UNIVERSITY OF PORTSMOUTH

Networked Climate Monitor

By

Daniel Hearn - UP801685

Project unit: (PJE40)
Supervisor: Rinat Khusainov

May 2020

Abstract

As the Internet of Things (IoT) has grown, the number of commercial products using IoT has also grown. This includes the personal weather stations which have started embracing IoT through web-based interfaces, mobile applications, and cloud-based storage. While many commercial weather stations exist, these products suffer from a variety of limitations that affect their suitability in many situations. These limitations include the high costs, limited modularity affecting the number of climate areas that can be tracked at once, and the lack of modularity in the types of climate data that can be tracked. Due to these limitations, a solution was designed that provides node modularity and sensor modularity along with similar features to the commercial products such as web-based interfaces, historical data visualisation, and climate trends. The solution also aims to be more affordable with a low initial cost that can be later expanded with additional nodes. From this, a personal weather station was produced to fulfil the requirements of such a solution. This system provides modular nodes that measure the temperature, humidity, and atmospheric pressure of their surrounding environment, a base station to store and provide access to a website that acts as a viewing and management platform for the climate data and sensors. The nodes are battery-powered, waterproof, and use low-power radio communication. The solution was implemented with two individual sensor nodes and a base station and was then tested to evaluate its effectiveness against the problem and the limitations of the existing solutions. The evaluation provided mostly successful results in achieving its purpose with some issues with affordability, node communication range, and future expandability. Evaluation of the future work for this project and in the wider research area has been provided.

Word Count: 22536

Tables of Contents

List of Figures

1. Introduction

The number of active IoT (Internet-of-things) devices is expected to grow considerably with McKinsey predicting 43 billion IoT devices worldwide by 2023 (McKinsey, 2019). Weather stations are one type of system adopting IoT technology, with newer products including Netatmo (Netatmo, 2012) and BloomSky (BloomSky, 2016) using IoT to provide additional functionality and connectivity. The (World Meteorological Organization, 2018) defines these automated weather observing systems to be an integrated system of measuring instruments, transmission units, data processing functionality, and interfaces that collect weather measurements. Older weather stations such as Davis Vantage Vue (Davis Instruments, 2009) were limited to a single weather station with a base station with an LCD screen displaying the weather conditions recorded by the weather station. While more recent weather stations come with functionality including built-in internet connectivity, websites, mobile applications, and connectable services.

Weather stations allow users to not only understand current and historical weather conditions but react to any changes in the weather conditions. Additionally, a consumer application is connecting weather stations to home automation systems through internet-based connectable services. The measurements are then used to control the actions of the home automation system, such as activating air conditioning when a room's temperature is too high. The connectable automation services such as IFTTT have made this home automation integration possible due, instead of installing complete home automation systems. Weather station community websites such as Citizen Weather Observer Program (CWOP), Met Office Weather Observations Website (WOW), and Weather Underground (WU) are available for users to share their measurements with others. CWOP sees considerable use with over 8000 active weather stations uploading their weather observations each day (CWOP, 2020)**.** Professional applications of weather stations include smart-farming where weather stations provide remote monitoring of weather conditions across large areas used by farmers (Business Insider, 2020).

While weather stations have started to take advantage of IoT technology they still suffer from many of the limitations of older devices. These limitations have been identified as the problems this paper will attempt to solve. One of these is the lack of modularity in the number of locations being concurrently measured and the types of climate data they measure. Another limitation is that these systems are often expensive, especially if being implemented into a wider home automation system where additional hardware and smart devices are required to fulfil the complete automation functionality. Many of the problems encountered in the use of these weather stations are similar to those of home automation, in which household penetration is expected to grow from 9.3% at the start of 2020 and is predicted to hit 19.3% in 2024 (Statista, 2020). Regardless of the difficulties identified by (Brush et al, 2016) that limit the user adoption of home automation, including inflexibility, high costs, poor manageability, and security. (Gomez & Paradells, 2010) similarly identified barriers in the wider adoption of smart home systems, with high prices and technological fragmentation being among the top barriers. These barriers must be considered during the development of new smart home and home automation systems to ensure adoption by users.

Based upon the existing product limitations and user adoption barriers, the system aims to solve the problem that it is difficult to get accurate and real-time information about indoor and outdoor climates in areas that you live in or own such as your house or office. While the system will provide greater modularity, affordability, and flexibility than existing products and deliver the same core functionality available in newer weather stations including websites and connectable services.

The paper is organised into 11 sections. Section 2 evaluates the existing research within the system's background and in related systems. Section 3 defines the methodologies and project management techniques used in each stage. Section 4 describes the elicitation, evaluation, and specification of the requirements. Section 5 outlines the methods and outcomes used in the system’s design. Section 6 reflects on the system’s implementation. Section 7 describes the testing methods used and the results. Section 8 evaluates the results of the tests and questionnaire against the requirements specification. Section 9 concludes work on the project and the future work that can be completed with the system and in the project area.

2. Literature Review

2.1 Background

2.1.1 IoT & Smart Homes

The Internet Research Task Force (IETF) describes IoT to be the interconnection between entities and networks that can perform thing-to-thing, thing-to-things, human-to-human, or human-to-thing communication patterns (Internet Engineering Task Force, 2019). This is important in designing how the different components of the system interact based on their communications patterns. The IETF further defines these things as computing devices that track and react to their environment. Additionally, the IETF identifies the following constraints of IoT devices: limited computational power, memory, power consumption, and wireless network bandwidth (Internet Engineering Task Force, 2014a). These constraints must be considered when designing the system to pick the correct hardware and functionality. (Gomez & Paradells, 2010) identified many IoT communication technologies that have become available including ZigBee, Z-Wave, INSTEON, Wavenis, 6LowPAN. These also should be considered when designing communication between the systems components as they are better suited to IoT applications. (De Silva, Morikawa, & Petra, 2012) defines smart homes to possess automatic control by tracking the behaviour of residents and responding by providing facilities, this is achieved by using sensors to gather data to identify actions and events. The smart home context must be understood as the potential main use case of the system, as it should be able to interact with other smart devices.

2.1.2 Weather Stations

(Bell, Cornford, and Bastin, 2013) analysed the use of the WU and WOW services by personal weather station systems. Though this reflects use of the services in 2013, it identifies many statistics on weather station use by consumers. Including the product usage share, the costs ranging between £100 to over £1000, the measurement intervals used including 15, 10, 5, minutes and 1 minute, and the types of system structure these products use with a majority including an outdoor sensor with an indoor console. This is useful in understanding what user's will want from a weather station but does not provide information on use of more modern IoT weather stations such as BloomSky that were released after the paper was released. This shows a clear gap for future research on the adoption of IoT technologies by weather station systems and how this affects consumer user of the systems. (Bell, Cornford, & Bastin, 2015) evaluated the flaws of the measurements provided by users of WU, WOW, and the CWOP. They identified a warmer temperature bias that was linked to other variables in the design and placement of the weather station. They provided a method of correcting the biases by evaluating these variable relationships, due to this they advise that any application of the data used from these services use quality control to remove errors and correct biases. The flaws identified included issues with sensor calibration, weather station design, communication, and software errors. As these flaws can cause measurement bias, they need to be accounted for in the system design. Similar usage statistics to (Bell et al. 2013) were described, but additionally identified that the majority of users put their stations in their garden, this is must be considered as the main consumer location during the requirements and design stage as it requires weatherproofing the weather station. (Williams, Cornford, Bastin, Jones, & Parker, 2011) also analysed the biases on the temperature data provided by users of the Weather Underground service when compared to the temperatures available from nearby Met Office weather stations, this demonstrated a significant warm bias similar to (Bell et al, 2015). Based on the few pieces of literature identified, there is limited research on the limitations or user problems with weather station devices apart from those focused-on measurement accuracies. User surveys should be produced from existing weather station users to understand any missing features, feature priorities, and usability issues.

2.2 Related Work

2.2.1 Weather Stations

Many papers have designed and implemented various weather station systems by combining existing hardware and software technologies. (Kusriyanto & Putra, 2019) implemented a weather station using an Arduino with an ESP8266 Wi-Fi module to provide an internet connection, and multiple sensor modules. The measured data is stored locally, displayed on an LCD screen, and available on an IoT dashboard website by sending the measurements over an internet connection. Tests were completed on the accuracy of the measured data compared to a trusted source. While (Saini, Thakur, Ahuja, Sabharwal, & Kumar, 2016) implemented a weather station using an Arduino Uno and the ZigBee protocol to wirelessly connect the Arduino to a computer. The measured data is displayed on a website hosted on the computer. The interface used is outdated, though it uses purpose specific display methods for the different types of measured data. While testing was completed on the accuracy of the data compared to a trusted source, no tests were completed on the maximum range of the weather station though use of Zigbee should provide better range and lower power consumption than (Kusriyanto & Putra, 2019). Similarly (Adityawarman & Matondang, 2018) implemented an automated weather observing system using an ATmega32u4 microcontroller, BME280 sensor, battery, and Long Range (LoRa) radio transceiver as a sensor node. This node sends the measured climate data to a gateway computer with a LoRa module attached and onto a server, a website then displays the measured data in graphs. No tests were completed on the maximum range of the weather station even though LoRa is designed for long-range communication. (Savic & Radonjic, 2016) created a weather station using a Raspberry Pi to provide measurements and store them locally. This data was then available for viewing through a bespoke Android app accessible via the local Wi-Fi network, though it had no access over the internet. Additionally, the application is limited to displaying the most recent measured values with no historical data or graphs, this made it the most basic system researched.

