 4.16 - ESP8266 SendDataToServer function flowchart

4.3.3. Node Device Case 3D Modelling

A case for node devices was modelled using Autodesk Inventor, and 3D printed for one node device. A render of the case is shown below in Figure 4.17, showing cut-outs for power, relay connections and sensor wires, and showing the relay board and microcontroller board in place. More details, including dimensions, are shown in Figure 4.18.

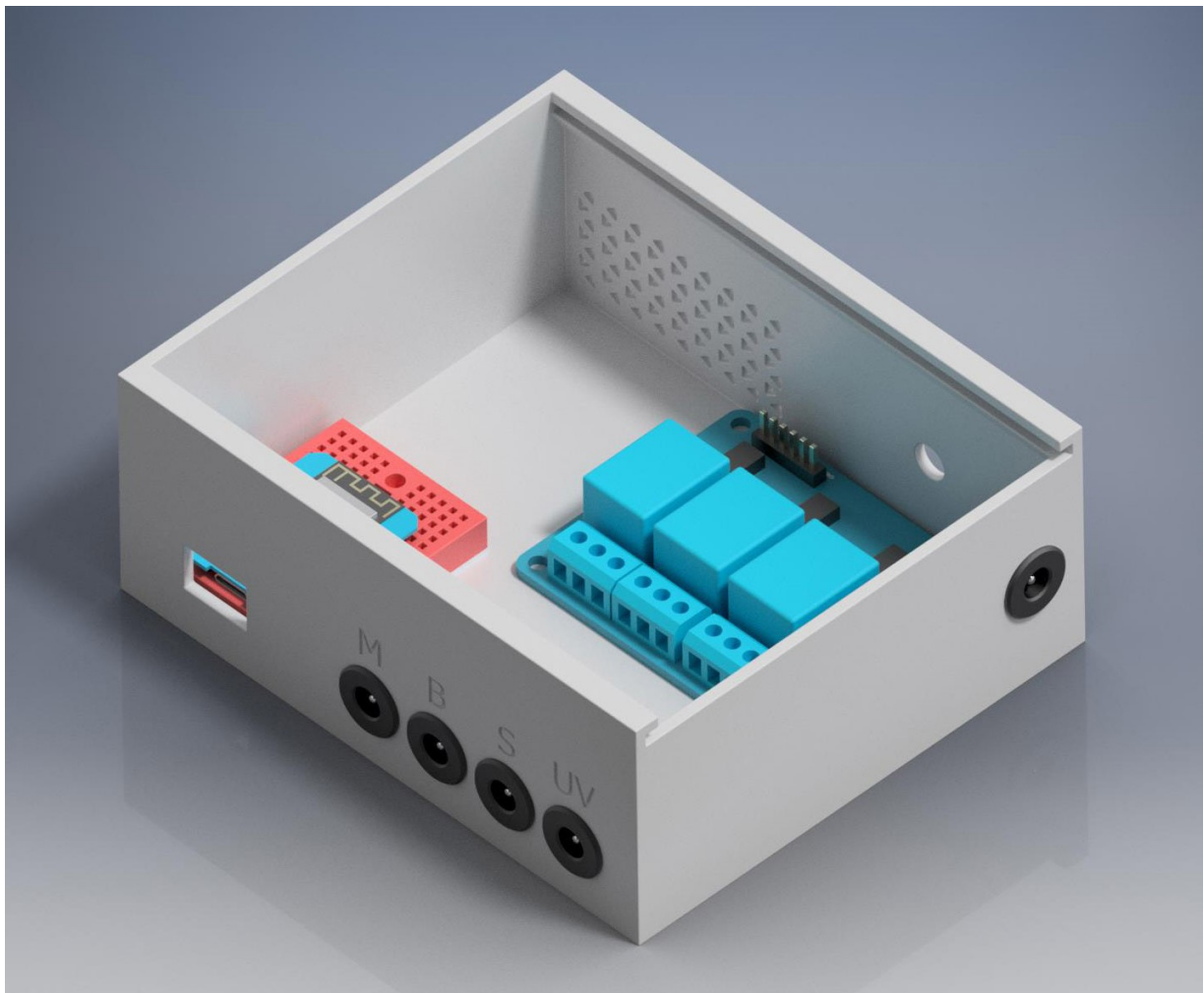


Figure 4.17 - Node device case render

4.4. System Design and Requirements Conclusion

This section has detailed the design of software and hardware components of the project. In the following section, these concepts are further developed and their implementations are described.

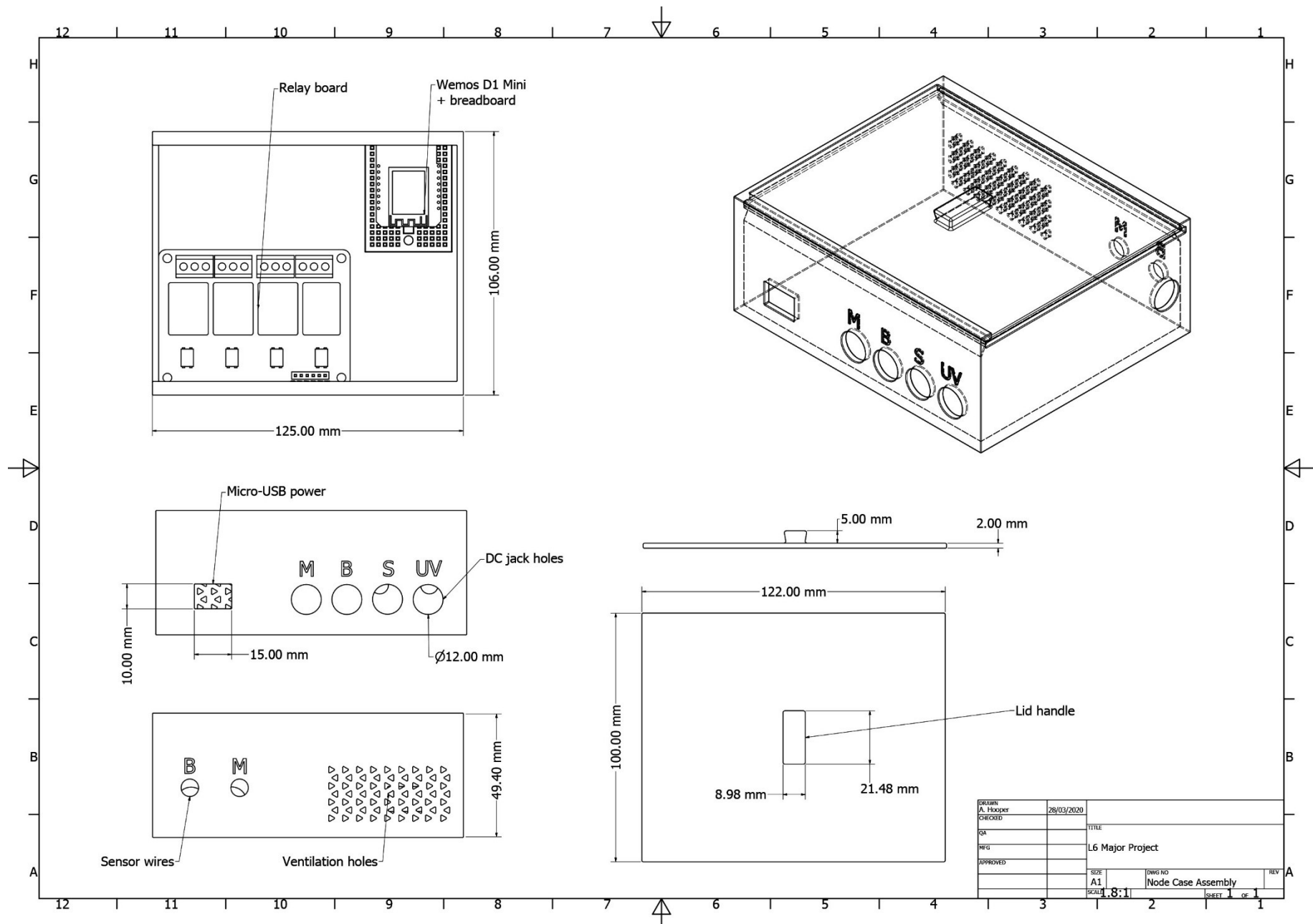


Figure 4.18 - CAD drawing of final node case

5. System Development

This section discusses the system's hardware and software development, along with the difficulties encountered. Examples and screenshots are provided for interesting code.

5.1. Web Server Development

The web server was implemented using PHP and the jQuery JavaScript library. All pages interact with a custom database object, which is a wrapper for the already existing PDO class. PDO prepared statements were used to protect against SQL injection attacks. No MVC framework was used due to the extremely custom nature of the site – though this may be useful if the application were expanded to care for other animals in the future. A custom CSS framework was implemented based on previous work, implementing a row and column system. This was done both as a learning opportunity and to allow for more customisation.

5.1.1. Register and Log in

To register an account with the site, the user must go through the usual registration process, and an admin must approve of the request to sign up. This decision was made because only users with permission should be able to use the site. Only administrators may access the settings page to change reptile care parameters. In the provided SQL file, two accounts are set up:

Administrator

Username: admin

Password: admin

Normal user

Username: test

Password: test

5.1.2. Sending Data to Nodes

When a node requests parameters, it provides an ID for a terrarium as a GET parameter. The server then searches the database for parameters linked with this terrarium, and outputs them, separated by commas. This is then sent as a HTTP response, which is read by the node and the string exploded by commas into an array. The code for this is provided below in Figure 5.1:

```
// Database connection details
$database = new PDODatabaseAccess("mysql:host=localhost;dbname=major-project", "root", "");

if ($parameters = $database->ProvideParametersForNodes($_GET["id"]))
{
    echo "&"; // To separate the header from the data
    echo $parameters["needs_uv"] . "," . $parameters["uv_on"] . "," . $parameters["uv_off"]
    . "," . $parameters["needs_heat_mat"] . "," . $parameters["heat_mat_target"] . ","
    . $parameters["needs_heat_bulb"] . "," . $parameters["heat_bulb_target"] . ","
    . $parameters["humidity_target"] . ",";

    //e.g. "&1,08:00,20:00,0,32.0,1,30,60,"; // Comma on the end so the Arduino knows to stop reading
    //UV-NEEDED, UV-ON, UV-OFF, MAT-NEEDED, MAT-TARGET, AMBIENT-NEEDED, AMBIENT-TARGET, HUMIDITY-TARGET
}
}
```

Figure 5.1 – Web server read parameters for node devices code

The `ProvideParametersForNodes` method queries the database and returns the array of parameters to be output.

5.1.3. Receiving Node Data

Saving node readings into the database is achieved similarly to providing data to the nodes. GET parameters are read from the URL provided by the node, and these values are inserted into the database using the database access class.

5.1.4. Dashboard

The dashboard shows data for each terrarium, including its readings, targets, and actuator states. The dashboard has two modes: summary, which shows all terrariums with brief information, and individual, with more detailed information for a specified terrarium.

The dashboard feature is implemented as a single page, using Asynchronous JavaScript And XML (AJAX) functionality to dynamically swap out data depending on which option the user has chosen from a HTML select input.

AJAX is used to retrieve data as the user selects terrariums, which is displayed in the individual or summary terrarium views. These are shown below in Figures 5.2 and 5.3:

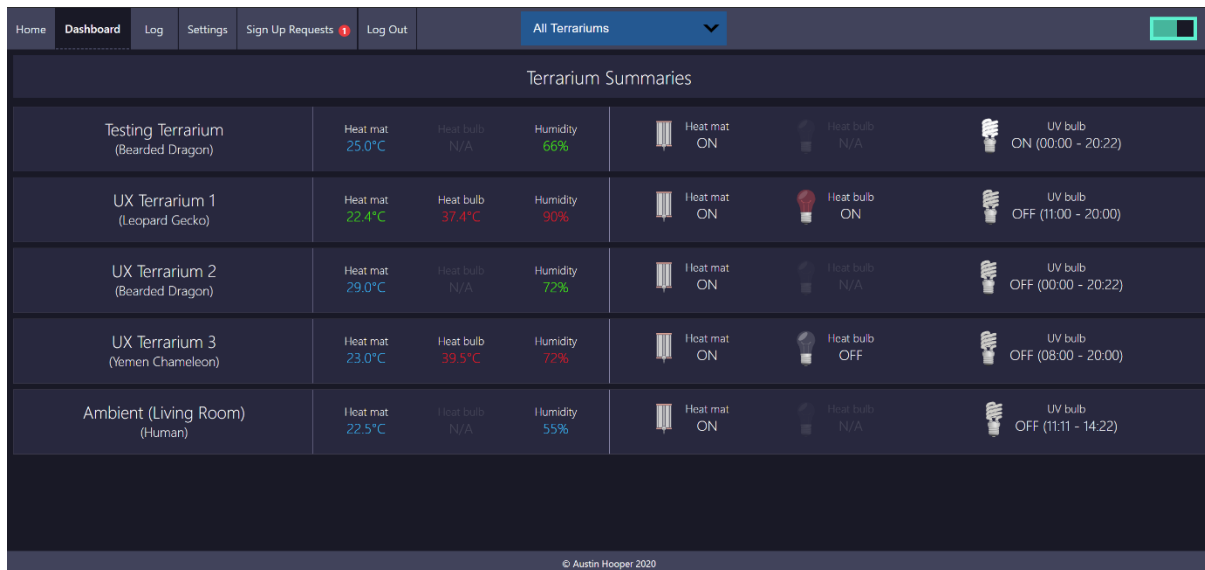


Figure 5.2 – Website dashboard summary view

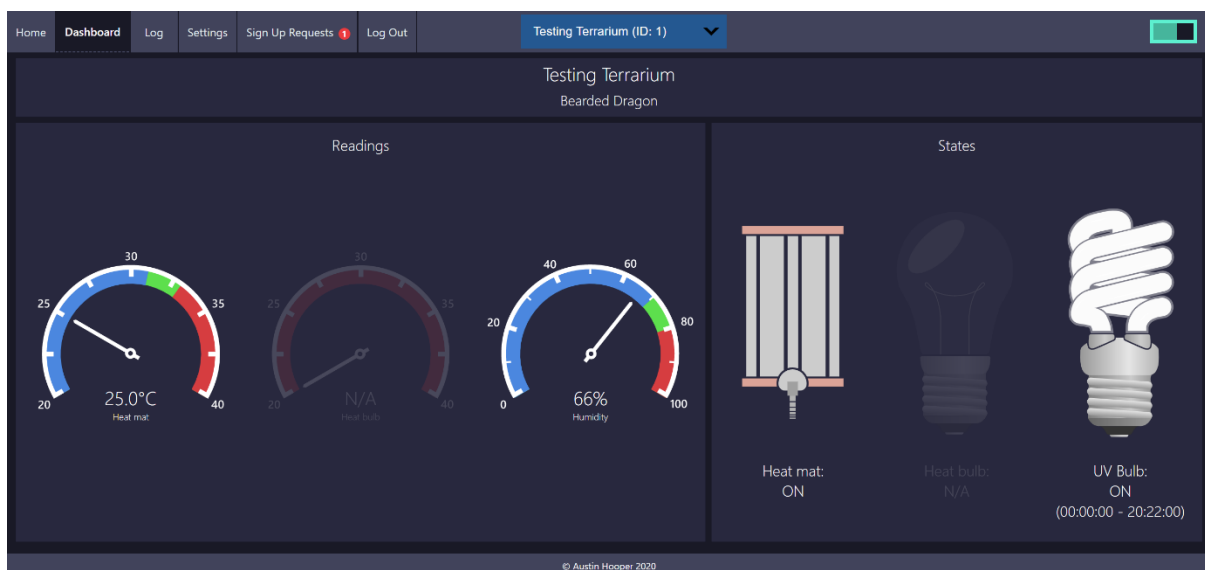


Figure 5.3 – Website dashboard individual view

Gauges on the individual dashboard were implemented using SVG graphics and custom `stroke-dasharray` values calculated based on the gauge circle's circumference, calculated using $2\pi r$. An arc of the circle, whose length was dependent on the target value, was then given a stroke to represent a shaded area.

5.1.5. Logs

The log page displays the most recent data for a specified terrarium. This is achieved through a dual Y-axis line chart – one for temperature, and the other for humidity. The X axis displays the time of the data log. For each reading line, there is an associated target reading, which is wider and transparent. The log functionality was implemented using the Google Charts JavaScript API [49].

AJAX is used to continuously poll the database for new readings. The jQuery `$.post` method was used, providing the name of the page to post to, and variables to post to it. In Figure 5.4 below, request and terrariumID values are posted. The result of this page is the PHP echoed results from the database as a JSON string, which is stored into the `newSingleRowData` variable for later use – adding the row to the chart.

```
// get the latest result for the terrarium with a given id
$.post("get-most-recent-logs.php", {request: "mostRecent", terrariumID: selectID},
function(newSingleRowData) {

    if (newSingleRowData != "") // If something was echoed out
    {
        newRow = JSON.parse(newSingleRowData);
    }
}
```

Figure 5.4 - Example AJAX code for retrieving website log data

Randomly generated example data is provided for 'UX Terrarium 3' in the web application to demonstrate the graph functionality. This was initially added for use in usability testing.

5.1.6. Progressive Web App Features

The site is implemented as a progressive web application (PWA). As discussed by Google [50], progressive web apps are:

- Reliable, as they function even without internet connectivity
- Fast, as they can cache files and information, and run natively on mobile and desktop operating systems
- Engaging, as they run natively on devices and support push notifications

PWAs function primarily using service workers. Service workers are scripts which are registered on the client device, and control one or more pages of a site. They primarily handle HTTP requests, push notifications, application install events, and application cache [32, pp. 22], [52].

A PWA can be installed from within the browser if all requirements are met (manifest file present, site served over HTTPS, service worker running [51]), as shown in the below Figure 5.5:

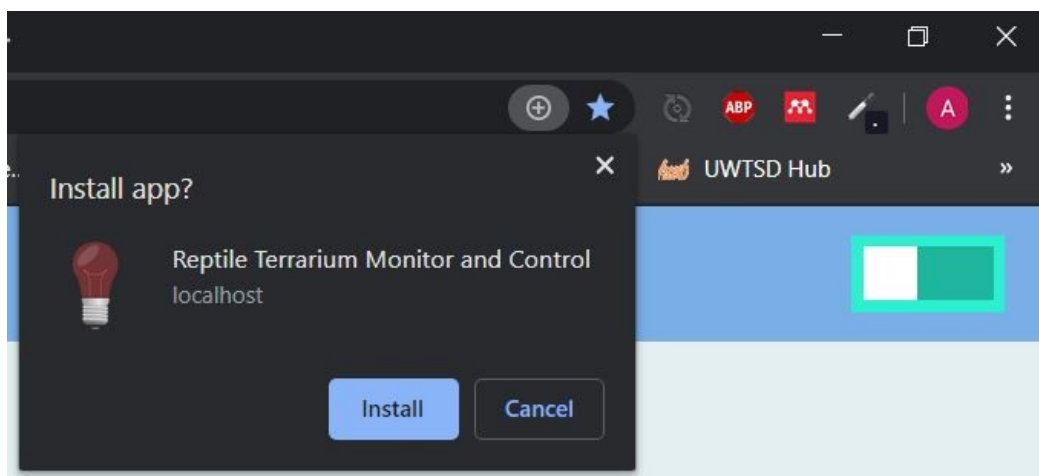


Figure 5.5 – Progressive web app install pop-up

The installed application is treated as a native app by the operating system – see below in Figure 5.6 an example of the installed app in Windows 10:

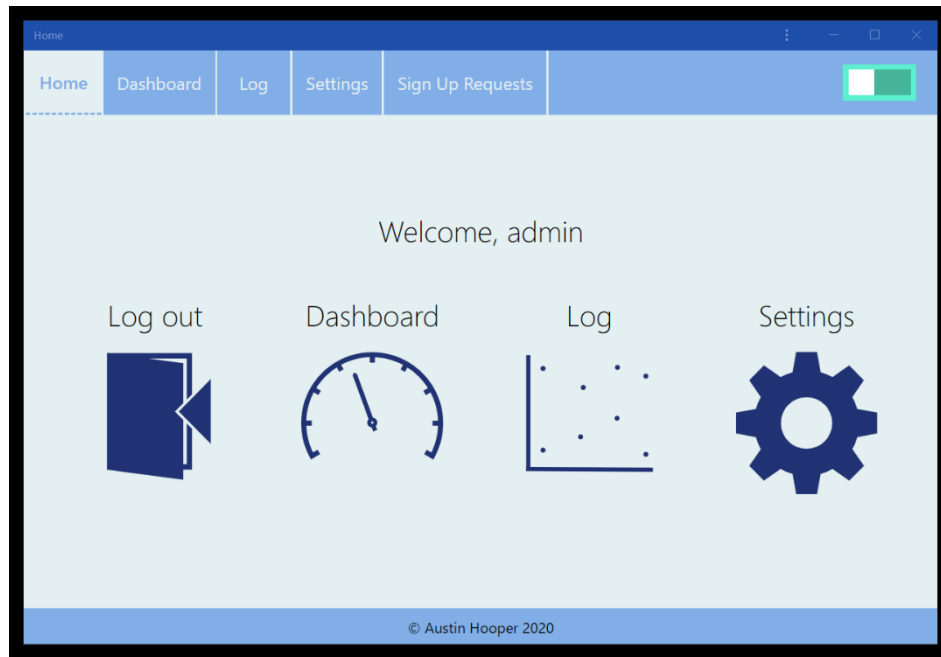


Figure 5.6 - Progressive web app in Windows

Many of the features PWAs provide are reliant on secure HTTPS connections, achieved by using SSL certificates. An SSL certificate is a digital certificate, providing authentication for a website and allowing for connection encryption [53]. In the case of PWAs, HTTPS is particularly important as it is required for application installation and push notifications [32, pp. 41]. An SSL certificate was generated and installed using commands detailed in Appendix E, and the web server was set to redirect to the HTTPS-prefixed address.