While the systems of the previous papers focus on a single node and receiver, (Kapoor & Barbhuiya, 2019) proposed using multiple Raspberry Pi Zero W as sensor nodes, with a Raspberry Pi 3 as a base station to support the multiple sensor nodes. Amazon Web Services (AWS) was in the storage and processing of the data. A machine-learning model was trained and used to send notifications of significant or alarming predicted changes in weather conditions to the user. This was the only paper to use condition predictions to provide greater functionality, this should be considered as an additional feature for the system that could be implemented through future work, but is outside the scope of this paper, though the system’s design should be flexible for these additional features.

While other literature did not have focused use cases, (Tenzin, Siyang, Pobkrut, & Kerdcharoen, 2017) considered the smart farming use case to implement a weather station. The system consists of a solar-powered weather station recording measurements, this data is then sent using an Xbee module to a Raspberry Pi which is then forwarded to a cloud storage service. The stored data was regularly exported from a database to third party programs for analysis purposes. Tests were performed to compare the measured data to the data collected by a commercial product demonstrating very similar results, though it lacks an interface for viewing the measurements it can be integrated or expanded to provide viewing functionality. (Kanagaraj, Kamarudin, Zakaria, Gunasagaran, & Shakaff, 2015) took a different approach by proposing a system with multiple weather stations positioned across the state of Perlis in Malaysia. This used Wireless Sensor Networks (WSN) and cloud services to provide a website displaying the current weather conditions for each weather station imposed over a map. The WSN structure provided mesh networks for the weather stations to use to connect to local embedded computer systems that uploaded the measured data to the cloud services. Use of WSN should be considered within the system due to the range benefits it provides. This had the most types of data measured and was the only paper to include air and ground measurements.

Based on the trends of the related work, weather stations are being designed either with Arduino compatible microcontrollers or Raspberry Pi, demonstrating the use of low-cost hardware which will be required in this system to make the final system affordable. Most papers aren’t focusing Additionally, traditional networking protocols and IoT specific protocols are being explored, though there has been limited research and testing on evaluating the use of LoRa to provide weather stations covering very large geographic areas, thus future work needs to be completed.

2.2.2 Home Automation

Additionally, many home automation systems implement weather station sensor nodes to collect physical measurements and use them in part of a wider system. Based on this there is research that could be completed in combining existing home automation components with weather stations to evaluate their compatibility and suitability for smart homes. (Rahman, Hossen, & Rahama, 2017) also implemented a sensor node using a Raspberry Pi but used the PubNub cloud-based dashboard to display the measurements retrieved by the sensor node. The system uses a sensor management system to manage multiple sensors providing a use for the sensor nodes in a larger smart home system. (Vujović & Maksimović, 2015) also implements a home automation sensor node by using a Raspberry Pi, and again is designed for use in a larger system. The node provides an API service to provide measurement data retrieval via the internet, thus the user must have significant technological knowledge to integrate the node into a larger system.

3. Methodologies and Project management

3.1 Methodology

3.1.1 Overall Methodological Approach

The methodological approach was based upon the premise that fixing the limitations of existing weather station systems with a new system that has a modular focused design will increase user adoption of weather station systems and give a better user experience for existing weather station users. By using a modular approach the cost of the system can be reduced as multiple sensor nodes can be used an placed in different locations without purchasing multiple independent weather stations, additionally users only need to purchase the sensor modules they require to collect the measurements they want. This helps provides the affordability aim of the project’s problem. This methodology mainly influenced the research and requirements steps as these provided the existing limitations and modular improvements which all later steps considered in their approach. How this methodology affected each step and their tasks has been described individually.

3.1.2 Research and Requirements Methodology

The requirements step used background research on existing products, which gave an insight into the features, limitations, purposes, and priorities for the most popular weather stations on the market. This research analysis then provided an understanding of how modular focused design fixes the existing limitations, with these fixes being added as requirement specifications to ensure that the system requirements fulfil the project’s aim and successfully fix the project’s problem.

3.1.3 Design Methodology

The design step was completed by directly designing functionality that satisfied the functional and non-functional requirements while considering the wider issues within IoT and weather stations, and the usability of the UI. This meant that the design followed the wider methodological approach by including the modularity focused requirements that ensured that the improvements over existing systems completed by the system.

3.1.4 Implementation Methodology

The implementation step followed the designs to implement each component of the system. This approach meant that the modularity requirements continued to be considered across creation of the system so that the methodology and project’s aim were met.

3.1.5 Testing Methodology

The testing step was completed by testing the system directly against the requirements specification to ensure the project’s aims were met and that the modularity was implemented successfully. Automated tests were used to confirm correct functionality of each component giving consistent results when compared to manual testing methods. Though manual tests used for non-functional requirements that required more complex testing methods.

3.1.6 Evaluation Methodology

The methodology used in the evaluation step was to combine the test results with the results of a questionnaire given to potential users of the system and evaluate whether these results satisfy the requirements specification. This ensures that the methodology was continued across all steps of the system so that the modularity was achieved, and the product limitations fixed, and thus completing the project’s aim.

3.2 Software Development Life Cycle

3.2.1 Life Cycle Selection

The waterfall model was selected as the software development life cycle. This life cycle started with the research and requirement step, then the design, implementation, testing and evaluation steps. This meant that each step and task was approached individually due to the reliance of each step on the previous step. Overall, the life cycle was mostly successful with the research, requirements, and evaluation steps being completed in the planned order as reflected on in section 3.6 though deviations in the other steps were taken as described in section 3.7.

3.2.2 Step and Task Breakdown

The research step was broken down into research and analysis of existing weather station products and research papers. The requirements step consisted of elicitation, analysis, and documentation tasks that were completed in sequence. The design step was broken down into tasks for each system component, as most of the components rely on each other. The implementation step included tasks following the same component order as the design step. The testing step was broken down into tests for each component in a similar order to the implementation step. The evaluation step included the questionnaire and requirement evaluation tasks.

3.2.3 Life Cycle Alternatives

While the waterfall model was selected, there were other possibilities such as the agile methodology. These other life cycles were not selected due to their overall unsuitability for the project. Agile was not deemed to be suitable due to the number of interconnected devices, using this model would have resulted in each component having features developed at once, rather than implementing each component individually then connecting them.

3.3 Resources

During the project, several resources were used. A desktop computer was used to write the code, create designs, test the system, and write the report. The base station and sensor node required various electronics components including breadboards, wires, micro-USB cables, and resistors. The base station used a Raspberry Pi, Adafruit RFM69HCW Radio, and a weatherproof case. The sensor nodes BME280 sensor, Adafruit Feather 32u4 RFM69HCW, and a weatherproof case. During the testing, a USB voltmeter was used to provide the sensor node power consumption, while a laser measure was used to test the sensor nodes communication range. Open-source libraries have been used and described in the implementation in section 6.

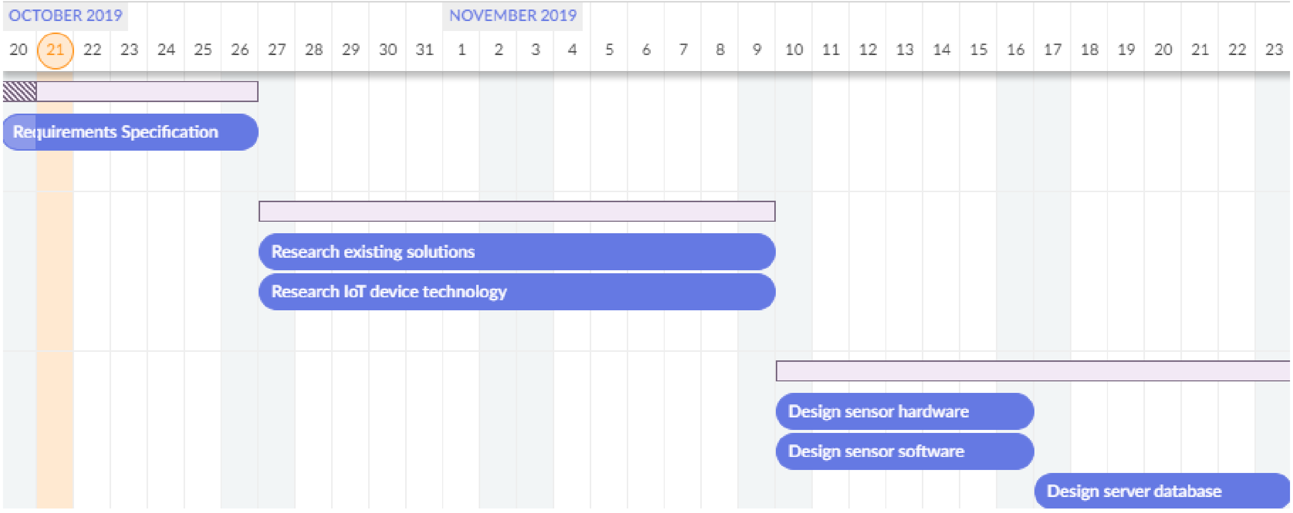
3.4 Project Management

A Gantt chart was used to track major tasks corresponding to the project steps, this is shown in figure 1. Each activity in this chart had a start and end date so that the current task could be easily understood, with the task end dates acting as deadlines. This allowed for tracking of which tasks were completed and if they were overdue, this was especially helpful for understanding the project’s progress and delays. This management method was successful for most of the project as it was regularly checked to understand the overall progress. But it suffered from a number of drawbacks, the first being that it was designed early on into the project’s creation so it did not take into account many later design choices that had impacts on the design, implementation and testing tasks. The second being that it did not always correctly reflect the progress on the project as it only included major tasks and not minor tasks that were mostly completed on time with only some of these being overdue.

Supervisor meetings were completed on most Wednesdays each week and provided opportunities to discuss any problems and plans for the next week's progress**.** This provided a vital way of ensuring accountability in the completion of deadlines by giving a day to have work completed by each week. Having the meetings on Wednesday allowed for development on Tuesdays, Wednesdays, and Fridays, with additional smaller pieces of work being completed on Mondays and Thursdays when other commitments required prioritisation. A Google Sheets spreadsheet was during the meetings to note the current week's progress and the next week's tasks so that precise progress could be understood.

A Trello board was used to track the individual tasks and subtasks during the implementation, testing, evaluation, and report writing. Lists were used to organise the area that tasks were being completed in. Within each list, cards were created for subtasks with each card being given a priority via a colour labelling system, and a due date. This allowed for understanding of the next tasks to be completed and when it should be completed by**.** Checklists were used to break down tasks even further and were especially useful during the implementation and testing tasks in which many small subtasks were being completed every day. This method was very successful as it provided multiple layers of prioritisation, deep organisation and structuring of tasks and subtasks, and was especially helpful for bug tracking. While Trello was used, Github Projects was considered due to its implementation alongside the project's source code, as pull requests could reference issues being tracked without plugins. But while it provided straightforward integration it lacked many of the features provided by Trello including the deadline, labelling and checklist features, thus demonstrating that Github Projects is only focused on bug tracking and limited feature tracking.

3.5 Project Plan



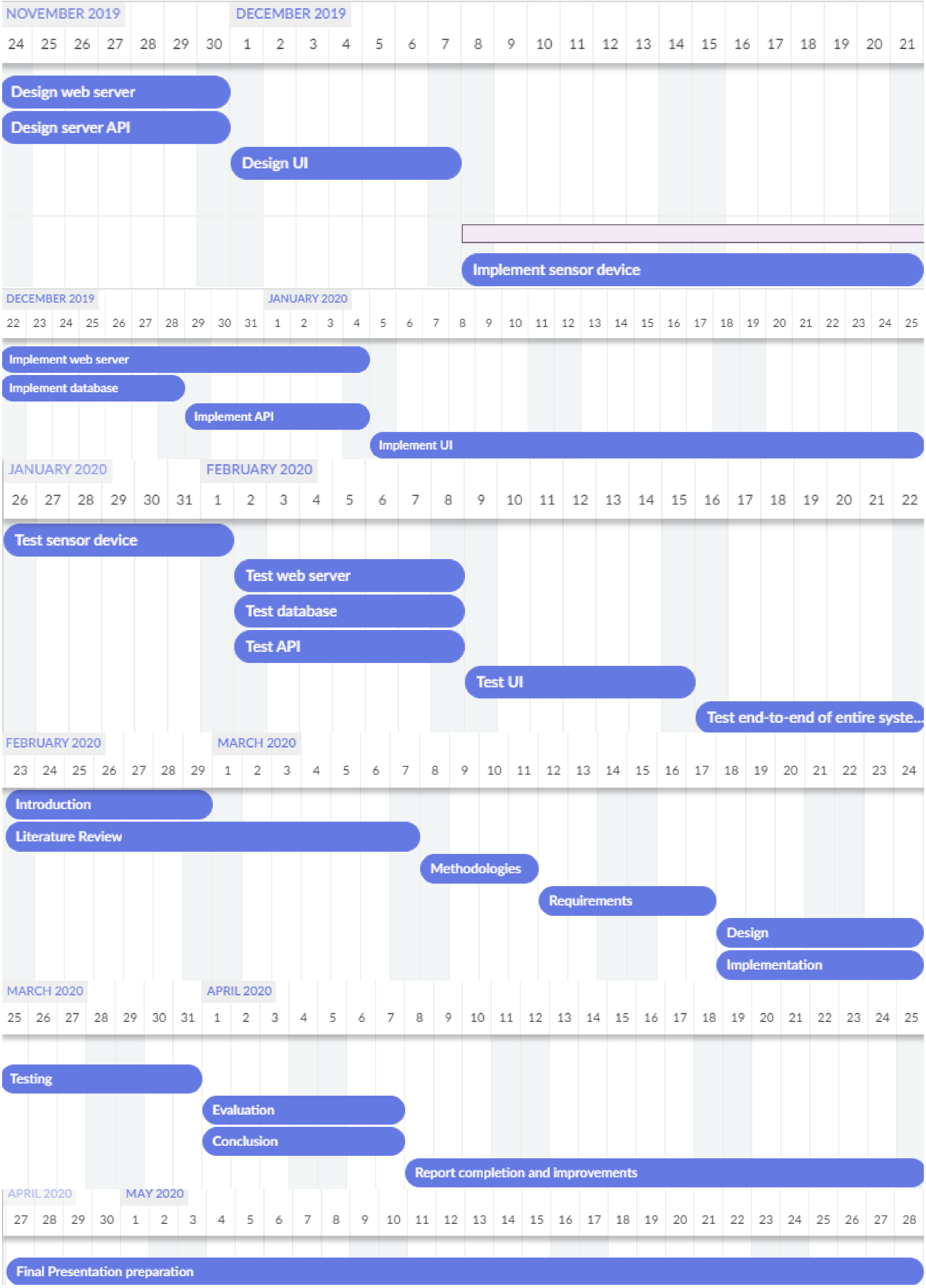


Figure 1. Project Plan

3.6 Reflection on Step Completion

3.6.1 Research and Requirements

The research and requirements steps were completed on time for each task's deadline according the project plan. This was mainly due to the indirect research and requirements elicitation methods used which required less time to complete than more direct methods such as interviews. Though some additional product research was completed during the design and implementation steps to understand the user interfaces provided by the products so that their advantages and disadvantages could be analysed. This was not accounted for in the project plan, though it could not have been completed before the design and implementation steps as it was reliant on the user interface requirements to be specified.

3.6.2 Design

The design step was mostly completed on time for the project plan's deadlines, though additional time was spent designing the user interface, which was not finished until the 18th of January, this deviation is described in section 3.7.2. This was much later than the previous December 8th deadline, which in reflection was not long enough to achieve a well-designed user interface that included the relevant requirements and usability. Instead at least two or three weeks could have been used to iterate over the user interface designs rather than the one week specified by the plan, as the three weeks were used to iterate while working on the implementation tasks.

3.6.3 Implementation

The implementation was completed on time for the deadlines even with the delays from the design step. Though the approach of creating combinable and reusable components for each part of the system meant that additional bug and test fixes were completed after the implementation step was formally completed, resulting in the deviation in section 3.7.2. In reflection this was unavoidable as spending time completing additional features and fixes at the end of the implementation step would have caused greater delays than completing them during the testing step.

3.6.4 Testing

The testing step was completed earlier than expected a week before the deadline, this was due to the planned week for testing each subsystem which was more than required for tasks such as testing the sensor node and implementing the end-to-end tests. This meant that the delays caused by the deviation in section 3.7.2 only affected the task deadlines within the testing step and not the overall step deadline.

3.6.5 Evaluation

The evaluation step was completed during the report writing as planned but was finished a week after its initial deadline due to the report delay deviation described in section 3.7.3. This step was completed over a few days as it relied on the questionnaire and requirement specification tasks which had already been completed, this helped ensure that the final report writing was finished before submission.

3.7 Deviations

3.7.1 Design and Implementation Overlap

Overlapping of tasks occurred during the website design and implementation**.** This was caused by the website design being delayed due to a focus on implementing the sensor nodes and base station over December and January along with changes in the database design. This resulted in the website designs being completed during implementation just before the website implementation was started.

3.7.2 Implementation and Testing Overlap

Additional time was spent improving the implementation during the testing due to additional unplanned features, bugs fixes and fixes for failing tests being completed. This resulted in delays for the website testing and in starting the evaluation step with these delays being exacerbated by the existing delays from the deviation in section 3.6.1.

3.7.3 Report Delays

The report had delays across all sections, this was due to other commitments and difficulty in starting the writing based on the notes that had already been established for a few months before the writing began. This resulted in the report being written later than originally planned with delays ranging from 2 weeks to a month depending on the section. The delays of this deviation had large impacts as the report sections were planned to be completed per week, though some were written in a day or two to catch up with the final deadline with the report being completed one week before it was due. The section delays were mainly due to completion of the larger literature review, design, and implementation sections as the structure of the notes for each these made it harder write formally when compared to writing a section without pre-established notes and ideas.