Local notifications are triggered when one or more terrariums have sensor readings which exit their target range. Notifications are built as follows in Figure 5.7, with the call to `new` displaying the notification:

```
const options = {  
  body: notificationBody,  
  icon: "images/warning.png",  
  image: "images/notification-image.jpg",  
  silent: false,  
  requireInteraction: false // Whether the user has to click a button to dismiss it  
};  
  
// Create a new notification object, which is immediately showed on construction  
var notification = new Notification(title, options);
```

Figure 5.7 - JavaScript notification code

This notification appears as follows in Figure 5.8 on Google Chrome for Windows 10:

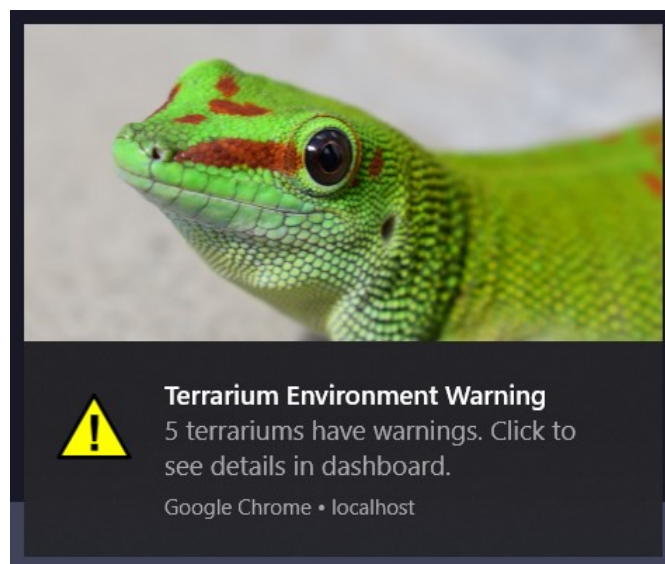


Figure 5.8 - Notification popup

5.2. Node Device Development

This subsection details development of node devices using the ESP8266 microcontroller. Further photos from throughout the project are included in Appendix F. Deep sleep was not implemented as it requires pin D0 to be connected to the RST pin for waking from sleep, which was already in use. A sensor was initially connected to pin D8, however, it was discovered that this pin is used to determine boot mode, so could not be used. As a result, the sensor was moved to the only remaining GPIO pin – D0, making it unusable for waking the node from deep sleep.

Had pin D0 been available, the only modifications necessary would be to short it to the RST pin and to replace the Arduino `delay()` function with:

```
system_deep_sleep(DELAY_DURATION) ;
```

where **DELAY_DURATION** is a value in μ s.

See in Figures 5.9 and 5.10 the final hardware for a node device:

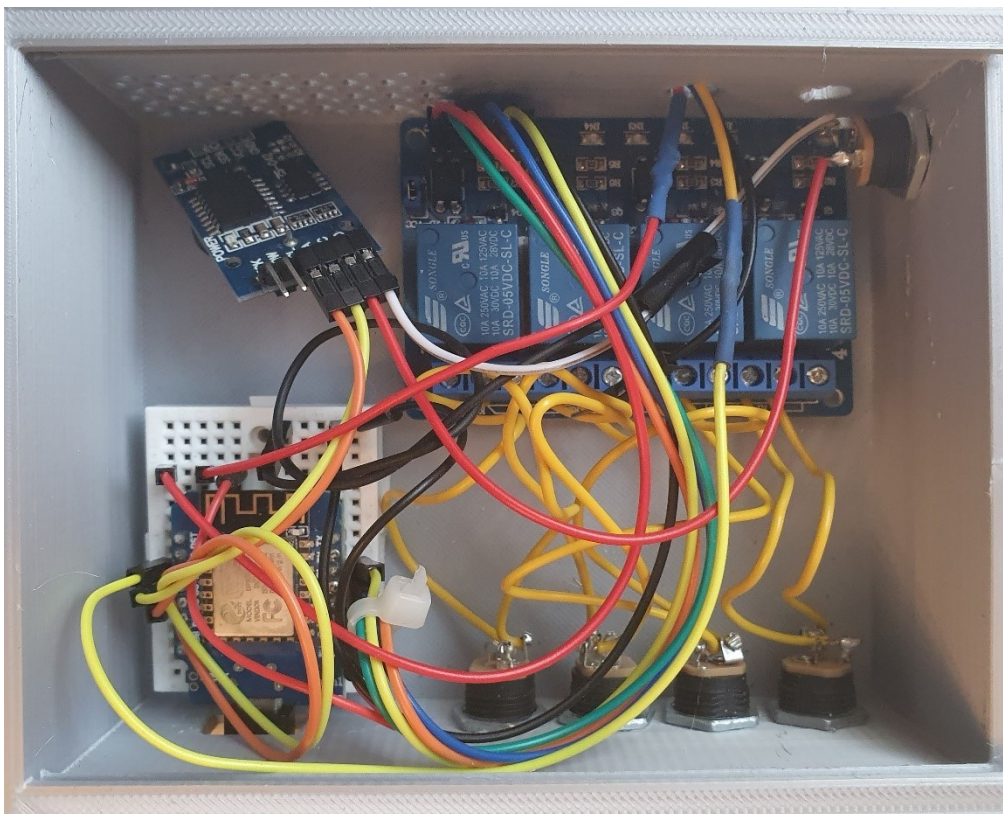


Figure 5.9 - Node device internals

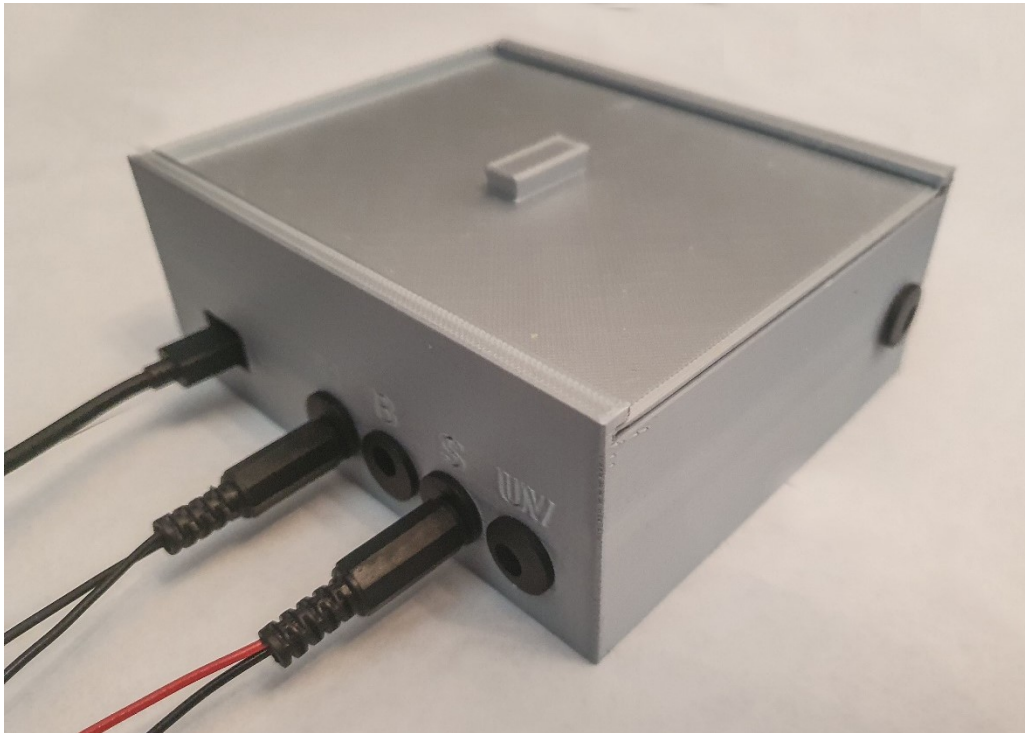


Figure 5.10 - Node device in 3D printed case

5.2.1. Node Device Software

The node device programming used the Arduino language. See below highlights of code. As each node device only represents one instance – itself, and the data it handles is simple, the node programming took a procedural paradigm approach as object-orientation provides no benefits in this specific scenario.

The **SendDataToServer** function converts the current sensor readings to strings, and creates a URL, using them as GET parameters. This is then sent to the server by accessing the created URL in a custom HTTP GET request. A section of this function is included in Figure 5.11:

```

// Really long URL for the GET parameters
url += "?enclosureID=" + String(DEVICE_ID) + "&heatMatTemperature=" + heatMatTemperatureString +
      "&heatBulbTemperature=" + heatBulbTemperatureString + "&humidity=" + humidityString +
      "&heatMatTarget=" + heatMatTargetString + "&heatBulbTarget=" + heatBulbTargetString +
      "&humidityTarget=" + humidityTargetString + "&heatMatPower=" + matPowerString +
      "&heatBulbPower=" + heatBulbPowerString + "&uvPower=" + uvPowerString;

Serial.println(url);

Serial.printf(">>> NETWORK: Connecting to %s to send data... ", SERVER_IP);

if (!wifi_client.connect(SERVER_IP, HTTPS_PORT)) {
    Serial.println("connection failed!");
}
else // If connected, send a custom GET request for the php page that takes data and saves it
{
    wifi_client.print(String("GET /") + url + " HTTP/1.1\r\n" + "Host: " + SERVER_IP +
                      "\r\n" + "Connection: close\r\n" + "\r\n");
}

```

Figure 5.11 - Node SendDataToServer code

The **ReadServerData** function requests a PHP page and provides an ID in the URL. The server then responds with details for that enclosure, which the node reads from the HTTP response. The data is separated from the header by an &, and each field is separated by comma. Below is a section of the code for reading each value to an array.

```

if (wifi_client.connect(SERVER_IP, HTTPS_PORT))
{
    Serial.println("connected, sending request.");

    String url = "/node-comm/node-retrieve-params.php/?id=" + String(DEVICE_ID);
    wifi_client.print(String("GET /") + url + " HTTP/1.1\r\n" + "Host: " + SERVER_IP
                      + "\r\n" + "Connection: close\r\n" + "\r\n");

    response = wifi_client.readStringUntil('&'); // Read up until &. Until \r\n should

    while (parametersPosition <= 7) // While not on the last parameter - This loop sa
    {
        parameters[parametersPosition] = wifi_client.readStringUntil(','); // Read unt
        parametersPosition++;
        Serial.println(parameters[parametersPosition]);
    }

    wifi_client.stop(); // Disconnect and thus stop running this while loop
}

```

Figure 5.12 – Node code for reading each parameter separated by comma

The `TimeWithinRangeCheck` function reads the current time from the DS3231 real-time clock, then converts this to a `time_t` object using another function. This is then repeated for the on and off times, and for hard-coded times of 00:00:00 and 24:00:00, to be used in later logic. Though a time of 24:00:00 is not possible, this was used because the C `time_t` type is a POSIX timestamp, representing the number of seconds since 00:00:00, Jan 1 1970 UTC [54]. Due to this, the most appropriate solution was simple integer logic.

The most important function is the `ReadAndReact` function. It uses the other functions to poll sensors, compare their readings with targets acquired from the server, and power or un-power relays as appropriate. Below in Figure 5.13 is a section of this function, showing the process for controlling the heating mat:

```
// Heat mat sensor readings asnd response, if present
if (needsMat)
{
    matTemperature = ReadSensor(matDHT, 'T');

    Serial.printf("# Mat target:  %.1f°C", matTarget);
    Serial.printf(" | ");
    Serial.printf("Mat temp:  %.1f°C ", matTemperature);

    if (matTemperature < matTarget) // if current ambient is under the target
    {
        Serial.println("| Under target, powering mat.    X");
        SetRelayState(HEAT_MAT_RELAY, true);
        currentMatState = true;
    }
    else if (matTemperature > matTarget)
    {
        Serial.println("| Above target, unpowering mat.  X");
        SetRelayState(HEAT_MAT_RELAY, false);
        currentMatState = false;
    }
    else
    {
        Serial.println("| Within 1°C of target.          ✓");
    }
}
```

Figure 5.13 - Node `ReadAndReact` code

5.3. Troubleshooting

5.3.1. Web Application Graph

The log page was initially set up to display the most recent 50 logs, regardless of timestamp. This was achieved by querying the database using the LIMIT keyword, in the below query:

```
SELECT * FROM log WHERE terrarium_id = <ID> ORDER BY log_id  
DESC LIMIT 50
```

After selecting the initial 50 readings on page load, the page would run a loop checking for new readings – readings with an ID different to the most recent. If a new reading was found, the oldest reading was removed from the graph in order to retain the maximum of 50 logs. This had the adverse effect of showing large gaps in readings when the nodes are powered off, and occasional line glitches, as shown in Figure 5.14:

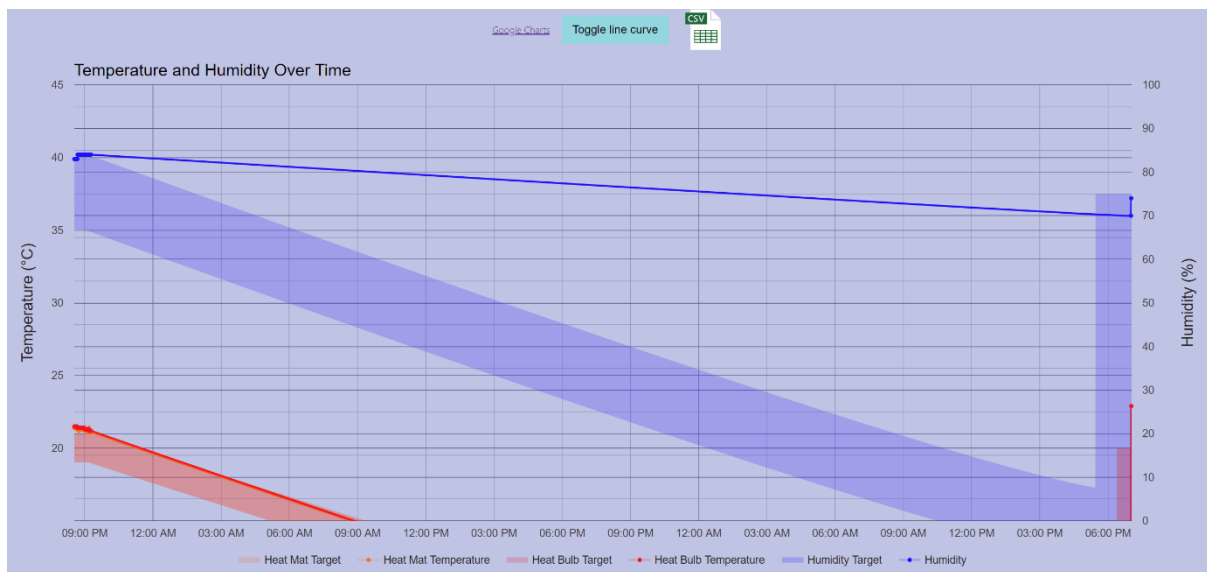


Figure 5.14 – Log graph with reading count limit

An improved and more practical limit for readings is by time. Instead of readings count, a query was implemented to retrieve all readings within a time range. This query is shown below:

```
SELECT * FROM log
WHERE terrarium_id = <ID>
AND timestamp >= DATE_SUB(NOW(), INTERVAL <number_of_minutes>
MINUTE)
ORDER BY log_id DESC
```

The user is now able to choose a time range from a select input, as shown in Figure 5.15, along with the output:

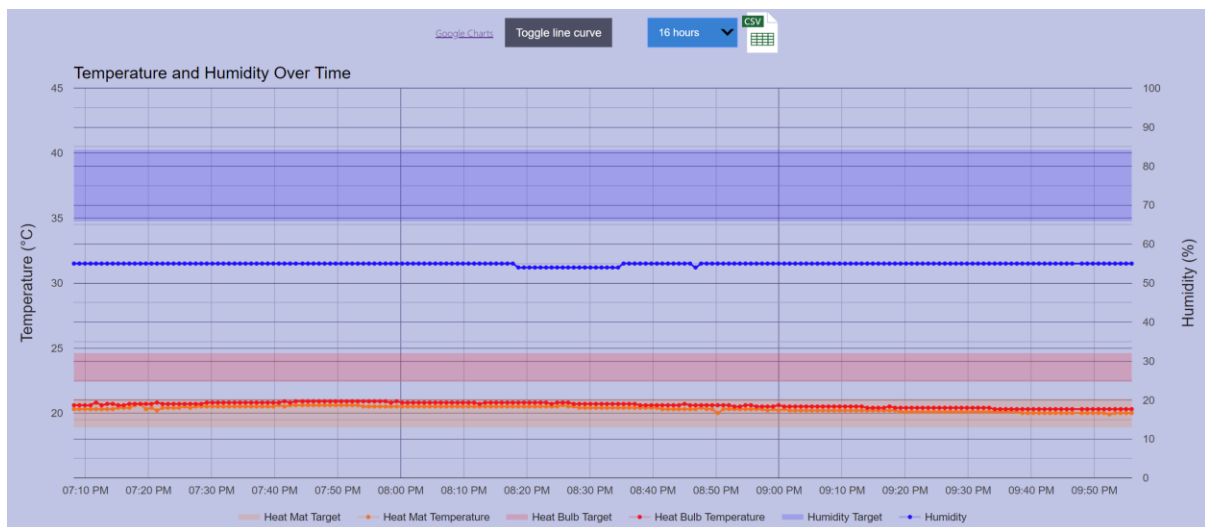


Figure 5.15 – Log graph with time range limit

5.3.2. SSL and HTTPS

Configuring a self-signed SSL certificate proved challenging. Difficulties were met with certificate fields (detailed in Appendix E). The certificate was installed directly to browsers, not the Windows certificate store, and had a Subject Alternative Names (SAN) field. Though this field is for storing additional domain names or addresses [55], some browsers require this field, even if there is only one.

5.3.3. Sensor Readings

Initially, the DHT22 sensor readings' humidity values were very inaccurate, 20-30% higher than expected and regularly above 90%. This was determined to be due to the length of the sensor wires (~2 metres), causing a drop in voltage reaching the sensor due to resistance. This was solved by halving the length of the wires.

5.3.4. Node Power

During development, it was observed that node devices would fail to read their sensors if all peripherals were connected. When the voltage across a sensor was probed (see Figure 5.16), it read 2.2 – 2.5V, where the sensors required 3 – 5V. This was because the peripherals were connected in series, which provided the same current across all devices, but a shared voltage. This was modified to be a parallel circuit, where the voltage is the same across all devices, with a shared current. However, this could not provide enough current for some devices.

This issue became apparent far into the project, when it would not be viable to change microcontroller board for one which handled higher power. To resolve this issue, as discussed in the design section, the real-time clock was connected to a separate, external power source, but grounded to both the external source and the node, to allow for it to be read properly using the I²C interface.



Figure 5.16 – Troubleshooting: real-time clock voltage reading

5.4. System Development Conclusion

This section has detailed the system implementation and discussed difficulties encountered, along with approaches taken to resolve these difficulties. The next section discusses system testing and provides results.

6. Testing

This section details testing undertaken on the system. Testing consisted of functional testing of the control system and usability testing of the web application. Test results are provided and visualised in graphs against the initial environment targets.

6.1. Functional Testing

Humidity and temperature were first tested separately to remove variables from the tests and also to gauge the functionality of the system for each independently, before testing both together. As temperature rises, humidity decreases and thus one environment variable could affect the other. Results in this section have been rounded to two decimal places. Each set of tests consisted of four test runs. For example, temperature control was tested before and after code modifications, with four tests for each of these, along with a further four extended tests.

The heat mat and sensor were placed to the side of the sprayer, in order to get a general ambient humidity reading, without water being sprayed directly onto the sensor. This is shown below in Figure 6.1:



Figure 6.1 - Testing setup

6.1.1. Standalone Surface Temperature Control

Initially, surface temperature tests were run for four hours each, as they were running on battery power due to safety concerns. After confirming safety of using wall adapters, extended tests were conducted for 36 hours each, to test consistency of the system over time. Ambient temperature was logged to determine how room temperature affected the system. The target was set at 32°C, which is a common target for many reptiles and is within the capability of the heating element.

The system's conditional logic for temperature control was initially as follows:

```
if temperature < target: power heat pad
```

This was modified as the temperature was always colder than expected, and an offset of $+\frac{1}{3}^{\circ}\text{C}$ was added:

```
if temperature < (target + 0.333): power heat pad
```

This was very effective in increasing the temperature to remain within the target range. See in Tables 6.1 and 6.2 the minimum, maximum, mean, and standard deviation of each test run before and after the offset was applied. These values were calculated from the top of the first temperature spike, to exclude the readings from when the heat pad was initially heating up.

Table 6.1 - Temperature test runs (°C) without + $\frac{1}{3}$ °C code offset

	Run 1	Run 2	Run 3	Run 4
Minimum	30	30.7	30.4	30.5
Maximum	32.9	32.4	32.4	32.3
Mean	31.51	31.58	31.51	31.49
Std. deviation	0.52	0.53	0.52	0.46

Table 6.2 - Temperature test runs (°C) with + $\frac{1}{3}$ °C code offset

	Run 1	Run 2	Run 3	Run 4
Minimum	31	30.9	31.1	31.1
Maximum	32.7	33	32.8	33
Mean	31.88	31.96	31.91	32.09
Std. deviation	0.45	0.51	0.48	0.53

The extended tests run for 36 hours, with the offset applied, showed similar results to the shorter tests, though with more extreme minimum and maximum values. These tests were run to validate the system's efficacy over time and to view the correlation between room temperature over time and the performance of the system. These test results are shown below in Table 6.3:

Table 6.3 – 36 hour temperature test runs (°C)

	Run 1	Run 2	Run 3	Run 4
Minimum	30.7	30.7	30.7	30.7
Maximum	33	33.4	33.2	33.3
Mean	31.85	32.03	31.93	31.98
Std. deviation	0.51	0.6	0.58	0.59

Graphs for each of these three sets of runs are shown in Figures 6.2, 6.3 and 6.4. The $\pm 1^{\circ}\text{C}$ tolerance target is shown with a green shaded area.