4. Requirements and Analysis

This section introduces a description of the problem being solved, the methods used to elicit and analyse these requirements, and the specification of the system's requirements. This section has been built upon the initial requirements as established within Appendix A the Project Initiation Document.

4.1 Elicitation

The requirements were indirectly elicited through background research of the existing weather station products. This method of retrieving requirements was completed as the problem is focused on producing a cheaper and more modular solution to the existing systems and not about delivering a new and unique system. This meant that direct requirements elicitation from users was not required but could have been used to better understand what secondary features, usability, or performance issues that users have found with existing systems. It would have also provided feedback for users to describe features that were not offered by existing systems.

The products found during this research have been displayed in table 1, along with their main feature and specifications. Additional features and specifications were researched and included in the full background research in the Appendix D. The features researched included: cost, whether they support multiple nodes, have modular nodes, types of sensor data collected, and the accuracy and ranges of the temperature and humidity values, maximum node range, website features available, connectable services, measurement frequencies and the number of nodes per account. These specifications were selected as they demonstrated the suitable purposes, and priorities of each product.

Name	Cost	Multiple Nodes	Modular Nodes	Temperature Accuracy	Temperature Range	Humidity Accuracy	Humidity Range
WeatherFlow	£234	Yes	No	±0.4°C	-38°C to 60°C	±4%.	0 to 100%
Ambient Weather Osprey Weather Station	£100	No	No	± 0.5°C	-40 to 65°C	± 5%	10 to 99%
AcuRite Atlas	£145	No	No	± 1°C	-40 to 70°C	± 2%	1% to 99%
Davis Vantage Vue	£350	No	No	± 1°C	-40° to 65°C	± 2%	1% to 100%
BloomSky	£295	No	No	±0.3°C	0°C to 65°C	± 3%	10% to 90%
BRESSER WiFi Professional Weather Station	£197	No	No	Not specified.	-40°C to 60°C	± 1%	1% to 99%
Netatmo	£149.99	Yes	No	± 0.3°C	-40°C to 65°C	± 3%	0 to 100%

Table 1. Existing products and their features

4.2 Analysis

4.2.1 Analysis Approach

Most of the requirements were analysed from the trends in the features and specifications for the existing projects researched in the elicitation. These requirements were focused on existing product features and improving upon their limitations. The rest of the requirements were then extracted through analysis of the problem specification, this provided much of the core functionality of the system. The relative importance of each requirement was then analysed from their importance in solving the problem and whether they matched the features offered by the existing systems as this indicates whether they are an important feature.

4.2.2 Project Feasibility

Due to the limited amount of time available to produce the system and affordability aim of the project some secondary and usability features have been given lower importance or left out so that the primary problem and aims of the project can be fully focused on. This approach may not result in a polished commercial product but a functional but rough prototype of a solution that solves the problem but has limited usability and secondary features. These features include using solar power for recharging the sensor nodes, providing weather forecasting on the website for a node’s location, and displaying a community sensor node map so that users can view other user’s sensor node data.

4.2.3 Approach Limitations

While this approach provides requirements for a system that is built directly from the analysis of the problem and a snapshot of the features and limitations of existing systems, it relies on assuming users want the features specified by existing products. But this approach is suitable when considering the project’s aim is to improve upon existing systems based on their limitations and not to produce a unique and new system that fundamentally reinvents personal weather stations. This approach could have been improved from direct user elicitation for the features that they want from a personal weather station and the importance of these features.

4.3 Results

4.3.1 Results Overview

The results of the analysis demonstrate that the products took different approaches to solve the problem, though there are clear similarities and trends in the specifications. This means the requirements specification must be focused on solving the limitations of existing products while keeping the features provided by the products. These product limitations mostly match those identified by the papers researched in the literature review.

4.3.2 Sensor Node Structure

The results of the analysis show that all the products offered outdoor sensor nodes with most providing indoor sensor nodes or similar functionality in the indoor console

4.3.3 Climate Data Measurements

All products offered temperature and humidity measurements, with some providing other data types including wind speed and direction, UV index, rainfall, light levels. Due to this the system should support temperature and humidity measurements, but also support any type of sensor hardware to collect other types of measurements.

4.3.4 Sensor Node Battery Power

All used batteries for their sensor nodes with some using solar power to recharge while others needed manually recharging or replacing their batteries**,** thus the system's nodes must be battery powered. The battery life ranged from two weeks to 2 years, these depended on whether they used solar power as those that did have shorter battery lives. Thus, the system will need a battery life of a few months to ensure usability while balancing affordability.

4.3.5 System Affordability

Most cost between £100 to £200, with two between £200 to £300, and one cost £345, based on this the cost of the system should be less than £100 to ensure that the system is more affordable than the existing systems.

4.3.6 System Modularity

Only some of the systems supported multiple sensor nodes with those having many limitations, thus the system will need to support multiple nodes in the system.

4.3.7 Measurement Storage

All systems had long term storage of the measurement data though they had different max lengths ranging from 1 month from the measurement to unlimited, this means that the system will need to support storing data for a long time-period, though a limit is required to ensure that the database is still performant and storage is exhausted.

4.3.8 Website

All products offered a bespoke website or built-in connectivity to a personal weather station website. These websites ranged in functionality, but all included recent measurement values, and all but one product displayed the measurements graphed over time, management of the sensor nodes and configuration options. All the products that had management and configuration options required user authorisation to access this functionality, though not authorisation was required to view the recent and historical climate data. Thus, the system will need to provide a website covering recent and historical climate data visualisation, sensor node management and configuration functionality. Additionally, all websites supported mobile, desktop, and laptop devices, thus the system will need to be compatible with modern browsers across these devices.

4.3.9 Measurement Accuracy and Ranges

All products had similar accuracies, ranging from 0.3°C to 1°C for temperature and 1% to 5% for humidity. Thus, the system should aim for the best possible measurement accuracies to be similar to these products while balancing costs and hardware support. Similarly, all products had nearly identical measurement ranges with the temperature ranges being between -40°C to 70°C and humidity ranges from 1% to 100%, thus the system will need to support as many locations and purposes as possible by supporting large measurement ranges for all types of climate data.

4.3.10 System Connectivity

All but two of the products offered connectivity to internet-based services such as Amazon's Alexa and IFTTT, due to the aim of this system being part of wider smart home systems it will need to be able to connect to these services. None of the systems required technical knowledge to use the system, thus the system should be suitable for anyone to use, install, and maintain the system.

4.4 Functional Requirements Specification

The requirements listed below describe the functions that the system will need to perform so that the system successfully solves the problem.

4.4.1 Indoor Sensor Node
The system shall have a node that uses sensors to measure the surrounding climate and is designed to be used indoors.
Rationale: The problem specifies that the system will need to measure climates and be modular so flexibility on where the sensor nodes can be placed will be required.
Importance: High importance as modularity is a main aim and a main limitation of the existing systems, with only some systems offering indoor climate measuring.

4.4.2 Outdoor Sensor
The system shall have a node that uses sensors to measure the surrounding climate and is designed to be used outdoors.
Rationale: The problem specifies that the system will need to measure climates and be modular so having flexibility on where the climate sensor nodes can be placed will be required.
Importance: High importance as all existing systems have outdoor sensors.

4.4.3 Sensor Node Temperature Recording
The sensor nodes shall record the temperature from the surrounding climate.
Rationale: The problem specifies that temperature is a key part of the climate data and all existing systems record temperature so it will need to be recorded.
Importance: High importance as all existing systems record temperature measurements.

4.4.4 Sensor Node Temperature Units
The sensor nodes shall record the temperature in Celsius while being able to view this temperature in other units of measurement.
Rationale: The sensors should all record the same unit of measurement, with this unit then being available for conversion at a later stage (i.e. Celsius converted to Fahrenheit).
Importance: High importance as the data is not useful if the unit of measurement is not specified or consistent, and all existing systems provide Celsius and Fahrenheit as temperature units of measurement.

4.4.5 Sensor Node Humidity Recording
The sensor nodes shall record the relative humidity from the surrounding climate.
Rationale: The problem specifies that humidity is a key part of the climate data and all existing systems record humidity so it will need to be recorded.
Importance: High importance as all existing systems record humidity.

4.4.6 Sensor Node Recorded Data
The sensor nodes shall record the climate data along with the time and date while making the climate data accessible for viewing on the website.
Rationale: The recorded climate data will need to be accessible via a website, so it will need to be stored in a location that the website can retrieve the data from. The time and date so that the changes in climate data can be tracked over time.
Importance: High importance as the problem specifies that a website will be required to view the data remotely over the internet, and all systems had a website that allowed for the remote viewing of the climate data.

4.4.7 Sensor Node Identifier

Each sensor node shall be identified by a unique identifier across the system.
Rationale: The climate data will need to be linked to the sensor that recorded the data once it has been stored within the system so that the data can be understood.
Importance: High importance as the otherwise users would not know which sensor recorded the data and tracking climate data in the system would be difficult.

4.4.8 Additional Sensor Types

The sensor nodes shall have the capability to connect to additional sensors that can record other types of climate data.
Rationale: A main aim of the project is modularity so having sensor nodes that can record multiple types of data from separate sensors while only using one node is required.
Importance: High importance as all existing systems record additional data other than just temperature and humidity, so having full sensor modularity will allow the system to be more flexible than existing systems, while also allowing them to be parallel with the types of climate data recorded by the existing systems.

4.4.9 Multiple Sensor Nodes

The system should support multiple sensor nodes that each record their own climate data.
Rationale: A main aim of the project is modularity so having multiple sensor nodes that can record data from their own sensors is required.
Importance: High importance as modularity is a key part of the problem and major limitation of the existing systems as only some of the systems can support multiple sensor nodes.

4.4.10 Sensor Battery Power

The sensor nodes should be battery-powered.
Rationale: The sensor nodes will need to be placed within ideal locations to get good climate measurements of the location, without restrictions on their placement.
Importance: High importance as all existing systems had sensor nodes powered by rechargeable or replaceable batteries.

4.4.11 Sensor Names

The sensor nodes should each have a user-defined name.
Rationale: The system will support multiple sensor nodes thus users will need to identify which sensor captured the data and where this sensor is located, thus a user-defined name allows users to describe the location of the sensor.
Importance: Medium importance as it does not affect whether the system works but has a large impact on the usability of the system.

4.4.12 Climate Data Accessible by External Services

The climate data should be accessible by connectable services that have permission from the user to access their climate data.
Rationale: This allows for the climate data to be used to achieve additional smart home functionality e.g. an external service gets a sensor’s temperature data and the service controls air conditioning in the room the sensor is located in.
Importance: Low importance as it does not directly impact the functionality of the system but allows for a greater use case within a smart home system, this feature is also available at most of the existing systems which can connect to various personal assistant applications and internet of things services.

4.4.13 Website

The system shall provide access to a website over the internet that acts as a dashboard for the sensor's and their climate data.

Rationale: The problem specifies that climate data should be accessible remotely, a website accessible via the internet would allow for easy access by users.

Importance: High importance as viewing the climate data remotely is a key part of the problem and all existing systems have a website displaying the climate data.

4.4.14 Website Authorised Data Access

The website shall only provide access to a user's climate data after the user's email and password has been provided and validated against the email and password stored for that user.

Rationale: The sensor and user data should be secure being an authorisation method so that only the owner of the account can access their account's data.

Importance: High importance as it keeps the project in parallel with the existing systems that all used user registration and login functionality.

4.4.15 Website Desktop/Laptop Access

The website shall be accessible with modern browsers on desktop and laptop devices.

Rationale: The website will need to be accessible by users that use desktop and laptop devices, thus the website needs to work correctly on modern browsers available on these devices. While supporting older browsers requires additional development time and is unnecessary for a prototype implementation of the system.

Importance: High importance as it is required for the system to achieve the project's aim of having a website to view the climate data on, and all existing systems have a website with a user interface specifically designed for desktop and laptop devices.

4.4.16 Website Mobile Access

The website shall be accessible on mobile devices with this website having a user interface optimised for mobile devices.

Rationale: The website will need to be accessible by users that use mobile and tablet devices, thus the website needs to work correctly on modern browsers available on these devices. While supporting older browsers requires additional development time and is unnecessary for a prototype implementation of the system.

Importance: High importance as it is required for the system to achieve the project's aim of having a website to view the climate data on, and all existing systems have a website or application with a user interface specifically designed for mobile devices.

4.4.17 Sensor Node List

The website shall provide access to a list of a user's sensor nodes, with each sensor being displayed with their name and a summary of their latest climate data.

Rationale: As the system will support multiple sensors for each user, the users should be able to view the sensor's they have and their recorded data.

Importance: High importance as it is required for the system to achieve the project's aim of having a website to view the climate data and keeps it in parallel with the existing systems that support multiple sensors.

4.4.18 Historical Climate Data Visualisation

The website shall provide graphical visualisation of a user's climate data over time based on user-specified date and time ranges.

Rationale: Viewing the historical data of a sensor allows users to understand the trends of climate data over time for analysis purposes.

Importance: High importance as it is required to achieve the project's aim of having a website to view the climate data and keep it in parallel with the existing systems as they all have graphical visualisations of the historical data.

4.4.19 Recent Climate Data
The website shall provide access to the most recent data recorded from each sensor node assigned to the user.
Rationale: The problem specifies viewing the current climate around each sensor.
Importance: High importance as it is required for the system to achieve the project's aim of having a website to view the climate data and keeps it in parallel with the existing systems as they all have dashboards displaying the recent data collected by each sensor.

4.4.20 Website Temperature Units
The website shall allow users to view recorded temperature data in Celsius or Fahrenheit.
Rationale: By storing the temperature data consistently in Celsius, these values can be converted to Fahrenheit so that users can view the data in the unit they prefer within the website without affecting the data stored within the system.
Importance: Medium importance as it doesn't directly affect whether the project meets its aim, but it is important for improving the usability of the system.

4.4.21 Climate Data Deletion
The website shall provide the ability to delete all climate data recorded by a sensor.
Rationale: Users may move a sensor's physical location and may want to reset the stored data as the previous climate data is not representative of the new location.
Importance: Medium importance as it doesn't directly affect whether the project's aim is achieved but it impacts the efficiency of testing the system and the usability of moving the location of sensor nodes.

4.4.22 Sensor Deletion
The website shall provide the ability to delete a sensor from a user's account.
Rationale: Users can have multiple sensors, so they may want to remove a sensor from the system, if they are no longer using a specific sensor.
Importance: Medium importance as it does not directly affect whether the project's aim is achieved but it impacts the usability of the system when multiple sensors are being used.

4.4.23 Website Sensor Naming
The website shall allow for the names of the sensor nodes to be edited.
Rationale: Users can have multiple sensors each with different names, so an interface will be required so that users can edit these names.
Importance: Medium importance as it doesn't directly affect whether the project's aim is achieved but it impacts the usability of the system when multiple sensors are being used and puts it in parallel with the existing systems that support multiple sensors per user.

4.4.24 Login
The website shall allow registered users to log in once they provide the correct email and password for their account.
Rationale: Users are required to register to access the website, thus a login page is required so that registered users can log into the website.
Importance: High importance as a user account system is required to separate users within the system, this also keeps the system in parallel with the existing systems.

4.4.25 Register

The website shall allow users to register by providing a valid email and password.

Rationale: The website will require users to register so a register page is required so that non-registered users can register.

Importance: High importance as a user account system is required to separate users within the system, this also keeps the system in parallel with the existing systems.

4.4.26 Forgot Password

The website shall allow users to reset their account password by receiving an email containing a link to a password reset form on their registered email account.

Rationale: The website will require users to register and login, so a password reset page is required so that registered users can reset their password if they have forgotten it.

Importance: High importance as a user account system is required to separate users within the system, this also keeps the system in parallel with the existing systems.

4.4.27 Add Sensor Node

The website shall allow users to add a new sensor node to their account.

Rationale: Users need to set up the sensors with the website so that it is linked to the user.

Importance: High importance as it is required for the project's aim to be achieved and impacts whether the climate data for multiple sensors can be viewed on the website.

4.4.28 Sensor Status

The website shall display the sensor's battery level.

Rationale: Users would need to know if a sensor's battery is running out of power so that they can charge the battery and have continuous sensor uptime.

Importance: Medium importance as users would likely already understand if a sensor's battery has run out as the time of the last recording would be older, and only some of the existing systems offered a battery status.

4.5 Non-functional Requirements Specification

The requirements listed below define the effectiveness of the system in achieving its functionality.

4.5.1 Recent Climate Data

Climate data on the website should be the most recent data within one minute of the data being recorded on the sensor node.

Rationale: The system should be quick to process data from the time it is sent from the sensor to be viewable on the website, as showing old data makes the system limited to users that want to see current climate data and make decisions upon it.

Importance: High importance as out of date data would not be useful to users and viewing the current climate data is a main aim of the project.

4.5.2 Temperature Climate Data Accuracy

Temperature climate data should be recorded within ±0.5°C accuracy.

Rationale: The recorded temperatures should be accurate in parallel with the accuracy of the existing systems while maintaining the affordability of the system and the availability of low-cost temperature sensors.

Importance: High importance as the system would be flawed and not meet the project's aim if it does not record temperatures that are useful enough to be used to analyse the climate.

4.5.3 Temperature Climate Data Range

The temperature sensor should be able to record within a range of -40°C to 80°C.
Rationale: The sensors should record temperatures in ranges that are in parallel with the existing systems temperature ranges while maintaining the affordability of the system and the availability of low-cost temperature sensors.
Importance: High importance as the system would be flawed and not meet the project's aim if it does not record temperatures in most environments and weather conditions.

4.5.4 Humidity Climate Data Accuracy

Humidity climate data should be recorded within ±5% accuracy.
Rationale: The recorded humidity should be accurate in parallel with the accuracy of the existing systems while maintaining the affordability of the system and the availability of low-cost humidity sensors.
Importance: High importance as the system would be flawed and not meet the project's aim if it does not record humidity values that are useful enough to be used to analyse the climate.

4.5.5 Humidity Climate Data Range

The humidity sensor should be able to record within a range of 0-100%.
Rationale: The sensors should record humidity in ranges that are in parallel with the existing systems humidity ranges while maintaining the affordability of the system and the availability of low-cost humidity sensors.
Importance: High importance as the system would be flawed and not meet the project's aim if it does not record humidity in most environments and weather conditions.