An existing commercial product, the Microclimate Ministat 100, was tested under the same conditions as the project's implementation. The same heating mat was connected to the Ministat, with the Ministat sensor placed next to the data logging sensor. The results were similar to that of the project's implementation, though with a temperature offset above the target, as shown in Table 6.4:

Table 6.4 - Microclimate Ministat 100 test runs ($^{\circ}\text{C}$)

	Run 1	Run 2	Run 3	Run 4
Minimum	31.6	32.2	32.5	32.1
Maximum	33.9	33.9	34.5	34
Mean	32.61	33.03	33.4	33.03
Std. deviation	0.52	0.55	0.58	0.54

Temperature Over Time (Including Ambient)
Against Initial Target of $\pm 1^{\circ}\text{C}$ - Without Offset

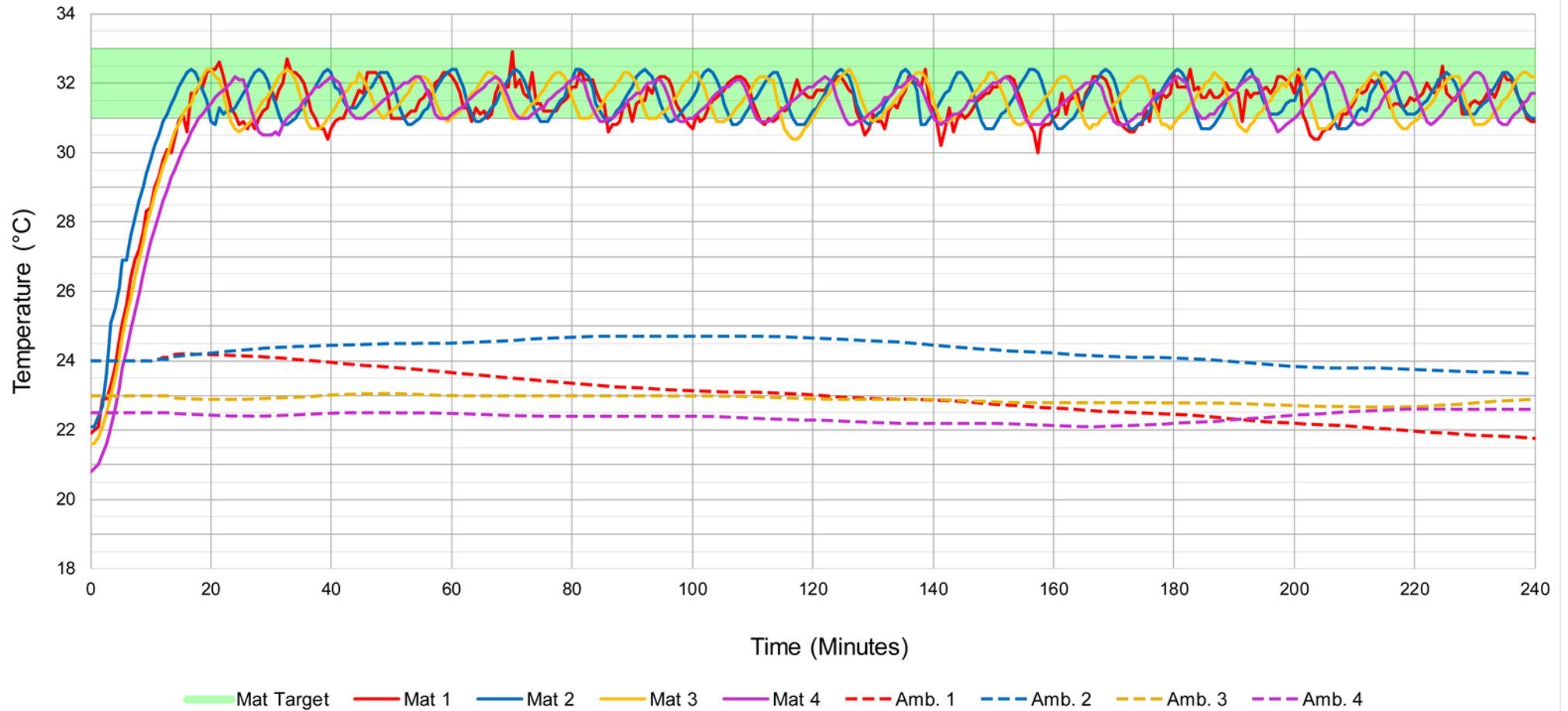


Figure 6.2 - Temperature test without $+\frac{1}{3}^{\circ}\text{C}$ code offset

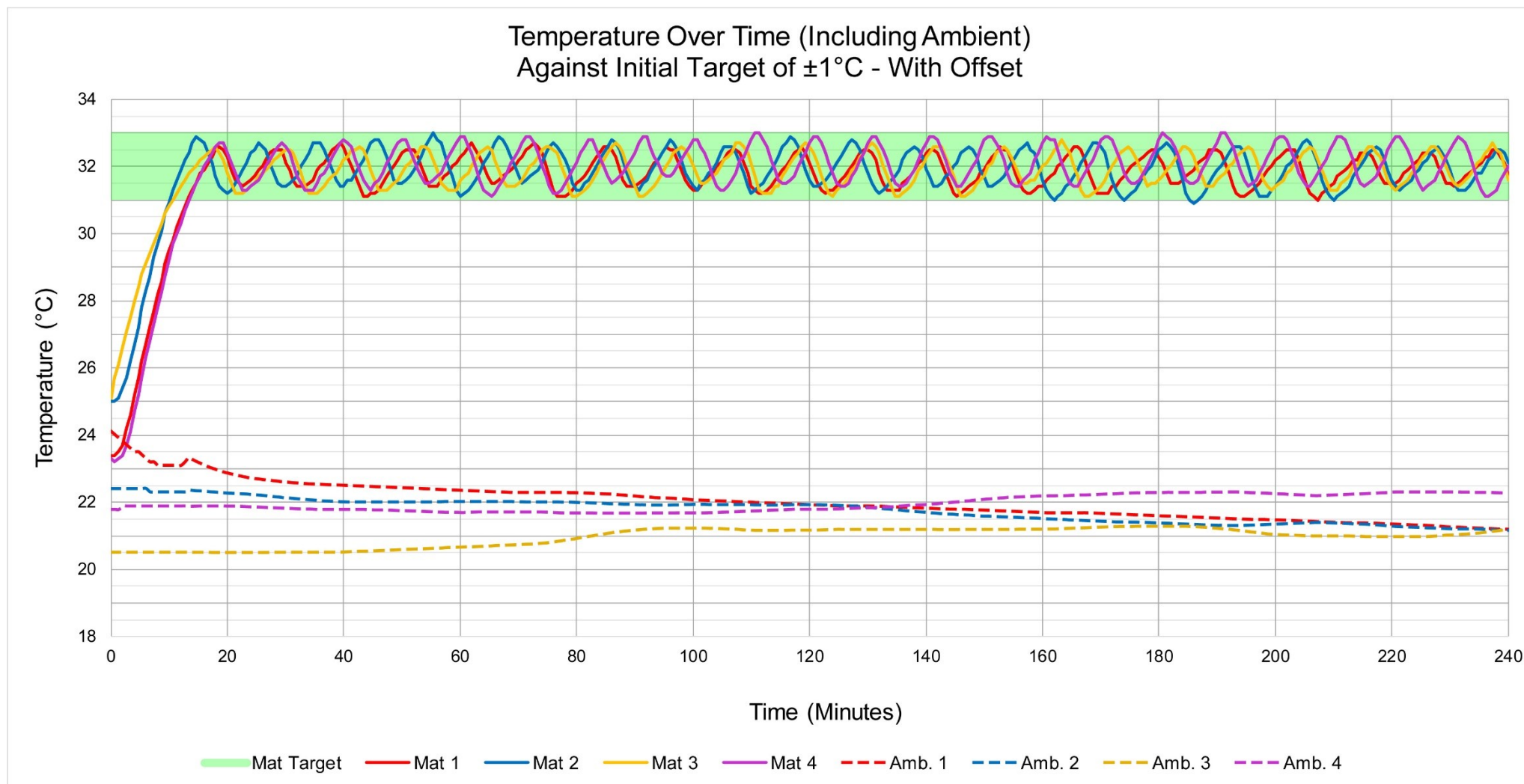


Figure 6.3 - Temperature test with $+\frac{1}{3}^{\circ}\text{C}$ code offset

Temperature Over Time (Including Ambient)
Against Initial Target of $\pm 1^{\circ}\text{C}$ - Extended Tests

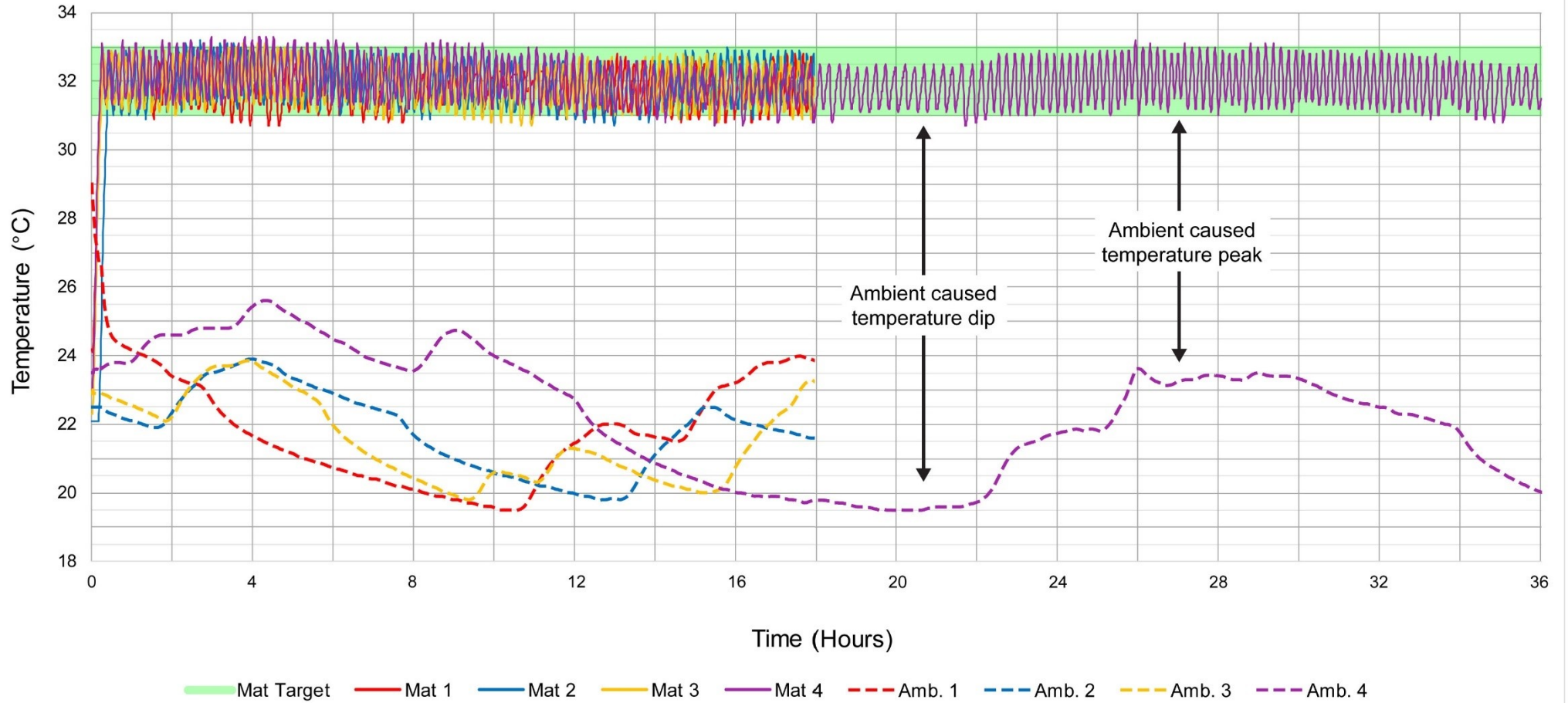


Figure 6.4 - Extended test runs with $+\frac{1}{3}^{\circ}\text{C}$ code offset

6.1.2. Standalone Humidity Control

In order to test humidity control on its own, the system was run alongside a standard reptile heat lamp (plugged in to a wall socket, not controlled – see Figure 6.5 below), to reduce the humidity to a typical value for a terrarium without misting. This allowed the system to spray when necessary, raising the humidity to the target. A carpet made for terrariums was also included to absorb some humidity like in a real terrarium.

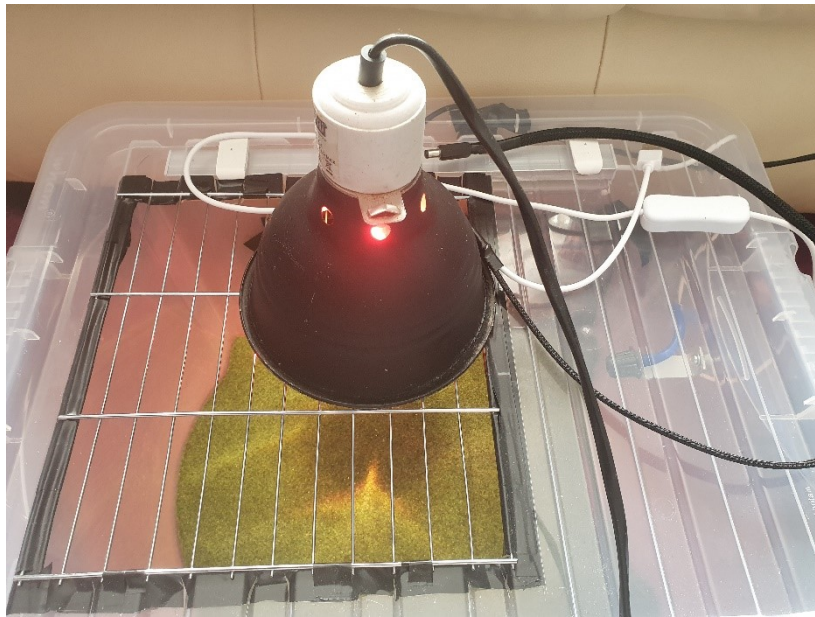


Figure 6.5 - Heating lamp setup

As with temperature control, the humidity control code was modified, with tests before and after the modification. Initially, the conditional logic was as follows:

```
if (humidity <= (humidityTarget - 10): power sprayer for 2.5s
```

This negative offset was in place as the humidity was expected to continue increasing after spraying had stopped. Due to sensor placement, this was not the case, and so the humidity was always below its target. The code was changed, with the -10% offset being removed. Tests were run for both of these scenarios with a target humidity of 75% to test for improvements after modifying the code. Results of these tests are shown in Tables 6.5 and 6.6, calculated after humidity stabilised at 1

hour, followed by graphs in Figures 6.6 and 6.7, with the $\pm 10\%$ target area shaded in green.

Table 6.5 – Initial standalone humidity control test runs (%)

	Run 1	Run 2	Run 3	Run 4
Minimum	61	63	62	64
Maximum	79	77	75	73
Mean	72.41	67.89	69.37	67.2
Std. deviation	4.22	2.39	2.49	1.8

Table 6.6 – Standalone humidity control test runs after code change (%)

	Run 1	Run 2	Run 3	Run 4
Minimum	72	70	70	72
Maximum	85	90	89	90
Mean	77.51	78.25	78.85	80.05
Std. deviation	2.55	3.69	3.66	4.19

Humidity Over Time (Including Ambient)
Against Initial Target of $\pm 10\%$ - Before Code Change

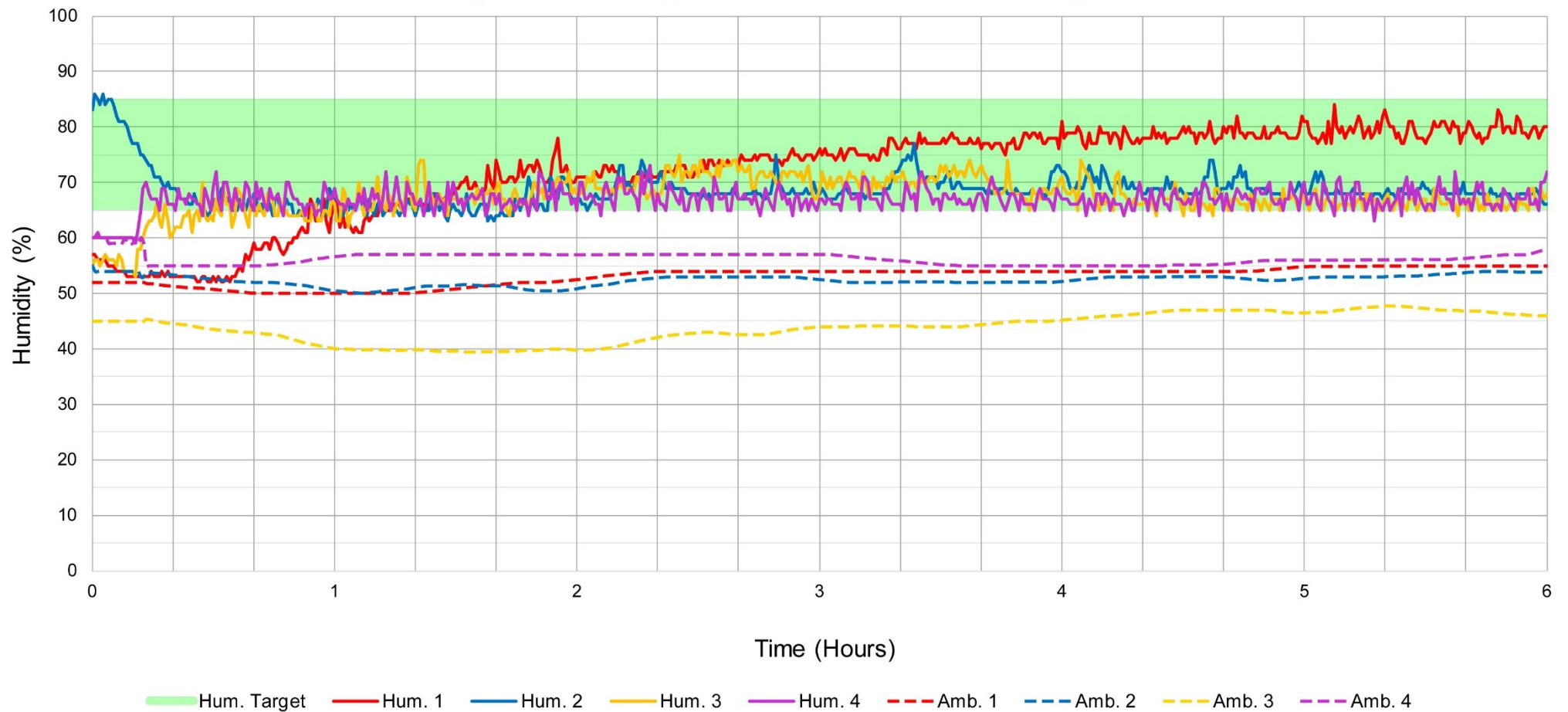


Figure 6.6 - Initial standalone humidity control test runs

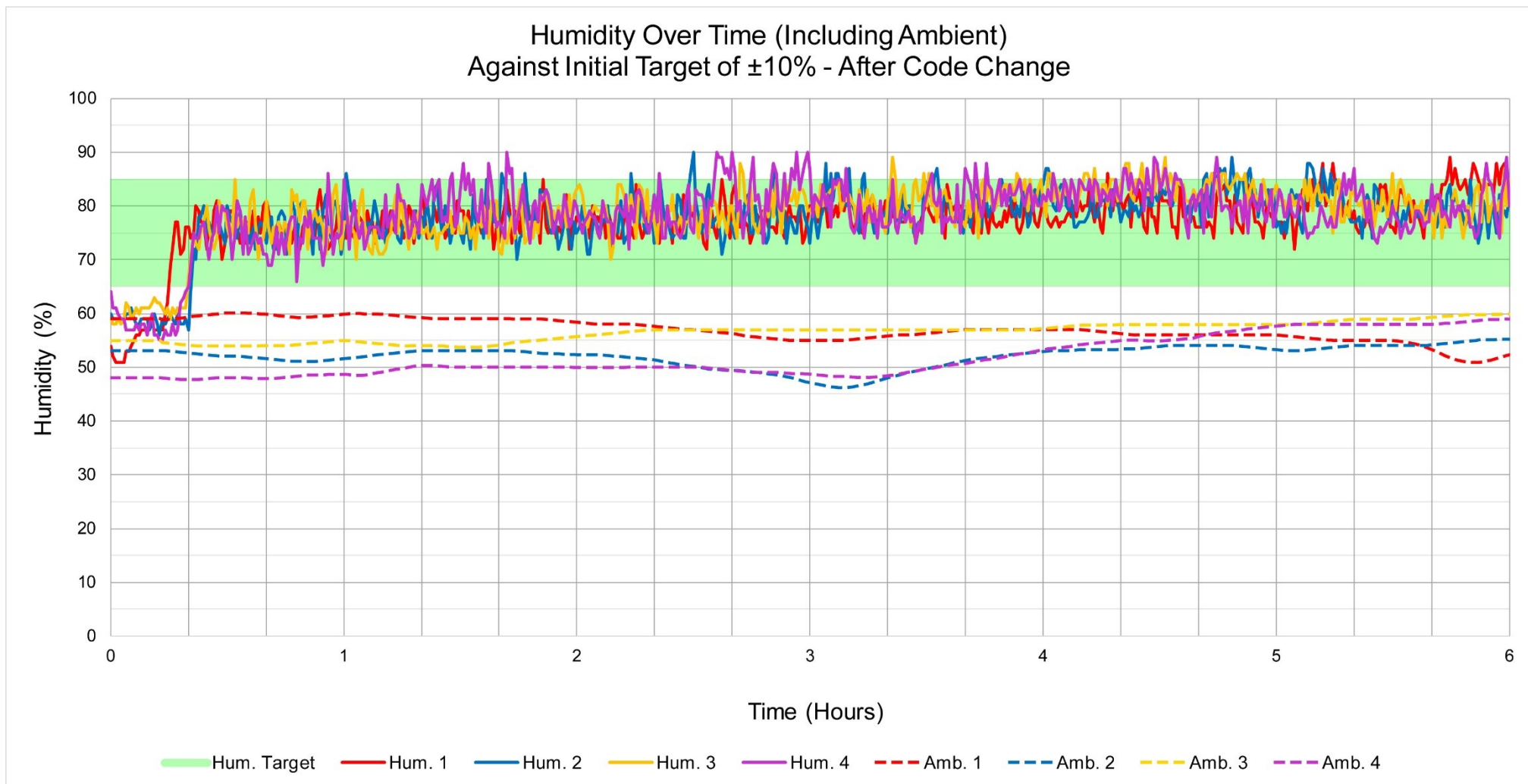


Figure 6.7 – Standalone humidity control test runs after code change

6.1.3. Combination Temperature and Humidity Control

A heat lamp was not used for these tests as the heat was provided by the heating pad. Target temperature and humidity remained the same as previous tests, at 32°C and 75% respectively. The results below in Table 6.7 and Figure 6.8 show that the system could not control humidity while the heating element was in use. This is discussed further in Chapter 7.