4.5.6 Concurrent Climate Sensor Nodes

The system should be able to handle between 1 to 10 climate sensor nodes within or around a single location or house.
Rationale: The system should be more modular than the existing systems while maintaining good performance within the network and in the system storing the climate data.
Importance: High importance as an aim of the project is modularity and not handling multiple sensor nodes in a single location would make the system flawed.

4.5.7 Climate data storage duration

The system should store a user's climate data for at least 6 months from the recording of that data.
Rationale: The system should store climate data long enough for historical data trends to be analysed and for data trends to be understandable for users, while ensuring that the database does not become too large or has bad performance.
Importance: Medium importance as not meeting this requirement would only affect the use of the historical data viewing features and not the viewing of recent climate data.

4.5.8 Sensor Recording Frequency

The sensor recording frequency should be as often every 5 minutes.
Rationale: The system should be able to record at a frequency that is useful and recent enough for users to make decisions based upon this data while maintaining the sensor node's battery life specified in non-functional requirement 10 and keeping the frequency inline or more frequent than existing systems.
Importance: High importance as out of date data would not be useful to users and would affect the suitability for the main aim of viewing current climate data.

4.5.9 Sensor Recording Frequency Consistency

The sensor recording frequency should be consistent with equal lengths of time between each time the sensor records data.
Rationale: The system should be able to record consistently as it would make the climate data trends difficult to understand and analyse if the intervals that the data was recorded at were different at each recording interval.
Importance: High importance as an inconsistent system would be confusing for users and meaning the system would only meet the project's aim of viewing current climate data some of the time.

4.5.10 Sensor Battery Life

The sensor nodes should have a battery life of at least 2 months before having to have their batteries replaced or recharged.
Rationale: The sensor nodes should be able to be left without interaction for a reasonable amount of time while balancing the affordability of the sensor nodes.
Importance: Medium importance as the system would still be functional with a short sensor node battery life but would have limited usability and affordability due to the need to replace or recharge batteries.

4.5.11 System Affordability

The system should be more affordable than existing solutions with the final system costing under £100 to implement.
Rationale: The existing systems are expensive, and the project's aim is to be more affordable and modular so a lower cost for the initial system is important and should be lower than the existing systems.
Importance: High importance as affordability is an aim of the system and the main limitation of the existing systems.

4.5.12 Installation Usability

Once the system has been built it should be easy to implement in a different location without technical changes or technical knowledge of the system.
Rationale: Having a system that requires technical changes during installation would limit the flexibility of the system and its usefulness for end users that would not have specific technical knowledge of this system.
Importance: Medium importance as the system would still be functional, but less usable than existing systems that only require limited technical knowledge to pair their sensors and base stations to their Wi-Fi and to position the sensor nodes.

4.5.13 Maintenance Usability

The system should be simple to maintain once implemented, without requiring technical knowledge.
Rationale: Having a system that requires technical changes after installation would limit usability by end-users that would not have the specific technical knowledge of the system.
Importance: Medium importance as the system would still be functional, but less usable than the existing systems that only require battery replacing or the systems that are completely independent as they are recharged through integrated solar panels.

4.5.14 Sensor Node Distance

The sensor nodes should be able to send the climate data while within 100 un-obstructed metres of the device that makes the climate data viewing on the website.
Rationale: The sensor nodes should be placed in a variety of locations within the local area, while being able to communicate with the data storage location. 100 metres in best

conditions was selected to keep in parallel with the existing systems and due to the modularity of this system, many sensor nodes will likely be used in different locations.
Importance: Medium importance as the system would still function if the max connection distance is less than 100 metres, but the usefulness of the system would be limited if it is attempted to be used in a larger area where the max connection distance is not sufficient for the best placement of the sensor nodes.

4.5.15 User Data Security

The user data shall be kept secure from malicious or accidental access by individuals other than the user that the data belongs to.
Rationale: Sensitive user data such as passwords should be kept secure as these being accessible by other individuals would pose security risks if their password were used in other systems.
Importance: Medium importance as it does not affect the system's performance or usability, but security should be important due to regulations such as GDPR and the consequences of user data being retrieved by other individuals.

5. Design

This section describes the approach taken in designing the system including the design methods and processes. The outcomes of each aspect of the system's design is then provided along with analysis of its reasoning, suitability, and effectiveness.

5.1 Approach

The approach taken was to select hardware based upon the requirements, these constrained the hardware suitable for the designs. The selected hardware is product specific as compatibility between hardware components and software can only be guaranteed once the specific hardware specifications. The design of the software architectures was then completed based on the compatibility with the selected hardware. This did mean that some hardware choices then needed changing when no suitable or compatible software could be identified.

5.2 Overall Design

The overall design structure was selected based upon the evaluation of the requirements and the possible service models that the system could offer. This resulted in the design displayed in figure 2. The sensor nodes send their recorded climate data wirelessly to the base station which processes the data via an API and puts it into the database, this lets sensors be placed in ideal measuring locations with low hardware specifications. The base station hosts a website via a web-server that provides users access to the climate data by visiting the web-server’s IP address in a web browser. Though this does require port forwarding the HTTP server on the Wi-Fi router so that the website can be accessed from the internet, this could impact the system’s usability by less technical users. In this design each implementation of the system is independent, this provides data privacy as all data is stored locally within the base station. The base station provides management of multiple sensor nodes and the dynamic addition and removal of the nodes.

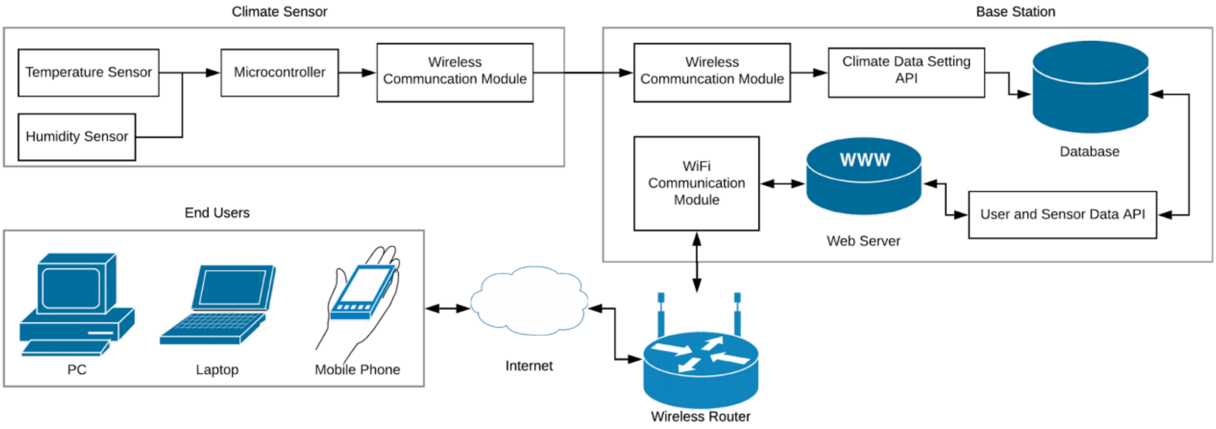


Figure 2. System Structure

Two alternative designs were considered, the first design reflected an external service model where a cloud service provides the database, API, and web-server, while the base station only performs node management and sending the data to the API. This has advantages including simpler multiple system installations, less technical knowledge required for installation, and more affordable individual system installations. The second design combined the chosen design with the local database, API, and website but with a cloud service providing additional analysis and trends of the climate data. The chosen design was selected over these alternatives as it provides a simple client-server model that is completely local to the implementation providing more straightforward interactions between components and greater data privacy. It fulfils all the requirements relevant to the overall design. Additionally, it is better suited to a prototype implementation of the project as it has fewer separate components and requires less communication between these components. Though the affordability and more technical knowledge required by this model are a necessary downside when compared to the disadvantages of the other models. This model provides a good starting point and can be expanded to a second alternative design to add additional secondary features such as climate data trends analysis**.** Though there are some downsides such as the base station will require more processing power and memory to handle multiple services, this will increase the cost of the chosen device. Additionally, the design requires multiple separate installations if a user wants to measure the climate in multiple geographical locations, as this requires multiple base stations each with their own website, settings, data, and sensor nodes.

5.3 Sensor Device

5.3.1 Sensor Device Hardware Requirements

There are design requirements and system requirements that affect the choice of climate sensor node hardware. This results in the main functionality of the nodes requiring working indoors and outdoors, recording temperature and humidity, wireless communication to the base station, battery power that lasts at least 2 months. Based on this functionality, the following hardware components are required: low-power microcontroller, a wireless communication module, temperature sensor, humidity sensor, and a waterproof case.