Table 6.7 – Combined temperature and humidity control results

		Run 1	Run 2	Run 3	Run 4
Temperature (°C)	Minimum	30.9	30.9	31	30.9
	Maximum	32.7	32.7	32.5	32.6
	Mean	31.8	31.81	31.8	31.89
	Std. deviation	0.48	0.49	0.35	0.39
Humidity (%)	Minimum	56	55	44	50
	Maximum	75	68	59	63
	Mean	63.88	62.02	53.78	56.84
	Std. deviation	3.64	2.53	3.22	2.41

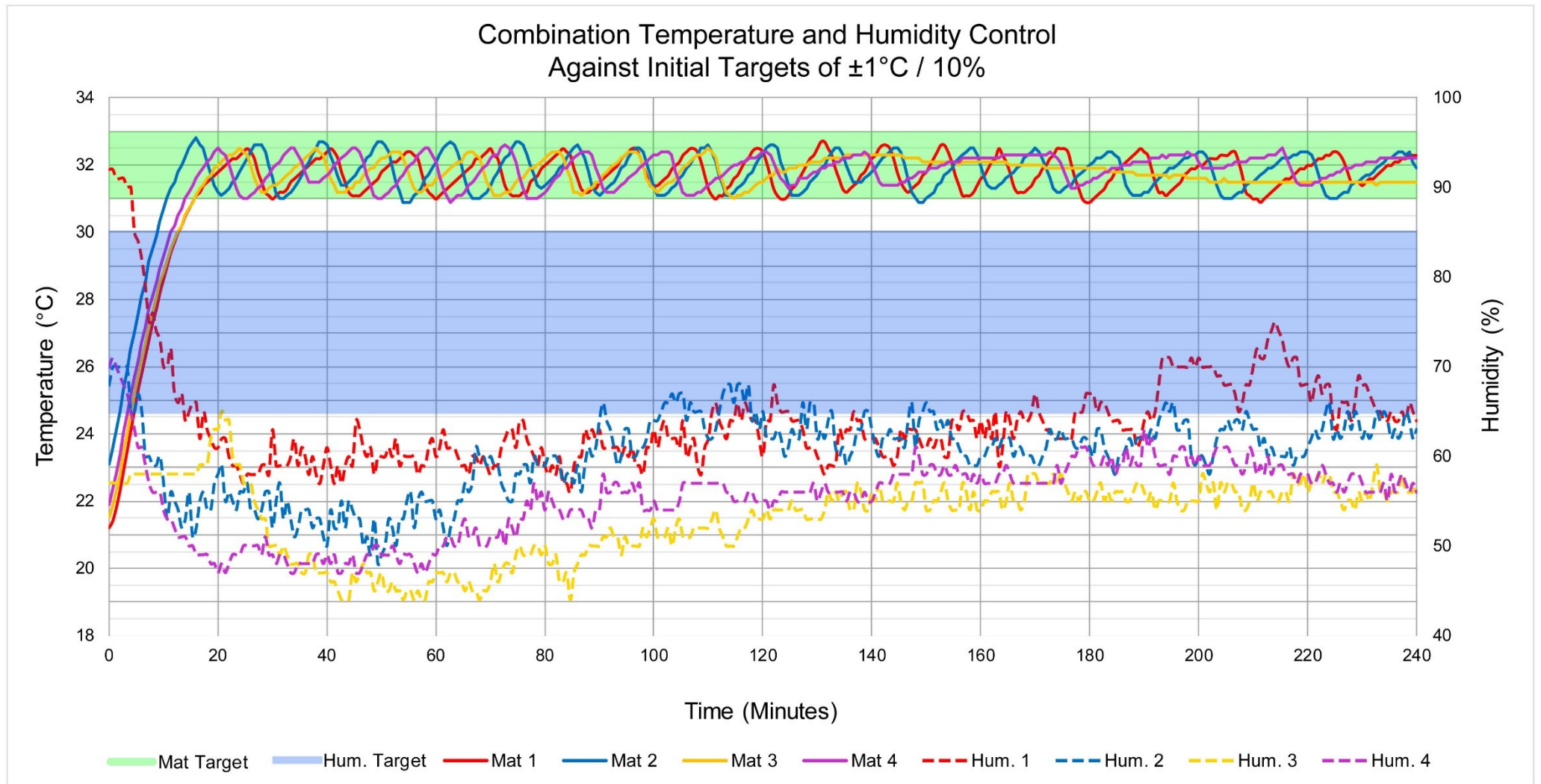


Figure 6.8 - Combined temperature and humidity control

6.4. Web Application User Experience and Accessibility Testing

User experience focuses on understanding an application's users, their needs, values, abilities, and therefore their limitations [56], [57]. Particularly in the context of web applications, accessibility has a significant role in ensuring that the user experience is positive, especially for people with disabilities. Web accessibility is defined more directly by the W3C [58] as the design and implementation of sites in a way which people with disabilities can use, specifically to perceive, understand, navigate, and interact with. This is expanded, specifying some examples of disabilities which should be catered for:

- Auditory
- Cognitive
- Neurological
- Physical
- Speech
- Visual

The current primary web accessibility guideline is the W3C's Web Content Accessibility Guidelines (WCAG) 2.1 [59]. The WCAG is internationally recognised – meeting level AA of these requirements is part of the UK Government's accessibility requirements [60]. Though accessibility is especially important for those with disabilities, it is also significant for those without disabilities.

6.4.1. User Experience Testing

Non-functional testing was conducted on the web application in the form of User Experience (UX) testing with 15 participants. This number of participants was chosen due to the scope of the project, and would ideally have been greater. Another consideration is that most participants were mostly classmates or were comfortable and familiar with web interfaces. As such, results may not necessarily represent the typical user of the web interface. Despite this, testing provided valuable insights into the usability of the system, along with many potential improvements. UX testing was conducted with each participant alone, away from any other potential participants – this ensured that nobody had an advantage having already seen the system. Due to the nature of the web application, the user experience testing focused on the users' abilities to perceive information which was presented on the pages.

The test consisted of a traditional list of tasks for participants to complete using the web app, questions to assess their ability to use the system, and System Usability Scale (SUS) questions. The SUS is a tool created by John Brooke in 1986, which provides ten statements for participants, each expecting a response as a Likert Scale number, from 1–5 [61], [62]. The scale gives a score from 0–100, by converting the raw scores for each Likert statement. The author's preference was to avoid a five-point scale due to the tendency for participants to choose "neutral," however, the SUS requires a five-point Likert scale for its score calculations.

Due to time constraints, testing was undertaken on the desktop view only (1920x1080 resolution, 15.6" display). A PHP function was written to set up the database logs table for each user experience test, on page load – this ensured every participant experienced the same data. In future, tests could be performed on smaller displays such as tablet and mobile devices, to test the application's responsiveness.

A sample response of the questionnaire and activities document is included in Appendix G, along with participant responses. The score for each participant was calculated as described in [62]. Participants were observed when undertaking the tasks, with a notes sheet used to describe any observations made about each participant. The test would have ideally been conducted using eye tracking

technologies or similar, to further aid in documenting testing and provide further insights.

The web application scored well on the SUS; the mean overall score was 78/100. The overall scores for each participant are visualised in Figure 6.9 below, compared to the average SUS score:

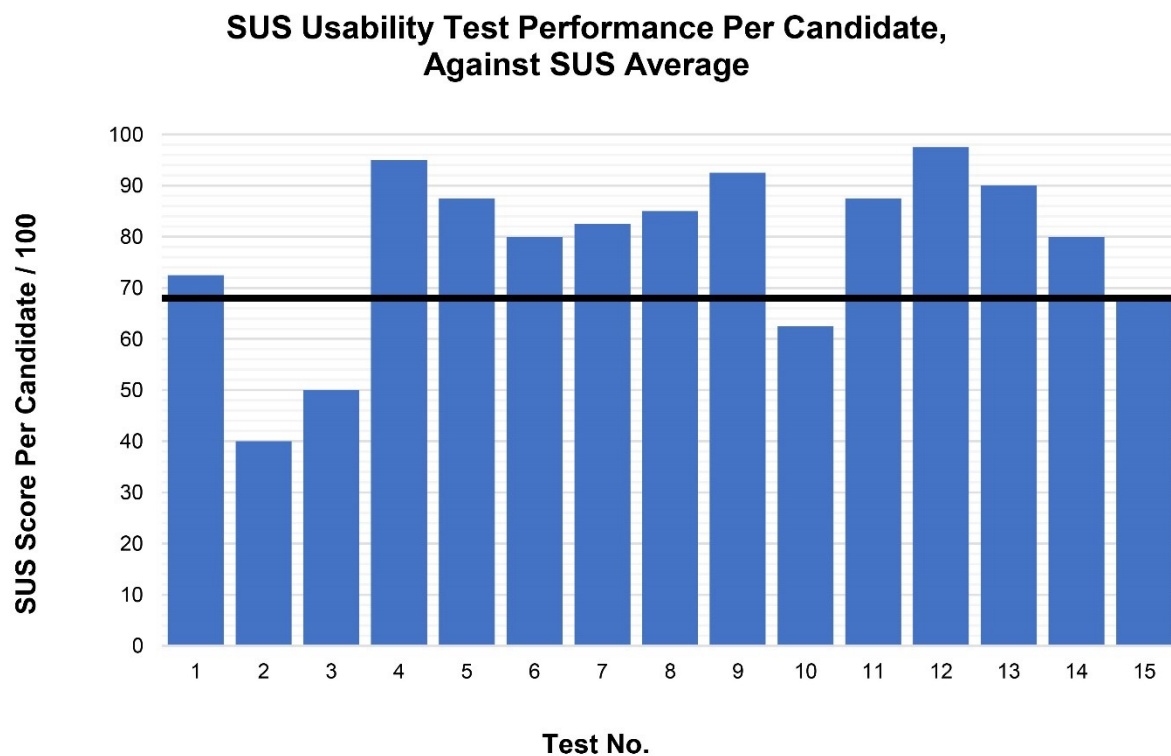


Figure 6.9 – Overall SUS scores per participant

See in Figure 6.10 the mean scores from 0-4 of each statement in the SUS, across every participant. A scale of 0-4 was used as this allows for the score axis to start at 0, along with the SUS using a range of 0-4, not the typical 1-5. These scores are normalised. Negative statements, such as “I found the system very cumbersome to use” have their scoring flipped compared to positive statements – 0/4 is best (“strongly disagree”), whereas 4/4 is worst. For negative statements, the normalised score was calculated as $4 - \text{score}$, whereas positive statements used the raw value. This allows for a more direct comparison of the performance for each statement. Standard deviations were also calculated per statement, and plotted on the graph as

error bars. Some error bars extend above the top of the graph, and were denoted using upwards arrows. These have been intentionally cropped, as the highest value possible on a 0-4 scale is 4, and allowing for higher values on the chart would be visually unbalanced.

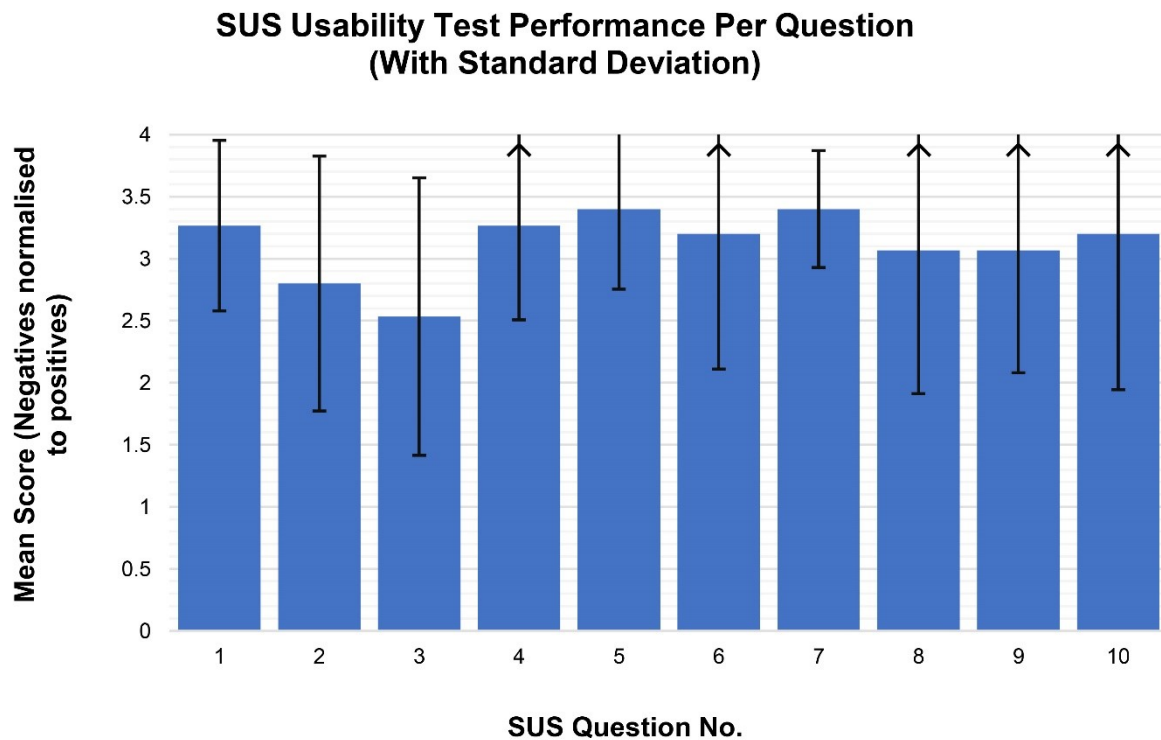


Figure 6.10 - SUS mean scores with standard deviations

For reference, the 10 SUS Likert statements are included at the end of the user experience questionnaire in Appendix G.

The individual SUS statement scores were also graphed using stacked column bar charts to visualise the ratios between positive and negative responses to the SUS statements. This is included in Figure 6.11.

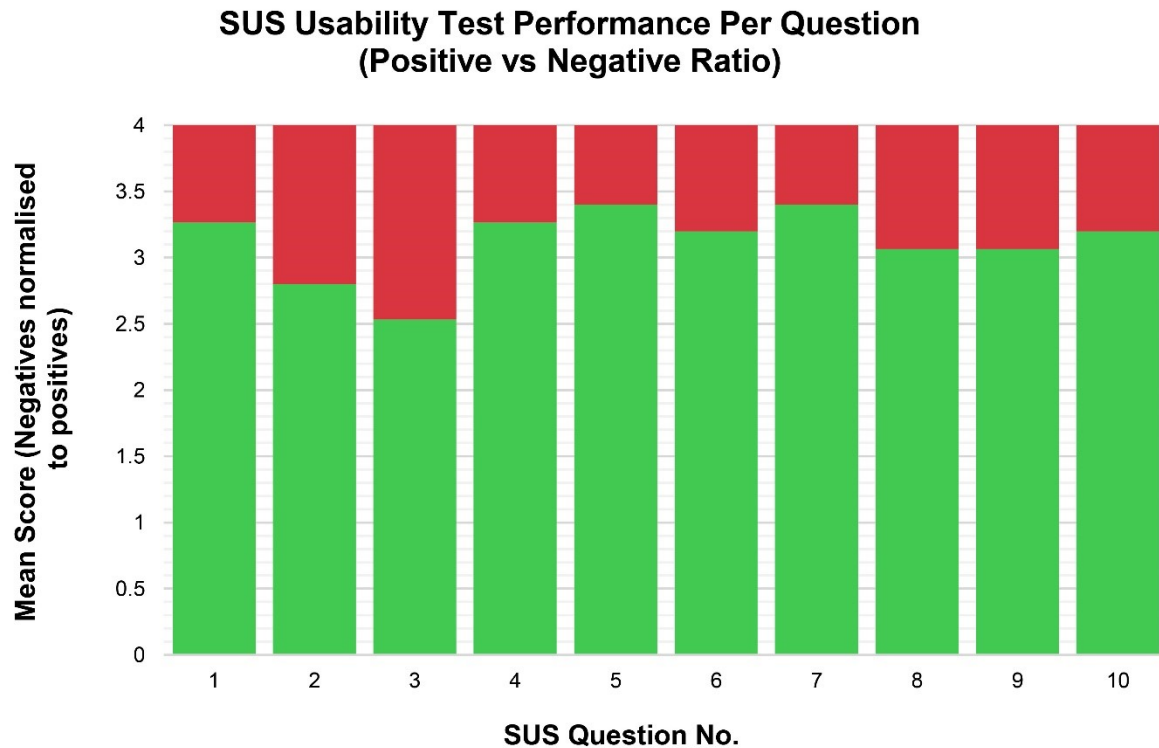


Figure 6.11 - SUS question mean scores, negative vs positive responses

Based on the previously discussed results, and observations from testing, the below usability aspects were highlighted:

Positive feedback

- Use of metaphors for page icons made navigating the site intuitive – participants were particularly fond of the animated home screen icons
- Many participants appreciated the option to change themes. This could be further developed to include other themes, such as high contrast to aid those with vision impairments
- The coloured sliders, and greying out of unneeded environmental controls made the settings page easier to use
- Navigating the dashboard was very positive, with participants noting that clicking a row to view details was intuitive
- The site's content was sectioned well into different pages and the layout was generally received positively

Negative feedback – modifications based on these areas are detailed in Appendix H

- Visited links were displayed in purple, which was difficult to see against a dark blue background
- The dashboard's sensor readings were coloured based on their relation to their targets. Many participants noted that the white text should be changed to green to represent the reading being on target, or inclusion of a traffic light type system.
- The graph on the log page was very crowded and difficult to read, and the dual Y-axes confused many participants or went unnoticed
- It was frequently mentioned that the site needed more labelling and contextual information, particularly on the settings page
- The horizontally stacked accordion on the settings page was difficult to find, a frequent suggestion was to change it to be vertically stacked
- The default HTML time input was used as it is accessible and supported across browsers, though many users had difficulty inputting the current time using this. A custom input could be created, though this was omitted as it would be less accessible.
- Greyed out gauges and actuator icons confused participants, who noted that they assumed this meant “broken” or “off.” This could be clarified to show “not needed”
- A “log out” link should be provided in the navigation bar, as this is where most people looked for it
- The dashboard summary view was crowded with information – a measure should be taken to separate each terrarium more clearly. This was particularly problematic for those with vision impairments.

6.4.2. HTML and CSS Validation

CSS was validated using the W3C CSS Validation Service [63]. All style sheets used for the web application were validated as “CSS level 3 + SVG,” with no errors. Many warnings were generated, though these were due to the use of browser prefixes. As these are proprietary, they are not technically valid against the specification.

HTML validation using the W3C Markup Validation Service [64] highlighted some errors which have since been addressed, including missing closing tags from PHP generated mark-up, missing alt tags on images, and invalid element nesting. Some errors have not been addressed due to time constraints, primarily duplicate IDs generated by PHP code and invalid use of the name attribute on certain inputs.

6.5. Testing Conclusion

This section has detailed testing undertaken on the system, and provided results in the form of tables and graphs. The following section discusses the results in further detail and evaluates the success of the project.

7. Evaluation and Discussion

The objectives of this project were met. The objectives were as follows:

- To conduct a literature review as secondary research
- To conduct informal interviews as primary research
- To design and develop an ESP8266 based monitoring and controlling system
- To design and develop a responsive and accessible web application to facilitate interacting with the hardware
- To test and evaluate the system against existing consumer products

The literature review and informal interview were successful and provided context for the project, though ideally there would have been further informal interviews throughout the project. In this sense, the literature review objective was fully met, though the informal interviews objective was not fully met due to social distancing restrictions. The control system was successful, having good temperature and humidity tolerances, though not within the initial, ambitious targets. This objective was largely well, having developed a functioning system which regulated temperature, humidity, and lighting, though suggestions for improvements were identified. Functional testing against an already existing product showed that the system performs favourably. This objective was largely met, though ideally the system would have been compared to more consumer products. The web application was tested and found to be easy to use and accessible. The objective of building a responsive and accessible web application to facilitate interaction with hardware was fully met, as shown in usability testing, though accessibility could be further improved. These tests combined show that the aim of creating an automatic monitoring and control system was met. Difficulties were met during the hardware development which were overcome through modifying the system's power delivery and modifying the node devices' code based on testing results. The main action to further meet the aim would be improving the control system to reduce tolerances.

The testing data showed that approaches discussed during the literature review can be successfully applied to a wider range of applications. It also showed that the increased focus on web interfaces compared to the solutions from the literature review was effective and worthwhile. The lack of further interviews, or interviews with other industry experts, did not affect the development of the project. However, it

affected the testing and evaluation of the deliverable. If more interviews were undertaken, the deliverable could have been evaluated in further depth, with further comparison to existing products and scenarios. Further interviews could also have provided insight to more improvements and future work.

7.1. Discussion of Results

7.1.1. Functional Testing Results

As is shown in Tables 6.1 and 6.2 in the Testing section, inclusion of an offset increased the minimum temperatures to remain closer to the target, while the maximum remained within the target range. The lowest of the minimum temperatures increased from 30.4°C to 30.9°C after modifying the code, while the highest maximum increased from 32.9°C to 33°C. Standard deviations of the two result groups were very close, showing that although the temperatures were consistently warmer, they were no more varied. As there were four mean values to represent each of the two test groups, a mean of these was calculated (the “grand mean” or “pooled mean”). The pooled mean before modifying the code was 31.52°C, whereas the pooled mean after modifying the code was 31.96°C. This shows the average temperatures were brought closer to the target of 32°C. In these tests, the mean was extremely close to the target, within margin of error. The range went marginally below the $\pm 1^\circ\text{C}$ target, at 0.1°C below.

The extended tests of 36 hours showed that the system performed well over an extended period of time. The primary change between this and the shorter runs was the higher variation – the maximum temperatures were higher (from 33°C to 33.4°C), and the minimums were lower (from 30.9°C to 30.7°C). These results were still close to the $\pm 1^\circ\text{C}$ target, but could be improved, though the pooled mean was still very close to the target, at 30.95°C. The extended test graph (Figure 6.4) shows that ambient temperature had an impact on the heating pad temperature. The heating pad temperature followed the same fluctuation pattern as the ambient temperature, though to a lesser extreme. When ambient temperature fluctuated by several degrees (up to 6°C in some tests), the heating pad temperature only fluctuated by several tenths of a degree, showing that the system is able to adapt to changing ambient temperatures. Higher ambient temperatures also resulted in faster heating

up of the heat pad, making for steeper and more frequent spikes of heating and cooling.

Compared to the Microclimate Ministat 100 – an existing product, this project's implementation performs well. During the 4 hour tests run on the Ministat, the minimum and maximum sensor readings were high, with the maximum exceeding the 32°C target by up to 2.5°C. A comparison of the Ministat performance and the project's implementation is shown below in Table 7.1:

Table 7.1 – Comparison of Ministat 100 and project (32°C target)

	Microclimate Ministat 100	Project implementation
Lowest minimum	31.6	30.9
Highest maximum	34.5	33
Mean	33.01	31.96
Highest SD	0.58	0.53

This table shows that the minimum and maximum values were far higher on the Microclimate tests, with the maximum greatly exceeding the target. The standard deviations of the two sets of data were comparable and within margin of error. As a result of the hotter temperatures, the Microclimate tests' overall mean was 1°C above the target, whereas the project's implementation was far closer, at 0.04°C under. This positive offset in the commercial thermostat is likely due to sensor inaccuracy, having a constant positive offset. Alternatively, the software may be causing this issue by reacting too late.

The system's humidity control was successful. Initial tests showed that the humidity remained close to $\pm 10\%$ of the 75% target, with the lowest minimum at 61% and the highest maximum at 79%. The pooled mean was 69.22%. As the humidity was overall below the target, the code was modified (described in Testing). Though the pooled mean was brought closer to the target, at 78.67%, the change had a detrimental effect. The maximum increased far over the target, reaching 90%. This shows that the system could be further improved, potentially using predictive

algorithms to avoid peaks. Fluctuations in room humidity of around 10% had no noticeable impact on the testing enclosure's humidity. Standard deviations were similar in both test groups, showing variation remained the same. However, one test run was anomalous (run 1, seen in the red line in Figure 6.6) in that its humidity kept rising past the target. At present, no reason has been found to explain this anomaly.

Combination temperature and humidity control was not effective. The temperature remained very close to its target, as with previous tests, with a pooled mean of 31.83°C – though the temperature cycle changed. As temperature overpowered the spraying system, the sprayer would activate when needed, and sprayed excess amounts of water, creating pools in the enclosure due to the humidity levels not rising. Once the humidity levels rose, they in turn overpowered the heating system, as is seen in test runs 3 and 4 in Figure 6.8, by the flattening and downward sloping of the temperature lines. However, humidity did not rise high enough to meet its target, with a pooled mean of 59.13% against the 75% target. This could be a problem with sensor placement, and not the system itself. As the humidity and temperature were acquired using the same sensor, it may be beneficial in future to incorporate a standalone humidity sensor. Unfortunately, this was not tested due to hardware faults, and time constraints preventing troubleshooting of these faults.

7.1.2. Usability Testing Results

The mean SUS score of the system was 78/100, 10 points above the average SUS score of 68 [61]. This shows that while the site does have usability flaws, it is above average according to the 15 participants.

Of the 15 participants' overall scores, the range was 57.5, with a minimum score of 40 and a maximum of 97.5. Supporting this wide range is a standard deviation of 16.1. It is hypothesised that the large range, and thus standard deviation, are attributed to human interpretations of the Likert scale, and of usability and quality. For example, the statement "I felt very confident using the system," expecting a Likert response, could be interpreted differently by each person. One person may have far stricter rules on what they consider "strongly agree" or "confident," than another, and may only select "agree" or "neutral." Another contributor to the range of scores is participants with vision impairments, who tended to rate the system more

negatively. This is a fair response, as the system made little consideration for those with vision impairments such as colour blindness, and is included in future work.

As is visualised in Figure 6.10 in Testing, many of the SUS statements had extremely varied responses. Half of the Likert statements (2, 3, 6, 8 and 10) had standard deviations of 1 or above – a large deviation on a scale of only 0-4. An interesting observation is that four of these five statements were negatively phrased, for example “I found the system very cumbersome to use.” This could imply that there were weaknesses in the application which some participants may have not factored in to their scoring. The lowest standard deviation was for statement 7, “I would imagine that most people would learn to use this system very quickly.” A frequent observation during the testing was that while many participants had difficulty using the system with no guidance, they noted that after asking for assistance, they found the system extremely straightforward to use. This supports the low standard deviation and agreeance of statement 7.

7.2. Recommendations and Suggestions

Some areas of this project would benefit from additional priority or time allocation, these are discussed below.

Hardware components could be adapted. The node devices could be changed to boards which can handle higher power, avoiding the need for additional power connectors. Solid State Relays (SSR) could have been used in place of the standard IoT-kit type electromagnetic relays, which would have longer lifespans while being silent, removing the clicking sound. Though the DHT22 sensors are reasonably accurate, this system would benefit from more accurate sensors, such as the AM2305. A further heating element could be supported in the form of a heat lamp. Control of a heat lamp to regulate ambient temperature of terrariums was planned, but not implemented due to time constraints.

The software development approach could be changed. A more object-oriented approach would benefit the web application, allowing for future expandability. The JavaScript could be made more object-oriented or procedural to avoid code duplication. The site could also have a manual override option to force the state of

actuators where necessary. Additionally, it could feature a care guides page, providing PDF documents for keepers to distribute to customers or for reference. Although the node devices' Arduino code would not benefit from object-orientation, it could have been made more memory efficient to facilitate future expansion, for example by replacing the Arduino "String" object with standard C strings or char arrays. The site development should have focused more on usability and accessibility, including allocating more time to responsiveness for different screen sizes. The log page could be expanded to show statistics like minimum, maximum and mean sensor readings, along with fixing some cache-related bugs.

8. Conclusions and Future Work

This section concludes the project, describing the work which was undertaken to meet the objectives and discussing areas for future work.

8.1. Future Work

This project could be expanded into many different areas as future work. Future work could involve further development of the core concept. The temperature and humidity target system attempted to create a constant, consistent temperature. In reality, a gradient should be introduced, with lower temperatures at night than during the day, and a varied temperature across the enclosure, as discussed by industry sources [65], [66]. Humidity should also vary over time, with periods of drier air. One of the complications of a constant or overly humid enclosure is respiratory infections in reptiles [67], [68]. Implementing this may also involve a dehumidifier to reduce humidity where necessary. Further development for future work would be to implement a dimming system as opposed to an on/off relay system. This would allow for far tighter tolerances and more accurate temperature control. An Artificial Intelligence approach could be used to predict sensor readings and act accordingly, along with a Fuzzy Logic approach to controlling dimmable actuators. As discussed in the introduction, many animals' eggs are temperature-sensitive. This could be an entirely separate project due to the specifics of egg incubation for various species.

This project has been developed specifically for reptiles, and for lizards as a subset of these. This was decided because a system for all reptiles would be outside the scope of this project. However, it could be expanded to suit other reptiles' needs. A system for snakes would need little modification. With a greater focus on humidity, the system would also be suitable for amphibians. Some reptiles, such as terrapins and larger lizards such as Monitors, require heated bodies of water, an area which could be addressed using similar methods to those in this project. This could also be applied to fish, benefitting hugely from a system similar to this project's implementation, for automatically controlling water quality. Freshwater tropical fish could benefit from monitoring and controlling of water temperature, acidity (pH), general hardness (GH), and Ammonia (NH_3 and NH_4^+) levels [69], to name just a few environmental variables. On the opposite end of the spectrum to reptiles, many

animals require constant cold environments. Emperor Penguins, for example, require ambient temperatures between -6°C - 0°C , and water temperatures of 1°C - 7°C [70], which could be controlled using different actuators.

Another possibility for future work is to take a more computer networks based approach, focusing on aspects such as custom socket programming or developing a bespoke network protocol to better suit the project's needs, or to move to Zigbee instead of Wi-Fi, which would allow for more sophisticated network configurations such as mesh networks, and would reduce power consumption. This could allow for the project to be scaled up for larger applications, such as in zoos or even applied to more general smart cities concepts. A further component of scaling up the project would be to incorporate push notifications, as opposed to client-side JavaScript notifications.

8.2. Conclusion

Overall, the project is considered successful. This project has consisted of research, design, development, and testing of IoT and web applications.

A literature review was conducted at the beginning of the project, with a focus on IoT applications for caring for animals and monitoring environments, with web-enabled features. An informal interview was conducted to gain understanding and context for the project background. An IoT control system was designed and developed using the ESP8266 prototype board, along with a PHP-based progressive web application for monitoring and controlling the system. The system was evaluated using user experience tests and functional testing of its environment control. The temperature control was found to be very effective, and very close to the $\pm 1^{\circ}\text{C}$ target. Standalone humidity control was effective, and also close to its $\pm 10\%$ target. Combined temperature and humidity control was unsuccessful, as the heating element overpowered the humidity. The system was tested in comparison to a commercial product, and found to control temperature better, being closer to the target. Ideally, more commercial systems would have been tested, however, this was omitted due to time constraints. In usability testing, the web application was generally very well received, scoring above average on the System Usability Scale. Usability testing also identified areas for improvement of the application. Unfortunately, later

interviews with industry experts reviewing the system were not conducted due to COVID-19 social distancing restrictions. This limitation also applied to usability testing – ideally a larger sample would have been used.

9. Reflection

I began this project long before I even knew what it would become. Learning IoT development during my final year of college, and first two years at University, I created a simple web site showing live sensor readings. As my knowledge grew through levels 4 and 5, I began applying ideas from our modules to the project. At the time of the project proposal, it was obvious what area I wanted to go into, as it would give me an opportunity to further develop my idea with newly acquired knowledge, guided by experts. Having now completed the project, I am still learning things which could have been applied to the project, and likely will be outside of academic endeavours. I am glad that I decided to continue with an idea I had already considered previously, as I had background knowledge, such as reptile care requirements, which allowed me to focus on the important, new areas of the project. The literature review gave me an excellent opportunity to gain academic context and review existing work by other academics, whom I could follow in the footsteps of, applying their findings and ideas to my own project. Having researched and developed this idea in an academic context, I now have a far better idea of what area I intend to work in. Before attending University, I expected to go into some form of programming, with no specifics. This project has given me an ambition to work in IoT and/or web development.

I have learned a lot throughout this project. I have worked in subject areas outside of my knowledge and comfort zone, for example working with basic electronics, and expanding greatly upon my knowledge of web development technologies, particularly JavaScript and PHP, and working with Progressive Web Apps. In future, it would be beneficial to do more preparation and background reading in areas which are new to me – for example, I could have spent time over the summer learning electronics ready for the project, as opposed to starting my development early. The project has also given me an opportunity to collaborate, share ideas, and pass on my knowledge gained through the project to others in my cohort, as they have done to me.

The project has also given me a great opportunity to put into practice the knowledge and skills I have learned during my time on the degree programme, particularly from Smart and Connected Devices 1, web development and generic programming modules. I have also developed many “soft skills” and project management skills,

which will potentially be as useful as the technical skills. One such skill is time management. Time management becomes crucial on a project of this scale, and while I feel that I managed time well, I should have balanced the work out more and worked on smaller tasks in a more organised fashion, using a technique such as timeboxing. This project and its various forms of assessment have also allowed me to develop my visual, written and verbal communication – the latter two of which have been major weaknesses in the past. These skills will be transferable to employment and other areas of life. One thing which I would do very differently in future projects is to be more thorough in my planning and project structure.

This project had an initial plan and timeline, but I had not planned enough for software development, which resulted in fixing unnecessary problems as they arose, as opposed to avoiding them from the beginning. In this sense, I took more of a software development approach, rather than a software engineering approach – which was definitely detrimental. I have enjoyed working with both of my supervisors, who I am extremely grateful to. Working with them has helped my organisation throughout the project, constantly checking progress and setting targets. With their guidance, I become more independent in my work - another transferable skill. It has been very satisfying and rewarding to work on an extended project and see it through to the end. It has given me confidence in myself, proving that I can complete a project of such a scale independently, and motivated me to continue my professional development as a software engineer.

References

- [1] Victoria State Government, "Code of Practice for the Welfare of Animals - Private Keeping of Reptiles," 2017. [Online]. Available: <http://agriculture.vic.gov.au/pets/other-pets/reptiles/code-of-practice-for-the-private-keeping-of-reptiles>. [Accessed: 23-Oct-2019].
- [2] UK Parliament, "Good Practice Guidelines For the Welfare of Privately Kept Reptiles & Amphibians," 2014. [Online]. Available: http://data.parliament.uk/DepositedPapers/Files/DEP2015-0315/Good_Practice_Guidelines_For_the_Welfare_of_Privately_Kept_Reptiles.pdf. [Accessed: 22-Oct-2019].
- [3] J. Van Demark, "Reptiles & Amphibians, Part I: Housing, Heat and Humidity," 2018. [Online]. Available: <https://rhinebeckanimalhospital.com/blog/42506-reptiles-amp-amphibians-part-i-housing-heat-and-humidity>. [Accessed: 23-Oct-2019].
- [4] C. Warwick, P. Arena, S. Lindley, M. Jessop, and C. Steedman, "Assessing reptile welfare using behavioural criteria," *Heal. Welf. Captiv. Reptil.*, vol. 35, no. 3, pp. 123–131, 2013.
- [5] J. J. Amiel and R. Shine, "Hotter nests produce smarter young lizards," *Biol. Lett.*, vol. 8, no. 3, pp. 372–374, 2012.
- [6] H. Siviter, D. C. Deeming, and A. Wilkinson, "Egg incubation temperature influences the growth and foraging behaviour of juvenile lizards," *Behav. Processes*, vol. 165, no. February, pp. 9–13, 2019.
- [7] Amazon, "Exo Terra PT2057 Light Dome Fixture, 18 cm: Exo Terra: Amazon.co.uk: Pet Supplies." [Online]. Available: https://www.amazon.co.uk/Exo-Terra-PT2057-Light-Fixture/dp/B00410U75O/ref=sr_1_6?keywords=reptile+heat+lamp&qid=1580681841&sr=8-6. [Accessed: 02-Feb-2020].
- [8] Komodo, "Advanced Cloth Heat Mat 3W." [Online]. Available: <https://www.komodoproducts.com/products/heating/advanced-cloth-heat-mat.html>. [Accessed: 02-Feb-2020].
- [9] Amazon, "AllC100 5.0 26W UVB Bulb for Reptile High UVB Output Tropical Compact Fluorescent Bulb forAmphibian Tortoise Lizard Succulent Plants Improve D3 Synthesis Increase Calcium Absorption (26W): Amazon.co.uk: Pet Supplies." [Online]. Available: https://www.amazon.co.uk/AllC100-Fluorescent-forAmphibian-Succulent-Absorption/dp/B07PJP2Z38/ref=sr_1_2_sspa?keywords=reptile+uv&qid=1580683028&sr=8-2-spons&psc=1&spLa=ZW5jcnlwdGVkUXVhbGlmaWVyPUEzTVBYREZFRkxDN

jZLJmVuY3J5cHRIZEIkPUEwMTE5NjM0MIEyUVVDMDg3TlowTiZlbnNyeXB0ZWRBZEIkPUEwNjc3MTQ4MjNUVks2OFNWRURMVSZ3aWRnZXROYW1IPXNwX2F0ZiZhY3Rpb249Y2xpY2tSZWRpcmVjdCZkb05vdExvZ0NsaWNrPXRydWU=. [Accessed: 02-Feb-2020].

- [10] Cambridge University Press, "TERRARIUM | meaning in the Cambridge English Dictionary," 2019. [Online]. Available: <https://dictionary.cambridge.org/dictionary/english/terrarium>. [Accessed: 27-Oct-2019].
- [11] UK Parliament, "Animal Welfare Act 2006," 2006. [Online]. Available: http://www.legislation.gov.uk/ukpga/2006/45/pdfs/ukpga_20060045_en.pdf.
- [12] P. Santos, J. Alves, and P. Carneiro, "Monitoring and control of a reptile terrarium," in *Proceedings of 2014 11th International Conference on Remote Engineering and Virtual Instrumentation, REV 2014*, 2014, pp. 438–439.
- [13] K. Lewis, "How to build a better bed: IoT and AI at the zoo - Internet of Things blog," 2017. [Online]. Available: <https://www.ibm.com/blogs/internet-of-things/iot-how-to-build-a-better-zoo-bed/>. [Accessed: 27-Oct-2019].
- [14] E. Henry, V. Adamchuk, T. Stanhope, C. Buddle, and N. Rindlaub, "Precision apiculture: Development of a wireless sensor network for honeybee hives," *Comput. Electron. Agric.*, vol. 156, no. 3, pp. 138–144, Jan. 2019.
- [15] A. Kviesis and A. Zacepins, "System Architectures for Real-time Bee Colony Temperature Monitoring," *Procedia Comput. Sci.*, vol. 43, pp. 86–94, Jan. 2015.
- [16] A. Zacepins, A. Kviesis, E. Stalidzans, M. Liepniece, and J. Meitalovs, "Remote detection of the swarming of honey bee colonies by single-point temperature monitoring," *Biosyst. Eng.*, vol. 148, no. 2016, pp. 76–80, 2016.
- [17] H. Li, H. Wang, W. Yin, Y. Li, Y. Qian, and F. Hu, "Development of a remote monitoring system for henhouse environment based on IoT technology," *Futur. Internet*, vol. 7, no. 3, pp. 329–341, 2015.
- [18] B. Shi, V. Sreeram, D. Zhao, S. Duan, and J. Jiang, "A wireless sensor network-based monitoring system for freshwater fishpond aquaculture," *Biosyst. Eng.*, vol. 172, pp. 57–66, 2018.
- [19] C. Tziortzioti, D. Amaxilatis, I. Mavrommati, and I. Chatzigiannakis, "IoT sensors in sea water environment: Ahoy! Experiences from a short summer trial," *Electron. Notes Theor. Comput. Sci.*, vol. 343, no. 2019, pp. 117–130, 2019.
- [20] L. T. Beng, P. B. Kiat, L. N. Meng, and P. N. Cheng, "Field testing of IoT devices for livestock monitoring using Wireless Sensor Network, near field

- communication and Wireless Power Transfer,” in *2016 IEEE Conference on Technologies for Sustainability (SusTech)*, 2016, pp. 169–173.
- [21] The Wireless Power Consortium, “The Qi Wireless Power Transfer System - Power Class 0 Specification - Parts 1 and 2: Interface Definitions,” 2017.
 - [22] P. V. Vimal and K. S. Shivaprakasha, “IoT based greenhouse environment monitoring and controlling system using Arduino platform,” in *2017 International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICT)*, 2017, pp. 1514–1519.
 - [23] Mouser Electronics, “DHT11 Humidity & Temperature Sensor.” [Online]. Available: <https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf>. [Accessed: 13-Dec-2019].
 - [24] S. N. Nnadi and F. E. Idachaba, “Design and Implementation of a Sustainable IOT Enabled Greenhouse Prototype,” in *2018 IEEE 5G World Forum (5GWF)*, 2018, pp. 457–461.
 - [25] S. Anupamashree, C. A. Gowri Priya, P. Harshitha, K. K. V, and S. Puneeth, “Development of IoT Based Remote Monitoring System for Air and Water Pollutants in Industrial Environment,” *i-Manager’s J. Wirel. Commun. Networks*, vol. 7, no. 1, pp. 34–40, 2019.
 - [26] J. Muangprathub, N. Boonnam, S. Kajornkasirat, N. Lekbangpong, A. Wanichsombat, and P. Nillaor, “IoT and agriculture data analysis for smart farm,” *Comput. Electron. Agric.*, vol. 156, no. June 2018, pp. 467–474, 2019.
 - [27] MICROclimate, “MiniStat 100 - Microclimate.” [Online]. Available: <https://www.microclimate.co.uk/product/ministat-100/>. [Accessed: 09-Nov-2019].
 - [28] MICROclimate, “Pulse B2 HT - Microclimate.” [Online]. Available: <https://www.microclimate.co.uk/product/pulse-b2-ht/>. [Accessed: 20-Mar-2020].
 - [29] MICROclimate, “MiniStat 300 - Microclimate.” [Online]. Available: <https://www.microclimate.co.uk/product/ministat-300/>. [Accessed: 20-Mar-2020].
 - [30] MICROclimate, “EVO - Microclimate.” [Online]. Available: <https://www.microclimate.co.uk/product/microclimate-evo/>. [Accessed: 21-Mar-2020].
 - [31] MICROclimate, “EVO Lite - Microclimate.” [Online]. Available: <https://www.microclimate.co.uk/product/evo-lite/>. [Accessed: 25-Mar-2020].
 - [32] Happy Pet Products, “Komodo On/Off Thermostat | Reptile | Happy Pet,” 2014. [Online]. Available: <https://www.komodoproducts.com/komodo-on-off-thermostat.html>. [Accessed: 09-Nov-2019].

- [33] Happy Pet Products, "Komodo Pulse Thermostat | Reptile | Happy Pet," 2014. [Online]. Available: <https://www.komodoproducts.com/products/monitoring-komodo/komodo-pulse-thermostat.html>. [Accessed: 22-Mar-2020].
- [34] I. Zoo Med Laboratories, "Repti Rain® Automatic Misting Machine | Zoo Med Laboratories, Inc.," 2018. [Online]. Available: <https://zoomed.com/repti-rain-automatic-misting-machine/>. [Accessed: 09-Nov-2019].
- [35] J. Collis, *Business research : a practical guide for undergraduate and postgraduate students.*, 3rd ed. / . Basingstoke: Palgrave Macmillan, 2008.
- [36] Z. Nawaz, S. Aftab, and F. Anwer, "Simplified FDD Process Model," *Int. J. Mod. Educ. Comput. Sci.*, vol. 9, no. 9, pp. 53–59, 2017.
- [37] S. Aftab *et al.*, "Using FDD for small project: An empirical case study," *Int. J. Adv. Comput. Sci. Appl.*, vol. 10, no. 3, pp. 151–158, 2019.
- [38] Inflectra, "Feature Driven Development," 2020. [Online]. Available: <https://www.inflectra.com/ideas/topic/feature-driven-development.aspx>. [Accessed: 14-Feb-2020].
- [39] Scott W. Ambler, "Feature Driven Development (FDD) and Agile Modeling," 2018. [Online]. Available: <http://www.agilemodeling.com/essays/fdd.htm>. [Accessed: 22-Feb-2020].
- [40] F. Siddiqui and M. Alam, "Ontology Based Feature Driven Development Life Cycle," *Int. J. Comput. Sci. Issues*, vol. 9, no. 1, 2012.
- [41] wemos.cc, "LOLIN D1 mini — WEMOS documentation," 2019. [Online]. Available: https://docs.wemos.cc/en/latest/d1/d1_mini.html. [Accessed: 08-Mar-2020].
- [42] Arduino AG, "Arduino - FAQ," 2019. [Online]. Available: <https://www.arduino.cc/en/main/FAQ>. [Accessed: 06-Oct-2019].
- [43] Michael Miller, "Rtc, v2.3.4." 2020.
- [44] Bernd Giesecke, "DHTesp, v1.17." 2019.
- [45] L. Aosong(Guangzhou) Electronics Co., "AM2302 Product Manual."
- [46] The jQuery Foundation, "jQuery, v4.4.0." 2020.
- [47] Mozilla, "JSON.parse() - JavaScript | MDN," 2020. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/parse. [Accessed: 08-Mar-2020].
- [48] The PHP Group, "PHP: PDO - Manual." [Online]. Available: <https://www.php.net/manual/en/book.pdo.php>. [Accessed: 08-Mar-2020].