5.3.2 Sensor Device Microcontroller and Radio

The Adafruit Feather 32u4 433mhz RFM69HCW microcontroller was selected as it supports interfacing with the sensors via GPIO and sending the data wirelessly via radio communication to the base station. The 433mhz radio frequency was selected as it requires no license and provides long-range. The radio module was also supported by the potential devices for the base station. Additionally, this microcontroller supports battery power and recharging via a micro-USB cable, this was not supported by alternative devices with more power consumption such as a Raspberry Pi. While separate microcontroller and radio modules were considered they added additional costs and complexity. Other wireless communication technologies were not considered such as Wi-Fi and Bluetooth due to the high-power consumption and short ranges provided by them, respectively. The radio provided supports ranges of over 100 metres in best conditions, while being packet-based and connectionless so the microcontroller and radio can sleep without affecting communications. The microcontrollers with LoRa radios were considered but have a range that may be too large for their application in this project so are not suitable as this range could reduce consistency and increase the power consumption. The 32u4 and M0 variants for Adafruit Feather were considered but the 32u4 variant was considered more suitable due to the lower power consumption caused by a slower processor and less flash memory and ram. The microcontroller is Arduino compatible thus supporting all Arduino libraries which will facilitate more efficient software design and implementation, instead of fully bespoke development.

5.3.3 Sensor

The BME280 sensor was selected to provide the weather measurements as it measures temperature, humidity, and atmospheric pressure while being interfaceable with the microcontroller via i2c.This sensor was selected compared to alternatives as it provides good accuracy and range within those specified by the requirements, though it has higher costs while the other sensors had low costs with worse accuracies and ranges. Additionally, the sensor has low power consumption and supports a sleep mode to ensure the longer length of the microcontroller’s battery.

5.3.4 Microcontroller Battery

A 3000mAh lithium polymer battery was selected as it provided the best power to cost balance of the researched batteries. Additionally, it supports the recharging functionality provided by the microcontroller. Based on the microcontroller's power specifications the battery also selected as its size potentially provides months of battery life.

5.3.5 Case

The Dri-Box FL-1859-200 IP55 Weatherproof Box (Dri-box, 2011) was selected to house all the sensor node's hardware. This case is large enough to fit all components inside without being too large to have ideal node placement and is waterproof while having holes to ensure airflow which is required for the sensors to provide accurate measurements.

5.3.6 Electronics Layout

The electronics diagram in figure 3 shows the connections between the hardware components.

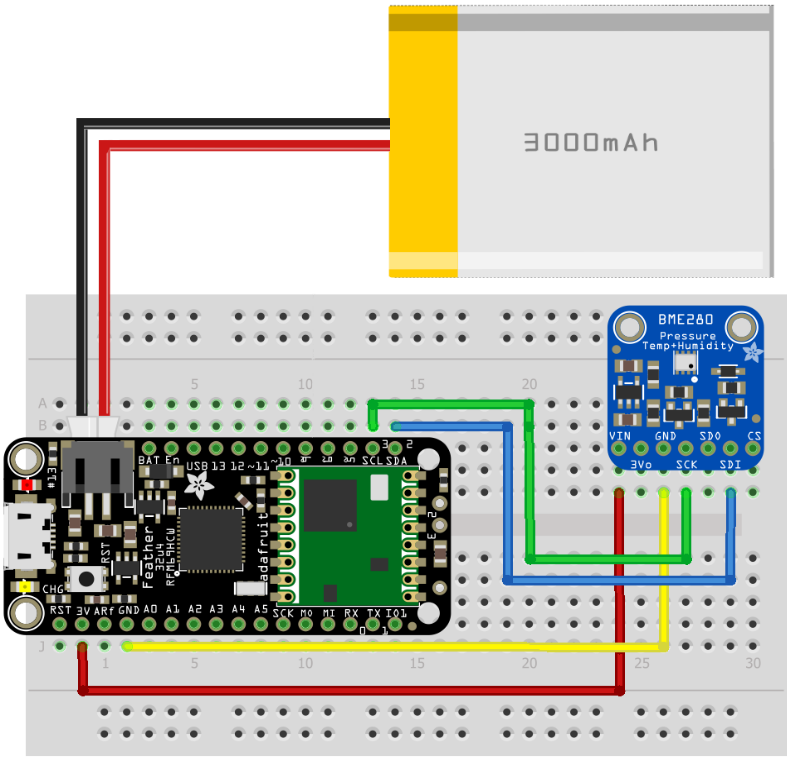


Figure 3. Electronic layout of the sensor node

5.3.7 Microcontroller Software

As the microcontroller is Arduino compatible it can take advantage of the many libraries available to facilitate component interfacing and improve the implementation efficiency. It supports the C language among alternatives such as MicroPython. The flowchart of the software is in figure 4, this is required to manage radio communication with the base station, retrieve measurements from the sensors, send measurements to the base station, and sleep when not required.

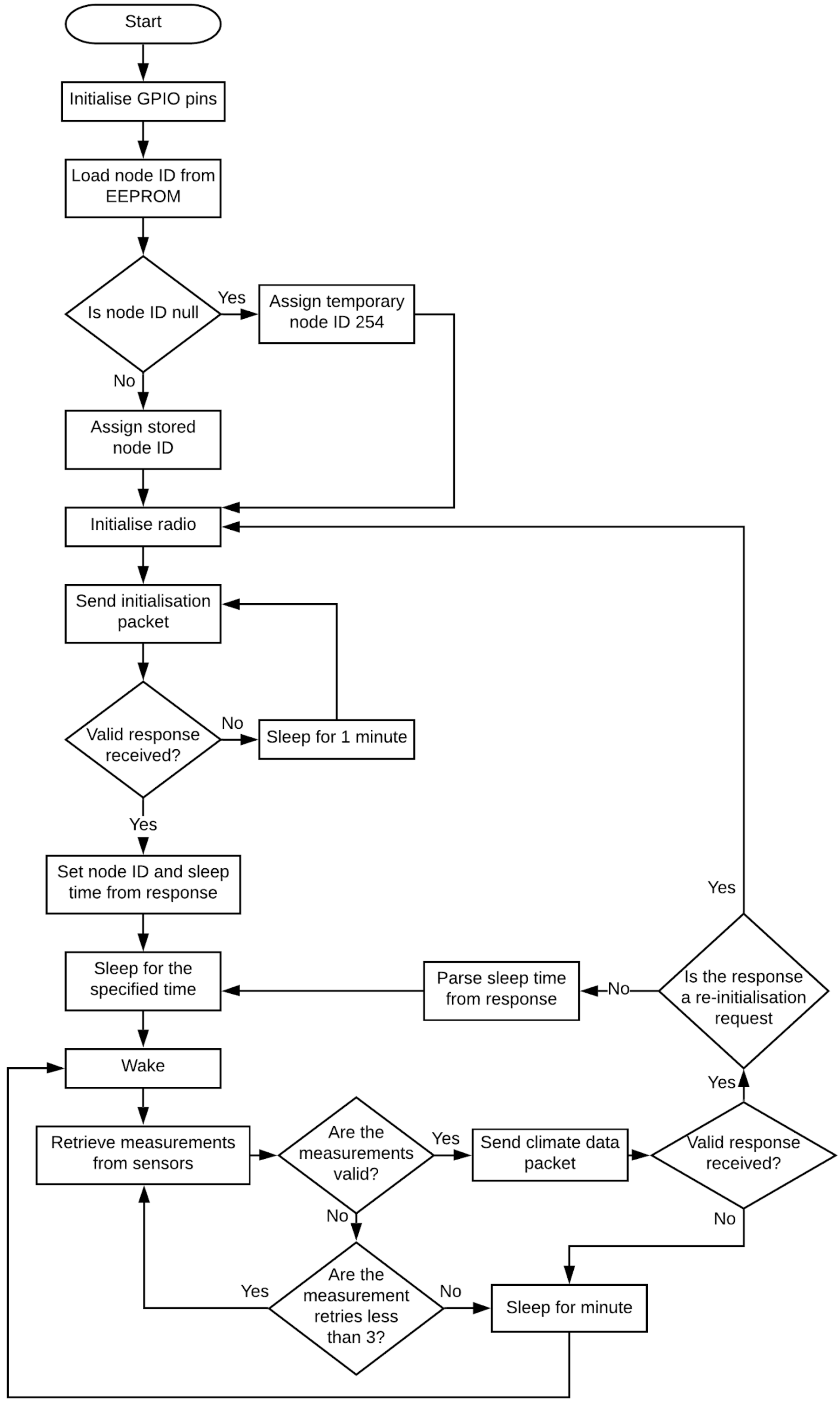

Start
Initialise GPIO pins
Load node ID from EEPROM
Is node ID null
Yes
Assign temporary node ID 254
No
Assign stored node ID
Initialise radio
Send initialisation packet
Valid response received?
No
Sleep for 1 minute
Yes
Set node ID and sleep time from response
Sleep for the specified time
Wake
Retrieve measurements from sensors
Are the measurements valid?
Yes
Send climate data packet
Valid response received?
Yes
Is the response a re-initialisation request
Yes
No
Parse sleep time from response
No
No
Are the measurement retries less than 3?
Yes
No
Sleep for minute

Figure 4. Flowchart of the sensor nodes

5.4 Base Station

5.4.1 Base Station Hardware Requirements

The base station is required to communicate with the sensor nodes, host the database, website and API while having a connection to a Wi-Fi network. Thus, the base station needed a small single-board computer and the radio module used by the sensor nodes.