- [49] Google, "Charts | Google Developers." [Online]. Available: <https://developers.google.com/chart>. [Accessed: 31-Mar-2020].
- [50] Google Developers, "Progressive Web Apps | Google Developers," 2019. [Online]. Available: <https://developers.google.com/web/progressive-web-apps>. [Accessed: 16-Feb-2020].
- [51] T. Ater, *Building Progressive Web Apps*. California: O'Reilly, 2017.
- [52] D. Sheppard, "Service Workers," in *Beginning progressive web app development creating a native app experience on the web*, Illinois: Apress Media, 2017.
- [53] Symantec, "What is an SSL certificate?," 2019. [Online]. Available: <https://uk.norton.com/internetsecurity-how-to-ssl-certificates-what-consumers-need-to-know.html>. [Accessed: 16-Feb-2020].
- [54] cppreference.com, "time_t - cppreference.com," 2019. [Online]. Available: https://en.cppreference.com/w/c/chrono/time_t. [Accessed: 24-Feb-2020].
- [55] DigiCert, "SAN Certificates: Subject Alternative Name – Multi-Domain (SAN)." [Online]. Available: <https://www.digicert.com/subject-alternative-name.htm>. [Accessed: 24-Feb-2020].
- [56] U.S. Department of Health & Human Services, "User Experience Basics | Usability.gov," 2020. [Online]. Available: <https://www.usability.gov/what-and-why/user-experience.html#>. [Accessed: 19-Feb-2020].
- [57] O. Sohaib, W. Hussain, and M. K. Badini, "User Experience (UX) and the Web Accessibility Standards," vol. 8, no. 3, pp. 584–588, 2011.
- [58] Shawn Lawton Henry, "Introduction to Web Accessibility | Web Accessibility Initiative (WAI) | W3C," 2019. [Online]. Available: <https://www.w3.org/WAI/fundamentals/accessibility-intro/#what>. [Accessed: 20-Feb-2020].
- [59] W3C, "Web Content Accessibility Guidelines (WCAG) 2.1," 2018. [Online]. Available: <https://www.w3.org/TR/WCAG21/>. [Accessed: 15-Jan-2019].
- [60] GOV.UK, "Making your service accessible: an introduction - Service Manual - GOV.UK," 2019. [Online]. Available: <https://www.gov.uk/service-manual/helping-people-to-use-your-service/making-your-service-accessible-an-introduction#meeting-government-accessibility-requirements>. [Accessed: 20-Feb-2020].
- [61] U. S. D. of H. & H. Services, "System Usability Scale (SUS) | Usability.gov." [Online]. Available: <https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html>. [Accessed: 20-Feb-2020].

- [62] J. Brooke, "SUS-A quick and dirty usability scale," 1996.
- [63] World Wide Web Consortium, "The W3C CSS Validation Service," 2009. [Online]. Available: <https://jigsaw.w3.org/css-validator/>. [Accessed: 30-Nov-2018].
- [64] World Wide Web Consortium, "The W3C Markup Validation Service," 2013. [Online]. Available: <https://validator.w3.org/>. [Accessed: 30-Nov-2018].
- [65] Zilla-Rules, "How to Create a Proper Day & Night Cycle in Your Terrarium," 2016. [Online]. Available: <https://www.zillarules.com/articles/how-to-create-a-proper-day-and-night-cycle-in-your-terrarium>. [Accessed: 24-Mar-2020].
- [66] RSPCA, "Leopard gecko care sheet | Advice for owners | RSPCA," 2020. [Online]. Available: <https://www.rspca.org.uk/adviceandwelfare/pets/other/leopardgecko>. [Accessed: 24-Mar-2020].
- [67] J. Schumacher, "Reptile respiratory medicine," *Veterinary Clinics of North America - Exotic Animal Practice*, vol. 6, no. 1. Elsevier, pp. 213–231, 01-Jan-2003.
- [68] Avian and Exotic Animal Clinic of Arizona, "Respiratory Disease in Snakes," 2015.
- [69] Aqueon, "Freshwater Aquarium Water Quality: The Nitrogen Cycle & Optimal Water Chemistry," 2017. [Online]. Available: <https://www.aqueon.com/articles/freshwater-aquarium-water-quality>. [Accessed: 13-Dec-2019].
- [70] AZA Penguin Taxon Advisory Group, "Penguin (Spheniscidae) Care Manual," 2014.

Bibliography

- A. F. Santamaria, P. Raimondo, F. De Rango, and A. Serianni, "A two stages fuzzy logic approach for Internet of Things (IoT) wearable devices," 2016, pp. 1–6.
- B. Frain, *Responsive Web Design with HTML5 and CSS3*, 2nd ed. Birmingham: Packt Publishing, 2015.
- B. Stroustrup, *The C ++ Programming Language*, 4th ed. Michigan: Pearson Education, 2013.
- C. Love, *Progressive Web Application Development by Example*. Birmingham: Packt Publishing, 2018.
- D. A. Hume, *Progressive Web Apps*. New York: Manning, 2018.
- Espressif Systems, "ESP8266 Low Power Solutions," 2016.
- Espressif Systems, "ESP8266 Technical Reference," 2019.
- GOV.UK, "Understanding WCAG 2.1 - Service Manual - GOV.UK," 2018. [Online]. Available: <https://www.gov.uk/service-manual/helping-people-to-use-your-service/understanding-wcag>. [Accessed: 19-Apr-2020].
- Jordi Serra, David Pubill, Angelos Antonopoulos, and Christos Verikoukis, "Smart HVAC Control in IoT: Energy Consumption Minimization with User Comfort Constraints," vol. 2014, pp. 1–11, 2014.
- K. Karvinen and T. Karvinen, *Make: Getting Started with Sensors*. California: Maker Media, 2014.
- M. A. Khan and K. Salah, "IoT security: Review, blockchain solutions, and open challenges," *Futur. Gener. Comput. Syst.*, vol. 82, no. 2018, pp. 395–411, 2018.
- Mozilla, "Web Thing API," 2019. [Online]. Available: <https://iot.mozilla.org/wot/>. [Accessed: 26-Nov-2019].
- S. Krug, *Don't Make Me Think, Revisited A Common Sense Approach to Web Usability*, 3rd ed. Berkeley: New Riders, 2014.

World Wide Web Consortium, "Web Content Accessibility Guidelines (WCAG) 2.1," 2018. [Online]. Available: <https://www.w3.org/TR/WCAG21/>. [Accessed: 24-Nov-2018].

Appendices

Appendix A – Project Timeline

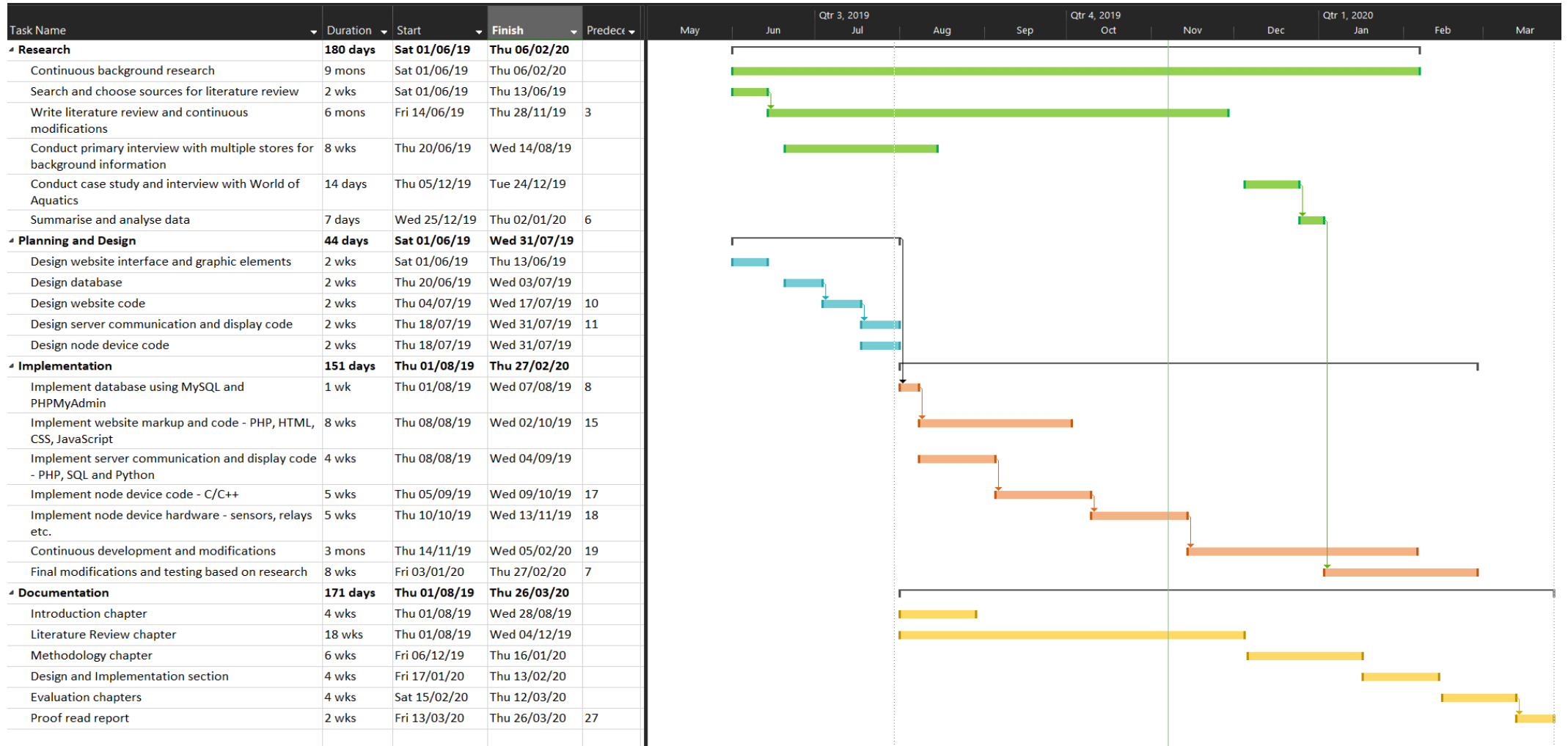


Figure A-1 – Project timeline Gantt chart

Appendix B – Hardware and Software Resources

Hardware

The below hardware will be used in the development of the system:

- A Raspberry Pi 3B will be used as a central server. A Raspberry Pi was chosen for a central server as the needs of this project do not require a standalone desktop or laptop pc to act as a server, and the Raspberry Pi has Wi-Fi and is capable of hosting a MySQL database and web server running PHP.
- The Wemos D1 Mini development board, based on the ESP8266 microcontroller, will be used for node devices. The D1 mini was chosen because it has a very small footprint while having Wi-Fi capability
- The DHT22 (AM2302) temperature and humidity sensor was chosen as it inexpensive and relatively accurate, having a temperature accuracy of $\pm 0.5^{\circ}\text{C}$ and a relative humidity accuracy of $\pm 2\%$. These accuracies are appropriate for a prototype system. The temperature threshold will be $\pm 1^{\circ}\text{C}$ of the target temperature for this system.
- A real-time clock – The DS3231 was chosen due to its low price and accuracy. An RTC was used instead of retrieving time from a server as it makes the node devices more dependent, allowing for future modifications
- Four relays will be needed to enable or disable the actuators
- Actuators:
 - A 12/24v heating lamp, powered by a bench power supply from the electronics lab
 - A 5V LED representing a higher-power ultraviolet bulb (USB powered)
 - A 5V water pump for use as a sprayer (USB powered)
 - A 5V heating pad (USB powered)

Languages and data formats

The proposed system will use the following languages and formats:

- The Arduino programming language will be used to program the node microcontrollers (D1 Mini). This was chosen both due to familiarity, and as Arduino is a subset of C and C++ functions, which runs faster than Python, the primary alternative language.
 - The Rtc Library for interfacing with the real-time clock
 - The DHTesp library for reading the DHT22 sensors
 - Standard Arduino libraries: SPI, Wire, ctime, cstdlib, math
- PHP will be used to display data on web pages and to send or retrieve data from node devices and a database. PHP was chosen as it has database access functions built-in, including prepared statements for more secure database manipulation.
- JavaScript will be used to provide additional functionality on web pages such as button click events and live data feeds using Asynchronous JavaScript And XML (AJAX).
 - The jQuery library will be used to simplify interacting with HTML elements and to provide the AJAX functionality, allowing for dynamic page updates
 - The Google Charts API will be used to graph information on the website, as developing a graphing library from scratch is outside of the scope of this project
- The JavaScript Object Notation (JSON) format will be used to temporarily store data on the server to be accessed by the website. This was chosen as it is very simple to access JSON data through JavaScript due to its inbuilt json parsing function
- The Comma Separated Value (CSV) format will be used to store log data for users to download. This was chosen as it is simple to implement and is readable by many software packages including Microsoft Excel. Excel is also accessible to everyone, so potential clients will be able to read the data.
- HTML and Cascading Style Sheets CSS will be used to display and style information and interfaces to the user

- The Scalable Vector Graphics (SVG) format will be used to present graphics and icons to the user. SVG was chosen as it is vector based, which gives two advantages over raster graphics: Infinitely scalable graphics, and smaller file sizes for simple files which results in faster page loading.
- SQL will be used to query a MySQL database to save and retrieve data

Software installations

Development of the system will use the following software:

- Adobe Illustrator CC 2018 will be used to create vector graphics for the web interface and for page wireframes, seen later in section 4
- Microsoft Visio 2018 and Project 2016 will be used for project planning and design
- MySQL will be used for the Database Management System (DBMS) which will run on the server. This was chosen as it has a web GUI (phpMyAdmin) which can be accessed remotely for administration, and is easily interfaced with PHP, for example using the PDO class
- XAMPP will be used to run an Apache web server during development before deploying to the server
- Microsoft VS Code will be used for the web development and Arduino development using the Arduino extension

Appendix C – Page Wireframes

Home

A desktop wireframe of a web application's home page. At the top, there is a horizontal navigation bar with four buttons: 'Home', 'Dashboard', 'Log', and 'Settings'. The main content area is centered and contains the text 'Welcome, guest!' followed by 'Log in:'. Below this are two input fields, one for 'Username:' and one for 'Password:'. A 'Log in' button is positioned below the password field. Underneath the button is the text 'Or [sign up](#)'. At the bottom of the page, there is a small copyright notice: '© Austin Hooper'.

Figure C-1 – Desktop: home logged out

A mobile wireframe of the same web application's home page. It features a hamburger menu icon in the top left corner. The main content area is centered and contains the text 'Welcome, guest!' followed by 'Log in:'. Below this are two input fields, one for 'Username:' and one for 'Password:'. A 'Log in' button is positioned below the password field. Underneath the button is the text 'Or [sign up](#)'. At the bottom of the page, there is a small copyright notice: '© Austin Hooper'.

Figure C-2 – Mobile: home logged out

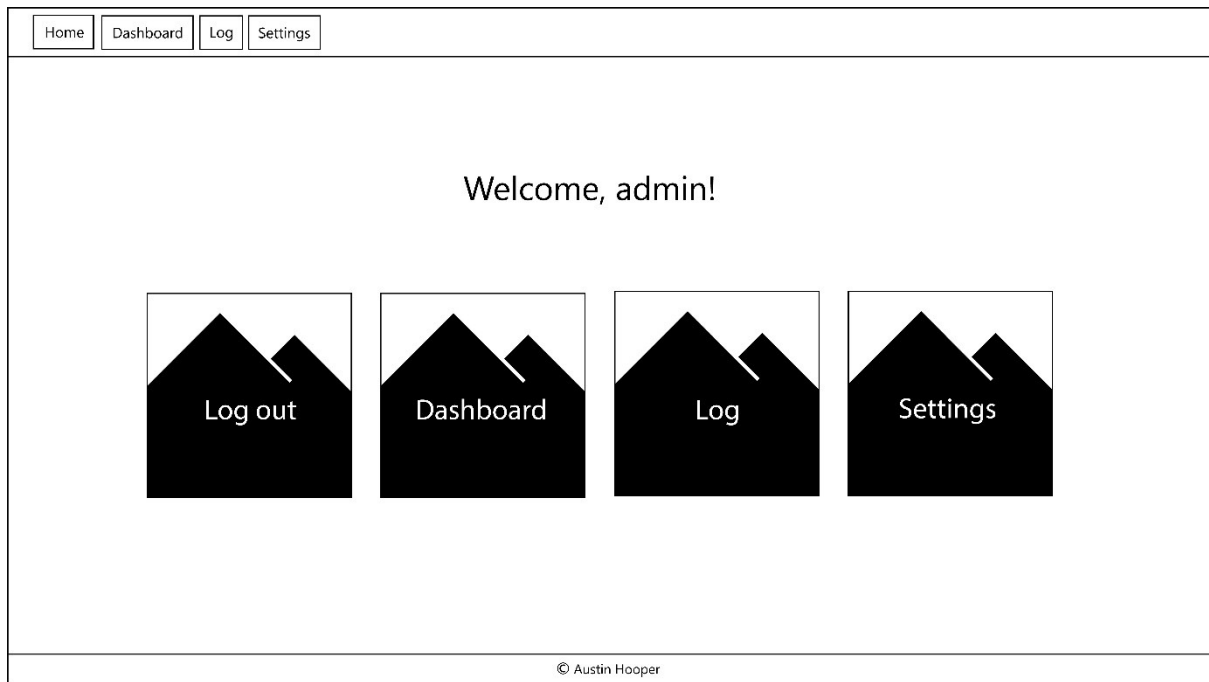


Figure C-3 – Desktop: home logged in

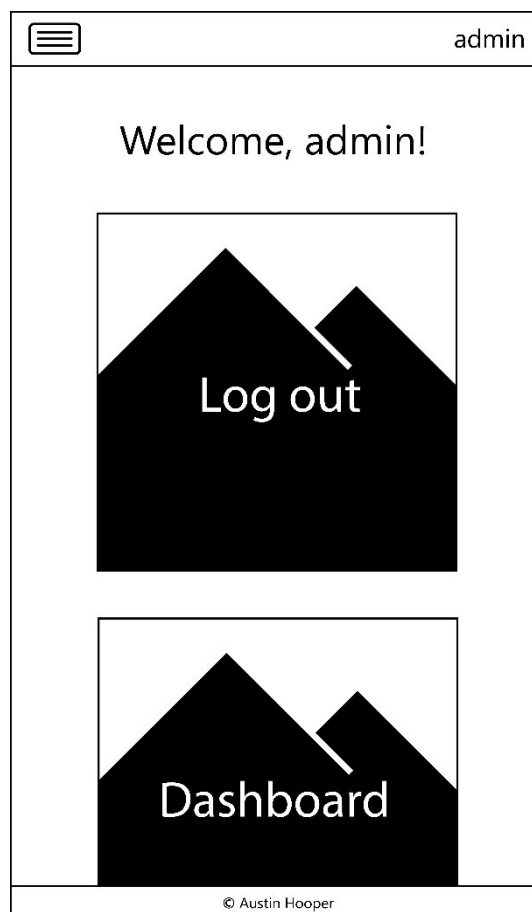


Figure C-4 – Mobile: home logged in

Sign up

Home	Dashboard	Log	Settings
------	-----------	-----	----------

Sign up:

Username:

Email:

Password:

© Austin Hooper

Figure C-5 – Desktop: sign up



Sign up:

Username:

Email:

Password:

© Austin Hooper

Figure C-6 – Mobile: sign up

Dashboard

Home

Dashboard

Log

Settings

All

All Enclosures

Bearded Dragons E...
(Bearded Dragon)

Basking spot
30.5°C

Heat mat
29.2°C

Humidity
70%

OFF

ON
(08:30 - 20:30)

ON

Bearded Dragons E...
(Bearded Dragon)

Basking spot
30.1°C

Heat mat
29.7°C

Humidity
80%

ON

ON
(08:30 - 20:30)

ON

Day Geckos Terrari...
(Day Gecko)

Basking spot
31.2°C

Heat mat
33.8°C

Humidity
74%

OFF

ON
(08:30 - 20:30)

OFF

Leopard Geckos Terr...
(Leopard Gecko)

Basking spot
31.1°C

Heat mat
26.2°C

Humidity
56%

ON

N/A

ON

Crested Geckos Terr...
(Crested Gecko)

Basking spot
30.9°C

Heat mat
29.2°C

Humidity
75%

ON

N/A

ON

© Austin Hooper

Figure C-7 – Desktop: dashboard summary view

		
All Enclosures		
Bearded Dragons Tank 1 (Bearded Dragon)		
Basking spot 30.5°C	Heat mat 29.2°C	Humidity 70%
 OFF	 ON	
	ON (08:30 - 20:30)	
Bearded Dragons Tank 2 (Bearded Dragon)		
Basking spot 30.5°C	Heat mat 29.2°C	Humidity 70%
© Austin Hooper		

Figure C-8 – Mobile: dashboard summary view

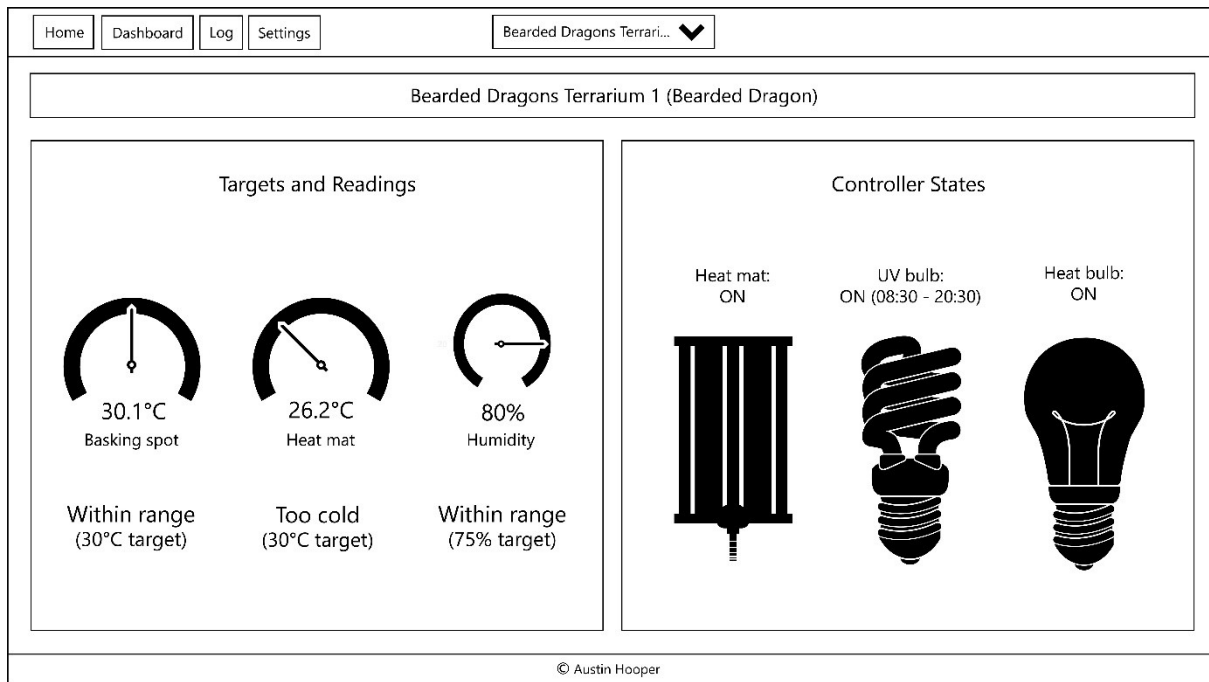


Figure C-9 – Desktop: dashboard individual view

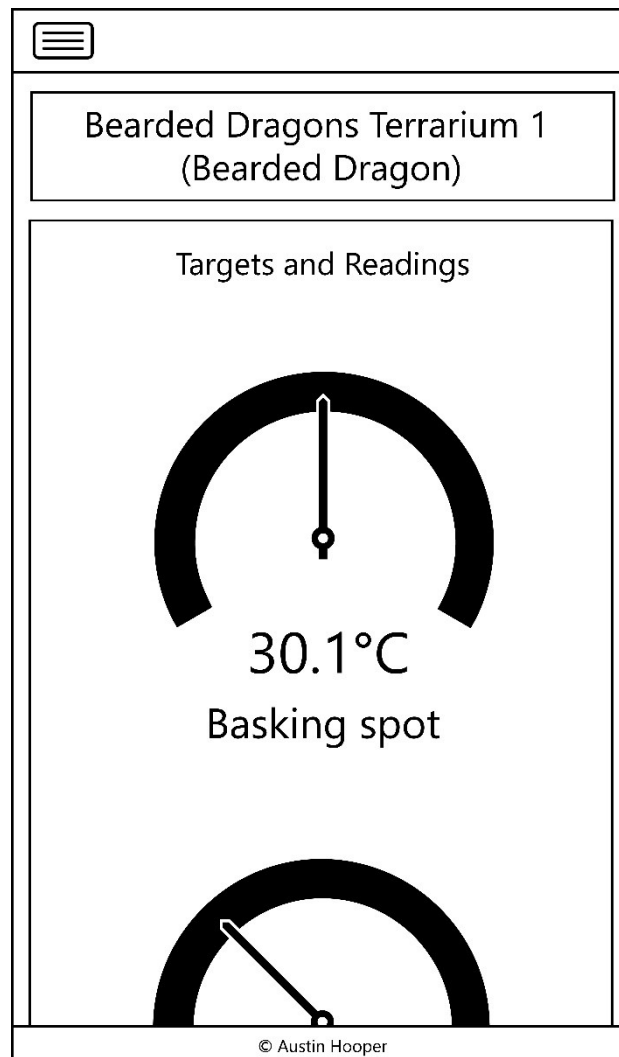


Figure C-10 – Mobile: dashboard individual view

Log

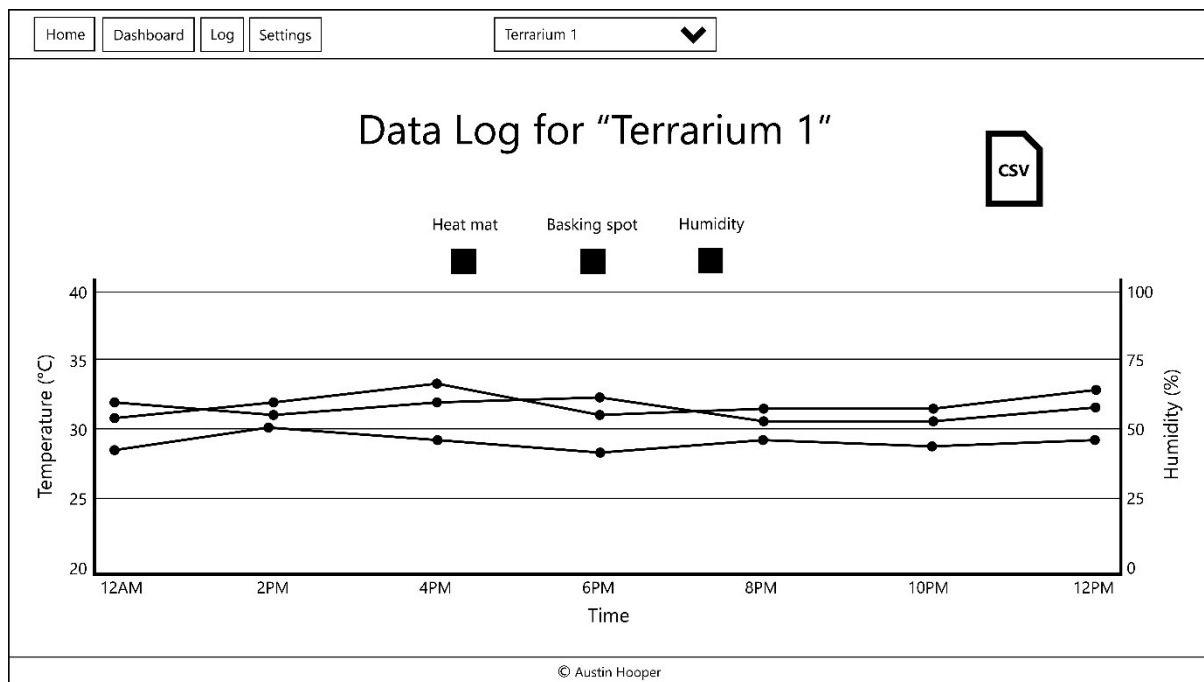


Figure C-11 – Desktop: log

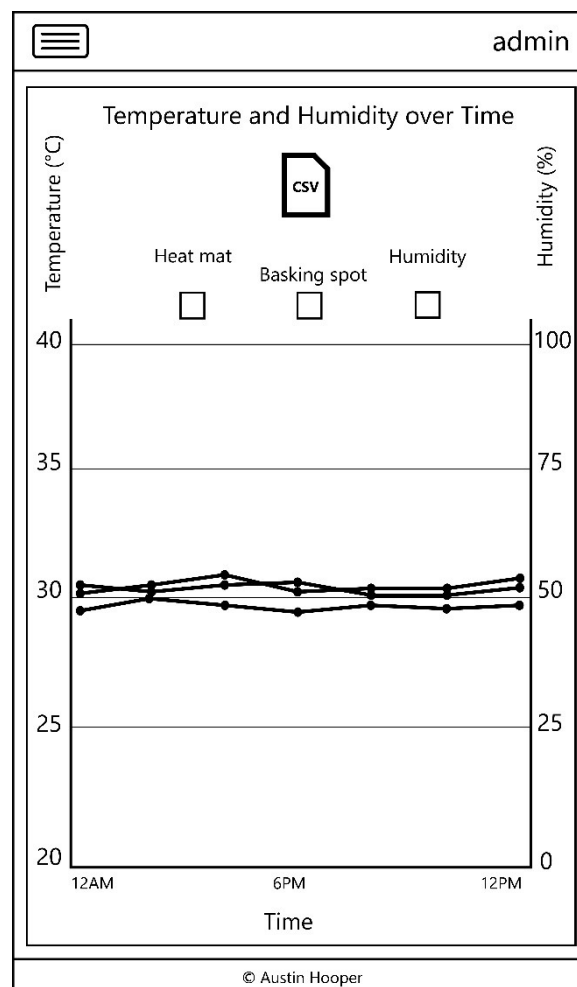


Figure C-12 – Mobile: log

Settings

[Home](#) [Dashboard](#) [Log](#) [Settings](#)

Reptile Settings

Bearded Dragon ▼

Name:

☒ Heat Bulb

30°C

☒ Heat Mat

31°C

☒ Ultraviolet
ON
OFF

Humidity

70%

Save

Add another reptile species

Terrarium Settings

Terrarium 1 ▼

Name:

Reptile Inhabitant: ▼

Save

Add another terrarium

© Austin Hooper

Figure C-13 – Desktop: settings

☰

Settings

Reptile Settings

Bearded Dragon ▼

Name:

☒ Heat Bulb

30°C

☒ Heat Mat

31°C

© Austin Hooper

Figure C-14 – Mobile: settings

Sign up requests

Home	Dashboard	Log	Settings
------	-----------	-----	----------

Sign up Requests

Username: username1	Email: user1@email.email	Accept	Deny
Username: username2	Email: user2@email.email	Accept	Deny

© Austin Hooper

Figure C-15 – Desktop: sign up requests

☰

Sign up Requests

Username: username1 Email: user1@email.email

Accept Deny

Username: username2 Email: user2@email.email

Accept Deny

© Austin Hooper

Figure C-16 – Mobile: sign up requests

Appendix D – ESP8266 Array Handling and Sensor Reading Subprocesses

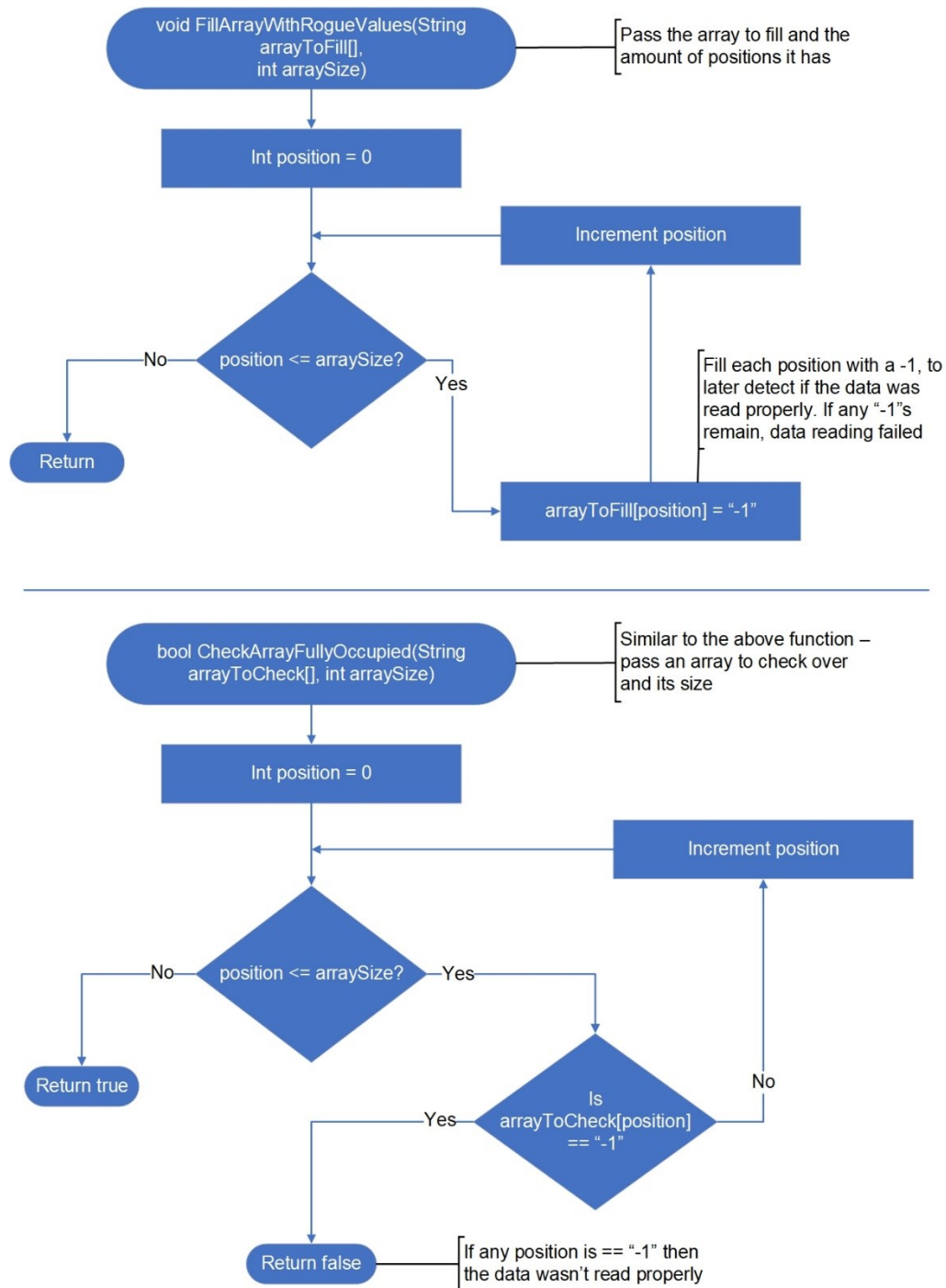


Figure D-1 – FillArrayWithRogueValues and CheckArrayFullyOccupied flowcharts

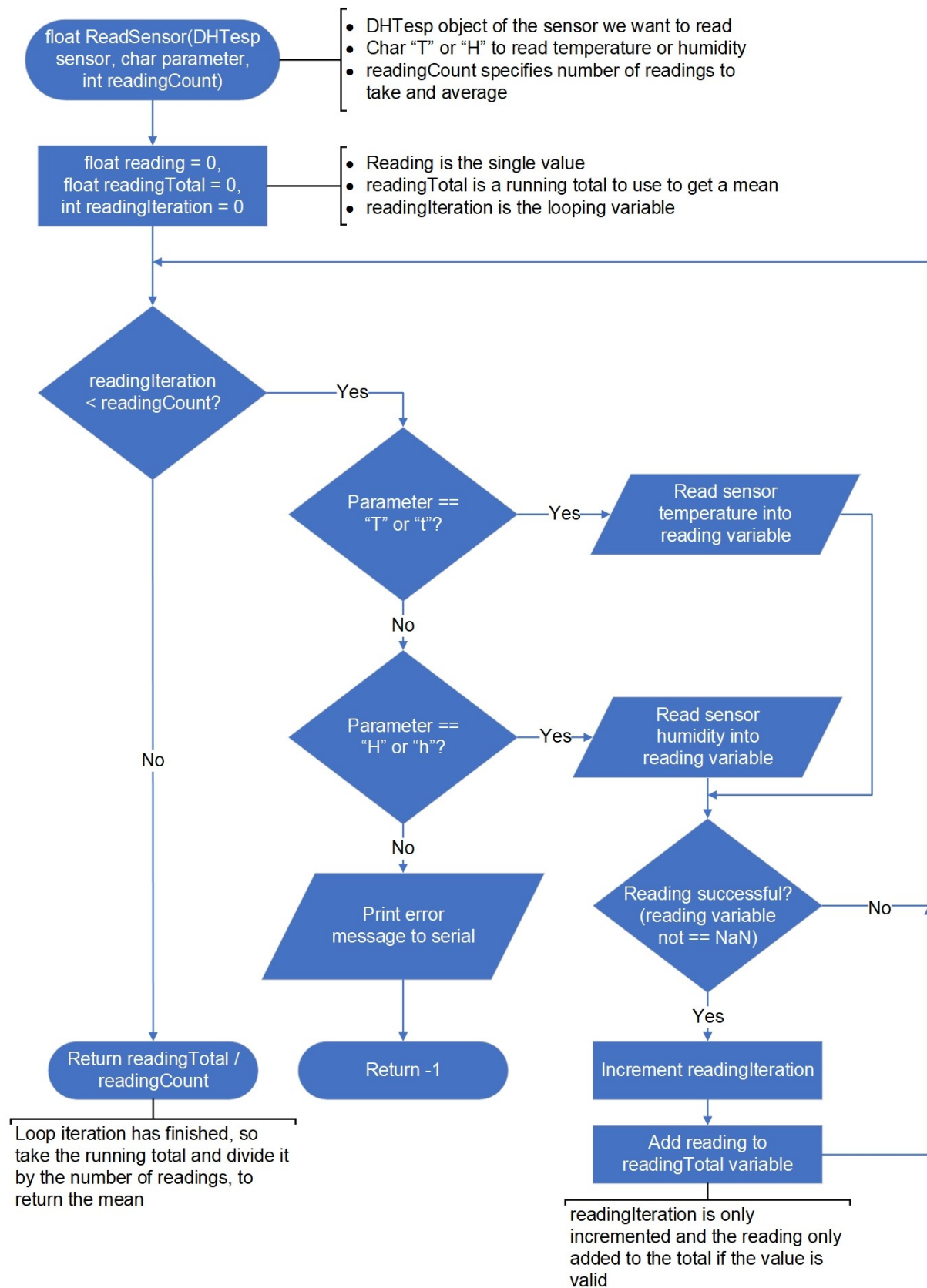


Figure D-2 – ReadSensor flowchart

Appendix E – SSL Self-Signed Certificate Process

Generating a self-signed certificate

In the Linux terminal on the Raspberry Pi, the below command was used to generate a certificate and key:

```
sudo openssl req -x509 -newkey rsa:4096 -sha256 -keyout  
/etc/ssl/private/apache-selfsigned.key -out  
/etc/ssl/certs/apache-selfsigned.crt -days 600 -config san.cnf
```

Here, openssl generates the certificate and key, each in the appropriate directory for the web server to use. They set the certificate to be active for 600 days, and link to `san.cnf` for further configuration. See below the contents of `san.cnf`:

```
[req]  
distinguished_name = req_distinguished_name  
x509_extensions = v3_req  
prompt = no  
[req_distinguished_name]  
C = UK  
ST = Wales  
L = Swansea  
O = UWTSD  
OU = Applied Computing  
CN = 192.168.1.1  
[v3_req]  
keyUsage = critical, digitalSignature, keyAgreement  
extendedKeyUsage = serverAuth  
subjectAltName = @alt_names  
[alt_names]  
IP.1 = 192.168.1.1
```

This configuration file was used to specify the certificate details such as location and address (as opposed to input via terminal), though more importantly to allow for adding Subject Alternative Names (SANs, hence the configuration file name). SANs

are additional domains which a certificate might apply to. While this certificate is only used for one, this is required by modern Chrome and Firefox to accept the certificate.

The below instructions were used for Chrome 79 on Windows 10.

Saving the certificate

In order to add a self-signed certificate from a website, first click the not secure icon, to the left of the address bar. Then, click on certificate, as shown below in Figure E-1. This will show a pop-up window, as is shown in Figure E-2. From here, select the details tab, and “copy to file... .” All default values can be chosen, along with choosing a file to save the certificate to. This will save the certificate file locally.

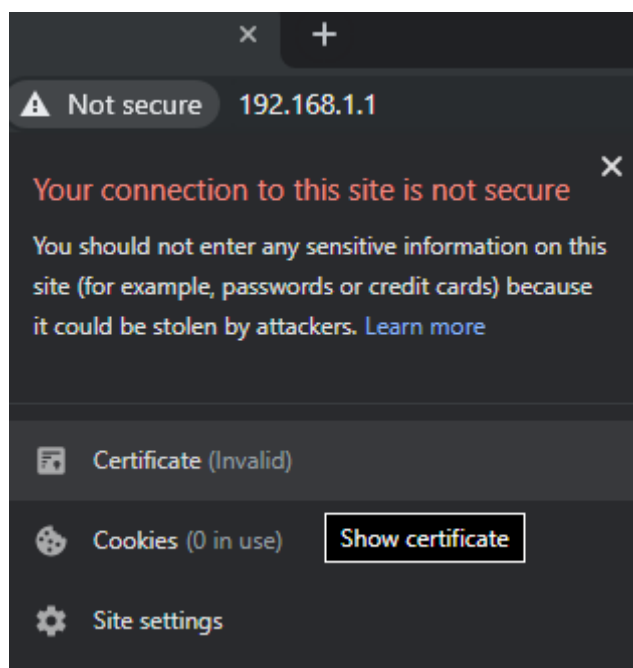


Figure E-1 – Dialog box for certificate information

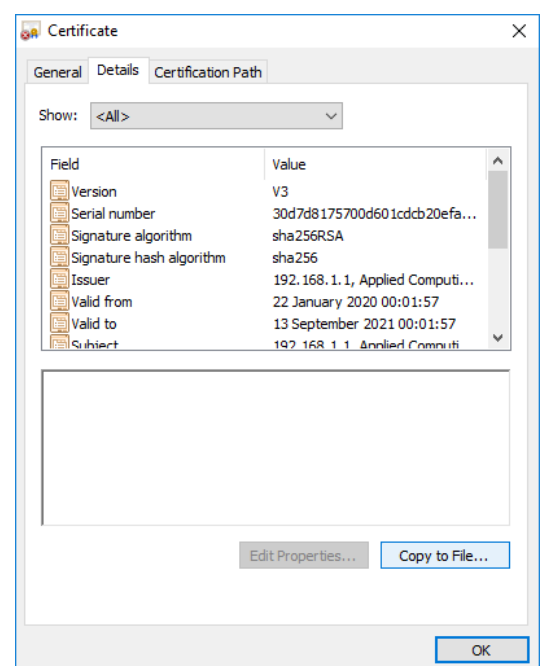


Figure E-2 – Saving the certificate

Installing the certificate

Installing the certificate directly to Windows' certificate store works for Edge and Internet Explorer, but not for Chrome or Firefox. For these browsers, the certificates must be installed to their own dedicated certificate store.

First, in Chrome's settings menu, search for SSL and click "Manage certificates" (Figure E-3). From here, in the pop-up window, Select the "Trusted Root Certificate Authorities" tab, and click the import button (Figure E-4). Follow the wizard guide, selecting the certificate saved earlier. When prompted to choose a certificate store, choose "Place all certificates in the following store." Click browse, and choose "Trusted Root Certificate Authorities" again (Figure E-5). Continue with the wizard until finished. If a pop-up displays discussing a security warning (Figure E-6), click yes.

The certificate is now installed, and will be effective after restarting Chrome.

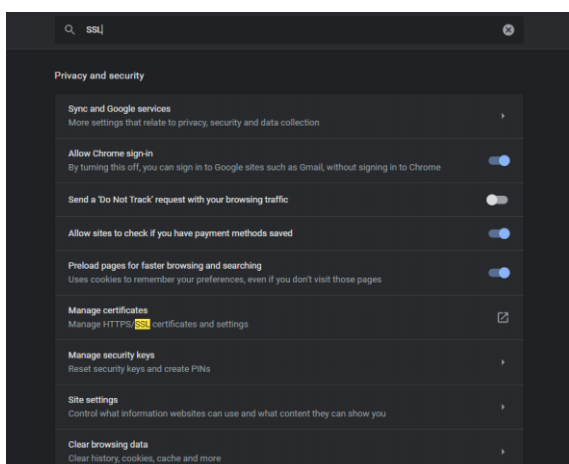


Figure E-3 – Chrome's manage certificates settings

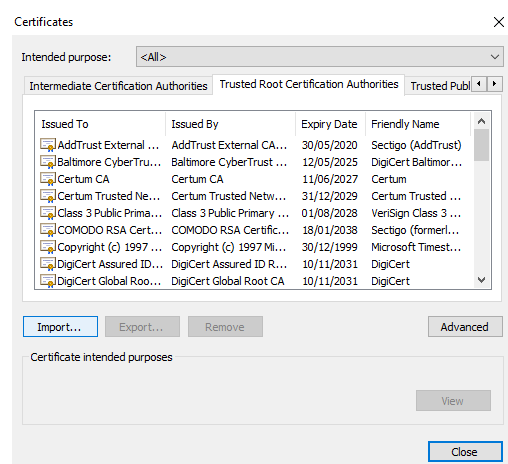


Figure E-4 – Certificates import window

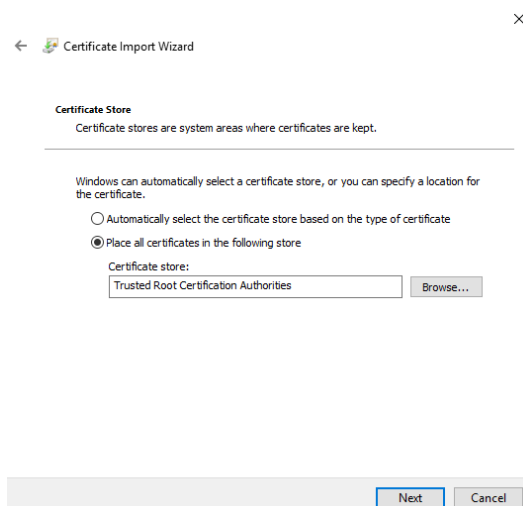


Figure E-5 – Certificate stores

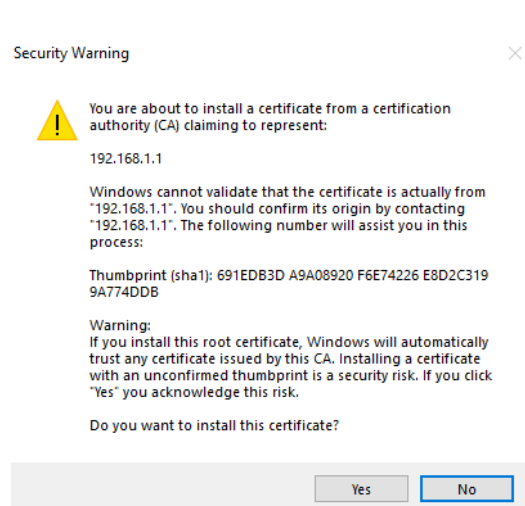


Figure E-6 – Security warning

Appendix F – Hardware Development Photos

A case was first prototyped in paper, before being 3D printed. However, later in the project, some power jacks were added and so a larger case was needed.

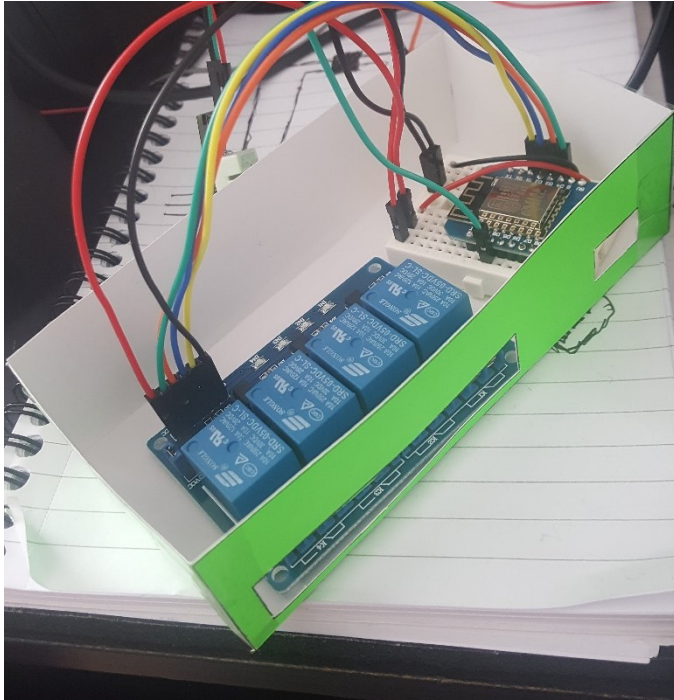


Figure F-1 – Paper prototype case

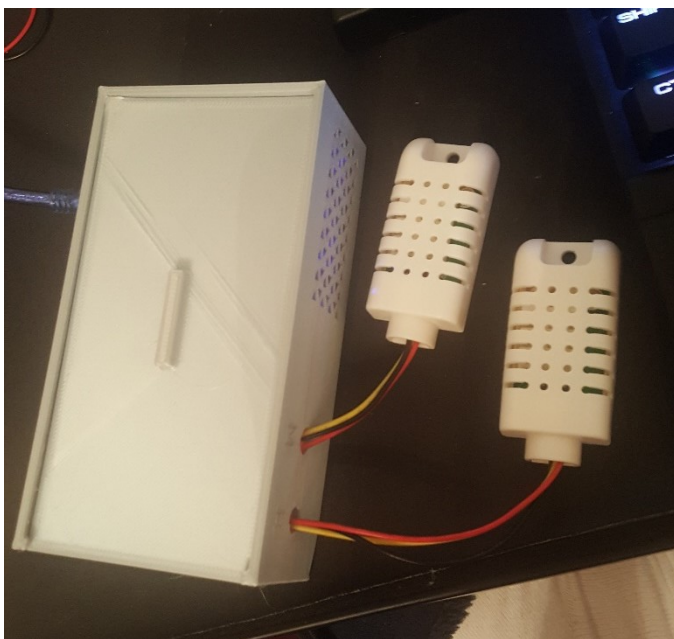


Figure F-2 – First 3D printed case

Figure F-3 shows the full system for a single node – a microcontroller board in the center, a relay board, two sensors, and real-time clock

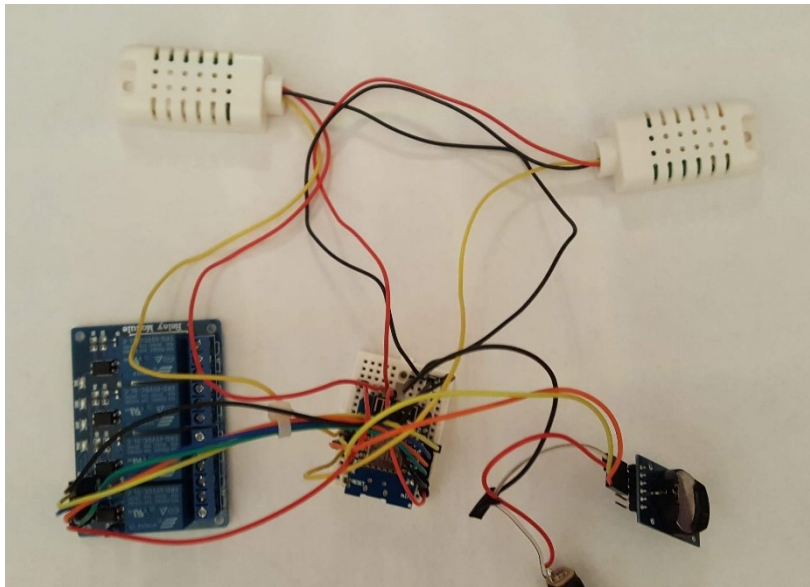


Figure F-3 – Full node system

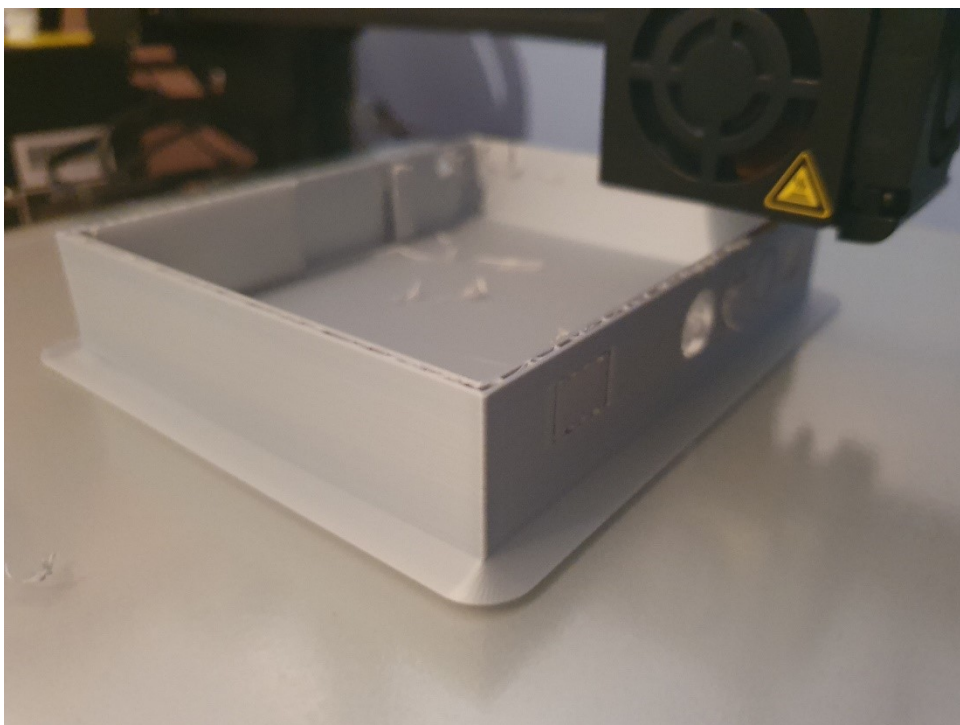


Figure F-4 – Larger case to accommodate relay plugs, mid-print

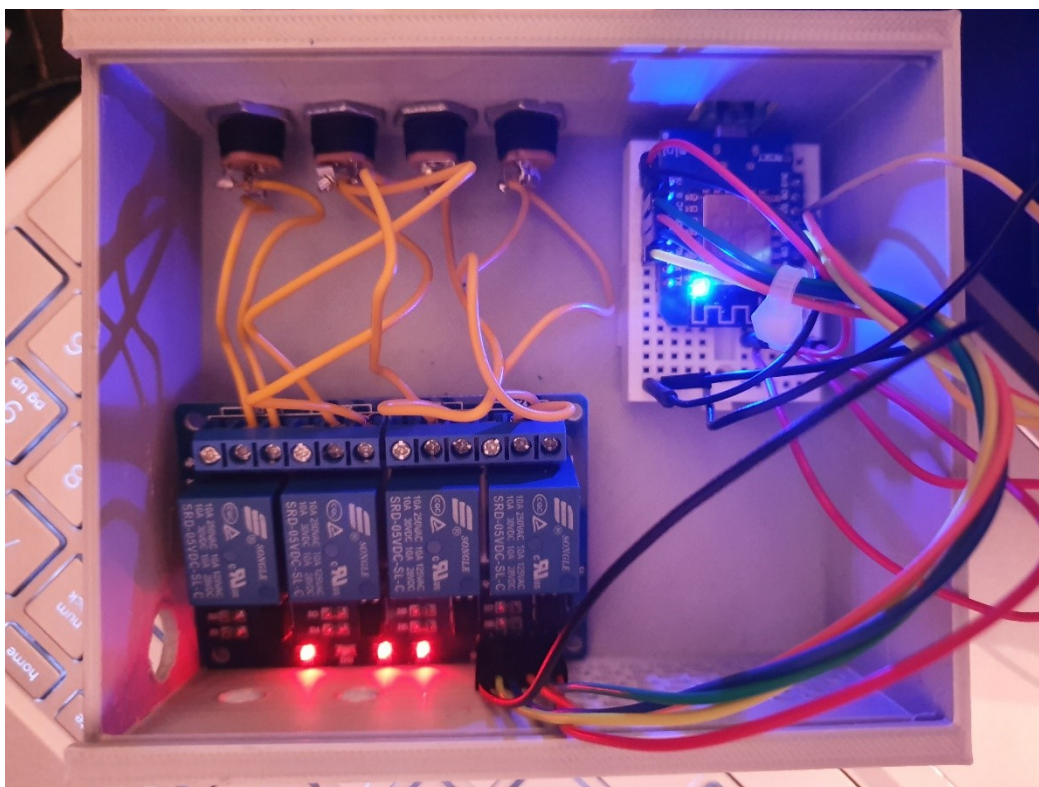


Figure F-5 – Larger case to with system inside, including replay plugs (top)



Figure F-6 – 3D printed pump tube adapter



Figure F-7 – LED light as a replacement for a heating lamp, due to time constraints

Appendix G – Sample User Experience Test and Questionnaire

IoT Web Application User Experience Test

This is a usability test of a web application for monitoring and controlling a reptile terrarium with an Internet of Things system. The system powers or un-powers heating elements, sprayers and UV lights based on current parameters and sensor readings, to control the environment of the reptile terrarium, in order to better suit the reptile inhabitant (pictured below).



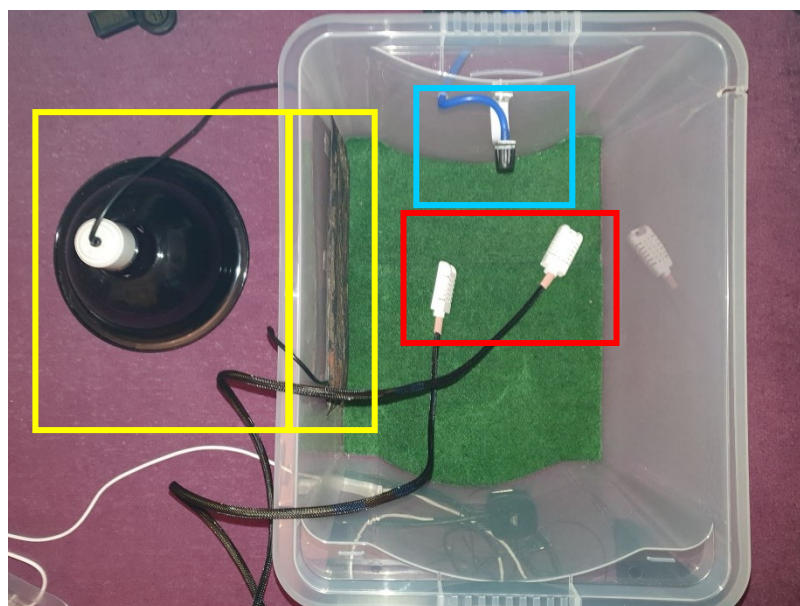
A reptile terrarium with heating lamps

<https://www.instructables.com/id/Reptile-tank-heating-and-lighting-guide/>



The happiest lizard

<https://animalchannel.co/happy-gecko-gets-new-toy/>



The prototype – two sensors (red), a sprayer (blue), and two heating elements (pad and lamp, yellow)

Background

These questions are anonymous and will be used to determine patterns in data.

1. What is your gender?

☒ Male ☐ Female ☐ Other ☐ Prefer not to say

2. Which age range do you fall within?

☐ Under 18 ☒ 18 - 24 ☐ 25 - 35 ☐ 36 – 55 ☐ 56 +

☐ Prefer not to say

3. Do you consider yourself to have any vision impairments, for example colour blindness? If so, please list:

N/A

☐ Prefer not to say

Please complete the below tasks, with as little assistance as possible. If you don't know the information to fill in fields, leave blank. If you believe it is not relevant or applicable, please enter "N/A."

Tasks

1. Sign up for an account. Use any details you like that fit within the requirements on the site.
2. This account has not yet been approved by an admin. For now, log in using these details:

Username: admin

Password: admin

3. Go to the **Dashboard** page. From the summary view, please fill in the below details for **UX Terrarium 1**:

Reptile species inhabitant:	<u>Leopard Gecko</u>	
	Temperature	State
Current heat mat temperature and state:	<u>22.4 degrees</u>	<u>On</u>
	Temperature	State
Current heat bulb temperature and state:	<u>37.4</u>	<u>On</u>
	Time range	State
Current ultraviolet time range and state :	<u>11:00 – 20:00</u>	<u>Off</u>
	Humidity	
Current humidity value:	<u>90%</u>	

Based on the readings' text format (where appropriate), what information can you infer about:

Is the heat mat temperature within its target? ☒ Yes ☐ No ☐ N/A

Is the heat bulb temperature within its target? ☐ Yes ☒ No ☐ N/A

Is the humidity within its target? ☐ Yes ☒ No ☐ N/A

4. From the summary dashboard page, select **UX Terrarium 2** to view more specific details about it. From information presented in gauges on this individual dashboard, please fill in the below details:

Use a range for the below answers.

e.g. 29.0 – 32.0 °C.

The target heat mat temperature: 33.5 – 35.5

The target heat bulb temperature: N/A

The target humidity: 70 - 80

Is the heat mat temperature within its target? ☐ Yes ☒ No ☐ N/A

Is the heat bulb temperature within its target? ☐ Yes ☐ No ☒ N/A

Is the humidity within its target? ☒ Yes ☐ No ☐ N/A

5. Go to the **Log** page. The default terrarium has no data, so will not display a graph.
 - a. Select **UX Terrarium 3** to view its details. As with previous tests, note the terrarium's target heat mat, heat bulb and humidity values.

The target heat mat temperature: 60

The target heat bulb temperature:

The target humidity: 20 - 30

- b. Change this to view readings for the last **3 days** instead

6. Change the theme from dark to light mode.

7. Go to the **Settings** page
 - a. Add a new **Reptile**.
 - i. Give it a unique name – this can be anything
 - ii. Set the heat bulb target to 30°C
 - iii. Set the heat mat to not needed
 - iv. Set the humidity target to 50%
 - v. Set the Ultraviolet time range to 08:00 **AM** to 09:00 **PM**
 - vi. Save this record
 - b. Edit this newly created reptile record: change any value and save changes
 - c. Add a new **Terrarium**
 - i. Give it a unique name – again this can be anything
 - ii. Set its reptile inhabitant to the reptile you created earlier
 - d. Edit this newly created terrarium record: change the name and save changes
8. Approve the sign up request you made at the beginning of this activity
 - a. Go to the **Sign Up Requests** page
 - b. Approve the sign up request you made at the beginning of this activity
9. Log out of the account

Please complete the below questionnaire from the System Usability Scale, based on your experience during this testing:

		Strongly disagree	Disagree	Neutral	Agree	Strongly agree
1.	I think that I would like to use this system frequently.	☐	☐	☐	☒	☐
2.	I found the system unnecessarily complex.	☐	☒	☐	☐	☐
3.	I thought the system was easy to use.	☐	☐	☐	☒	☐
4.	I think that I would need the support of a technical person to be able to use this system.	☒	☐	☐	☐	☐
5.	I found the various functions in this system were well integrated.	☐	☐	☐	☒	☐
6.	I thought there was too much inconsistency in this system.	☒	☐	☐	☐	☐
7.	I would imagine that most people would learn to use this system very quickly.	☐	☐	☐	☐	☒
8.	I found the system very cumbersome to use.	☒	☐	☐	☐	☐
9.	I felt very confident using the system.	☐	☐	☐	☒	☐
10.	I needed to learn a lot of things before I could get going with this system.	☒	☐	☐	☐	☐

Finally, any observations you have below:

Saw smiling gecko, pleased.

The graph overlay was a little difficult to tell apart.

Liked the dark mode.

Comments

Thank you for your time.

Appendix H – Usability Feedback Changes

Based on the user experience testing, the below changes were made:

Issue	Change	
Visited links were difficult to see due to their colour	Link colour changed to a brighter shade, contrasting with the dark background in dark mode.	
Sensor reading colours confusing	Colours changed from red, blue, white to red, blue, green based on relation to target (Figure H-1).	
Dashboard crowded, difficult to discern between each terrarium	Border added for each terrarium row to separate them from each other (Figure H-1).	
Graph on log page crowded and difficult to read	This issue needs further work, which could include loading only tables for mobile and graphs on desktop, or loading a different graph view on mobile.	
Site needs more labelling and contextual information	Labelling added on settings page, with additional context information added to select elements and inputs across the site.	
Accordion on settings page confusing and difficult to find	Accordion has been changed to stack horizontally, making it easier to notice and use (Figure H-2).	
HTML time input confusing and difficult to use	No change. Frameworks such as jQuery UI would be useful for custom time inputs in future work.	
Greyed out gauges and content were confusing – participants did not know what being greyed out meant	Additional text was added as overlays for gauges and actuators which are not needed (Figure H-3)	
Log out link difficult to find	Link added to navigation on every page.	

Before

Terrarium Summaries						
Testing Terrarium (Bearded Dragon)	Heat mat 30.1°C	Heat bulb N/A	Humidity 62%	 Heat mat ON	 Heat bulb N/A	 UV bulb ON (00:00 - 20:22)
UX Terrarium 1 (Leopard Gecko)	Heat mat 22.4°C	Heat bulb 37.4°C	Humidity 50%	 Heat mat ON	 Heat bulb ON	 UV bulb OFF (11:00 - 20:00)
UX Terrarium 2 (Bearded Dragon)	Heat mat 29.0°C	Heat bulb N/A	Humidity 72%	 Heat mat ON	 Heat bulb N/A	 UV bulb OFF (00:00 - 20:22)
UX Terrarium 3 (Yemen Chameleon)	Heat mat N/A	Heat bulb 39.5°C	Humidity 72%	 Heat mat N/A	 Heat bulb OFF	 UV bulb N/A

After

Terrarium Summaries						
Testing Terrarium (Bearded Dragon)	Heat mat 30.1°C	Heat bulb N/A	Humidity 62%	 Heat mat ON	 Heat bulb N/A	 UV bulb ON (00:00 - 20:22)
UX Terrarium 1 (Leopard Gecko)	Heat mat 22.4°C	Heat bulb 37.4°C	Humidity 50%	 Heat mat ON	 Heat bulb ON	 UV bulb OFF (11:00 - 20:00)
UX Terrarium 2 (Bearded Dragon)	Heat mat 29.0°C	Heat bulb N/A	Humidity 72%	 Heat mat ON	 Heat bulb N/A	 UV bulb OFF (00:00 - 20:22)
UX Terrarium 3 (Yemen Chameleon)	Heat mat N/A	Heat bulb 39.5°C	Humidity 72%	 Heat mat N/A	 Heat bulb OFF	 UV bulb N/A

Figure H-1 – Dashboard changes

Before

Settings

Add Reptile

Species name:

☒ Heat bulb

Basking temperature: Target: 30°C

☒ Heat mat

Mat temperature: Target: 30°C

☐ Ultraviolet

Humidity: Target: 50%

UV on: UV off:

Save reptile Edit existing Reptiles

Reptile Settings Terrarium Settings

After

Settings

Reptile Settings

Add Reptile

Species name:

☒ Heat bulb

Basking temperature: Target: 30°C

☒ Heat mat

Mat temperature: Target: 30°C

☐ Ultraviolet

Humidity: Target: 50%

UV on: UV off:

Terrarium Settings

Figure H-2 – Accordion changes

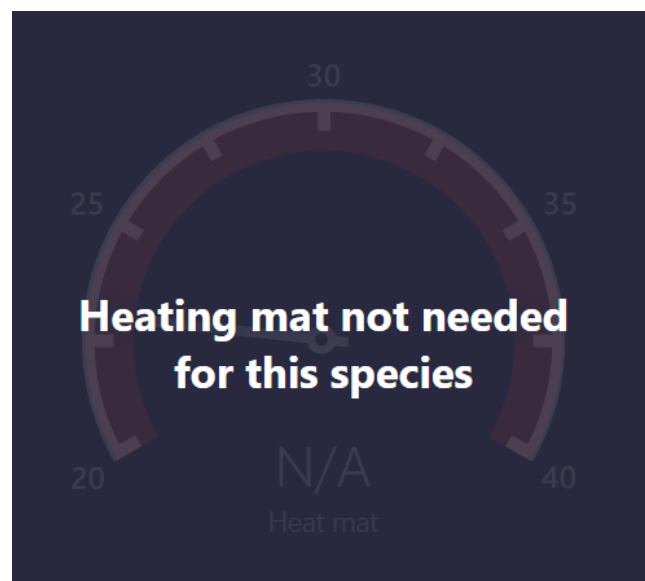


Figure H-3 – Gauge labelling added when not gauge needed