5.4.2 Base Station Computer

A Raspberry Pi 3 A+ was selected as it provides the processing power, memory and storage required to host the software along with the radio interfacing and Wi-Fi connection. A microcontroller was not possible due to software requirements, this also meant that the Raspberry Pi Zero W was not suitable due to it having half of the computing power available with the 3 A+. Additionally, the Pi 4 was not considered as it had higher costs and computing power that was too high for the software requirements.

5.4.3 Radio

The 433mhz RFM69HCW radio module was chosen as the sensor nodes and base station require the same radio module and frequency to communicate, this module can interface with the Raspberry Pi over SPI.

5.4.4 Case

Any case that could fit the Raspberry Pi and radio module was suitable. Thus, the Therpin DIY Waterproof Electronic Junction Box (Therpin, 2017) was selected as it provides the space required to house the hardware, though it did not require waterproofing it helps ensure that the components are protected.

5.4.5 Electronics Layout

The electronics diagram in figure 5 shows the connections between the base station hardware components.

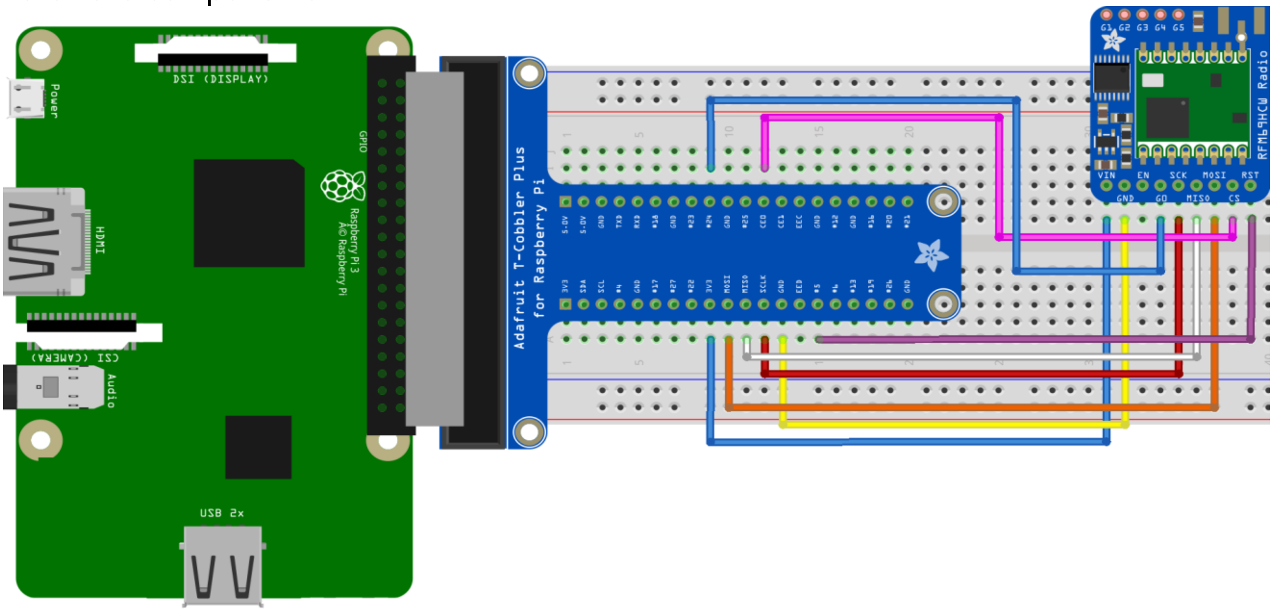


Figure 5. Electronic layout of the base station

5.4.6 Operating System

The Raspberry Pi will use the Raspbian operating system as it ensures compatibility with the hardware while having slightly higher computing power required than alternative lightweight Linux distributions.

5.4.7 Required Software Subsystems

The software will consist of the following subsystems, sensor node communication manager database, API, Wi-Fi connection management, and a web-server to provide network access to the API and website. These designs have been given their own sections due to the many design choices and subsections used. The API design is described in section 5.6, the database in section 5.7, and the website in 5.8.

5.4.8 Sensor Node Communication Manager

The communication manager communicates with the sensor nodes to process initialisation requests, tracking the currently active nodes, receiving the climate data, and then send data to the API, the logic flowchart of the program is shown in figure 6.

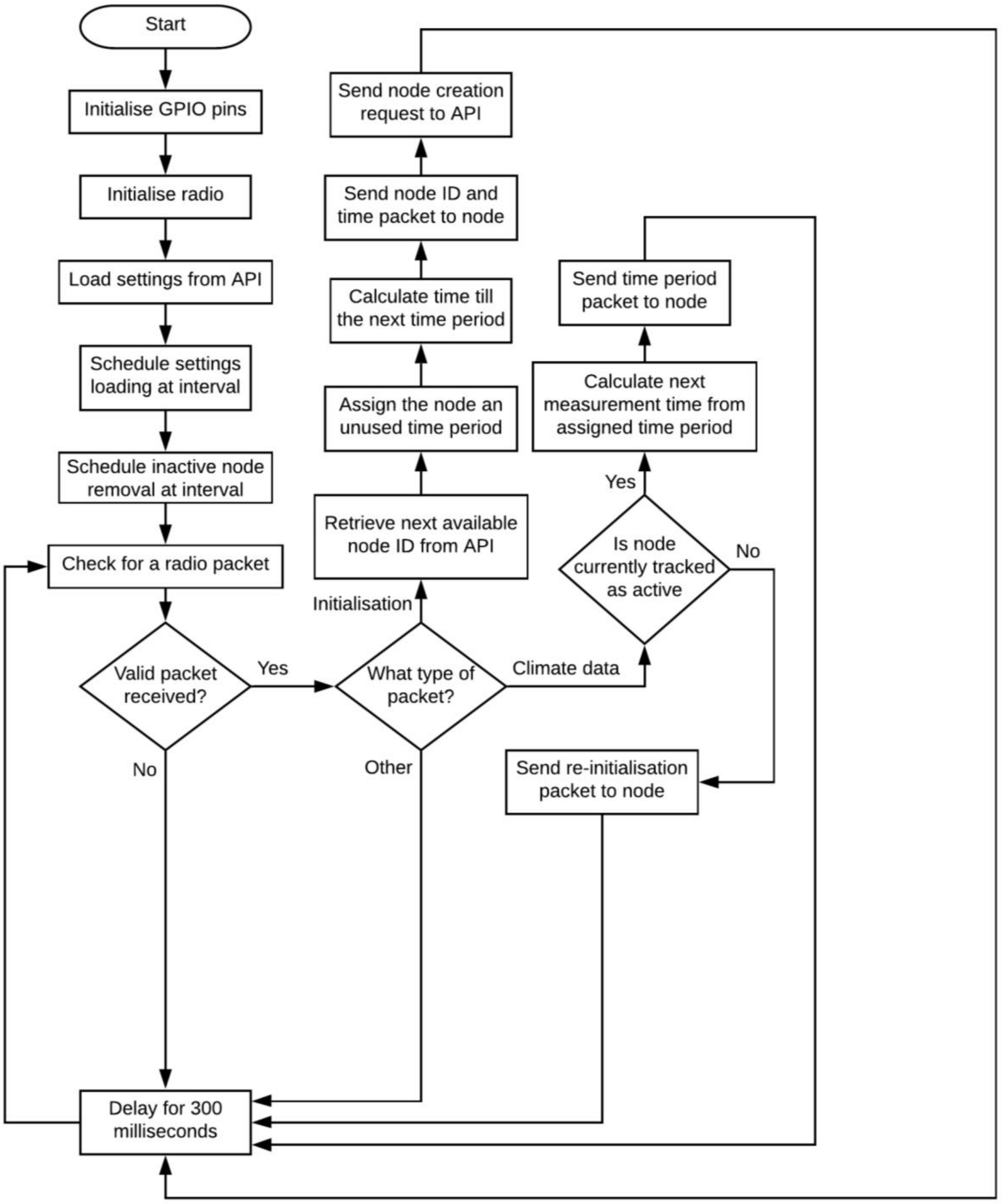

Start
Initialise GPIO pins
Initialise radio
Load settings from API
Schedule settings loading at interval
Schedule inactive node removal at interval
Check for a radio packet
Valid packet received?
Yes
No
What type of packet?
Initialisation
Climate data
Other
Retrieve next available node ID from API
Assign the node an unused time period
Calculate time till the next time period
Send node ID and time packet to node
Send node creation request to API
Is node currently tracked as active
Yes
No
Calculate next measurement time from assigned time period
Send time period packet to node
Send re-initialisation packet to node
Delay for 300 milliseconds

Figure 6. Flowchart of sensor node communication manager

5.4.9 Web-server

The web-server is required to host the website and API so that they are accessible within the base station's Wi-Fi network and on the internet. The selected technology should be suitable for the Raspberry Pi and not cause large additional overheads on the CPU and RAM. Additionally, the Web-Server is not required to handle large volumes of requests due to the limited number of potential users concurrently accessing the web-server

5.4.10 Wi-Fi Manager

The Wi-Fi manages which Wi-Fi network the base station is connected to, this is required so that the user can input their local Wi-Fi network during initialisation by creating a Wi-Fi hotspot using a mobile phone which the base station initially connects to. The logic of this program is shown in figure 7.

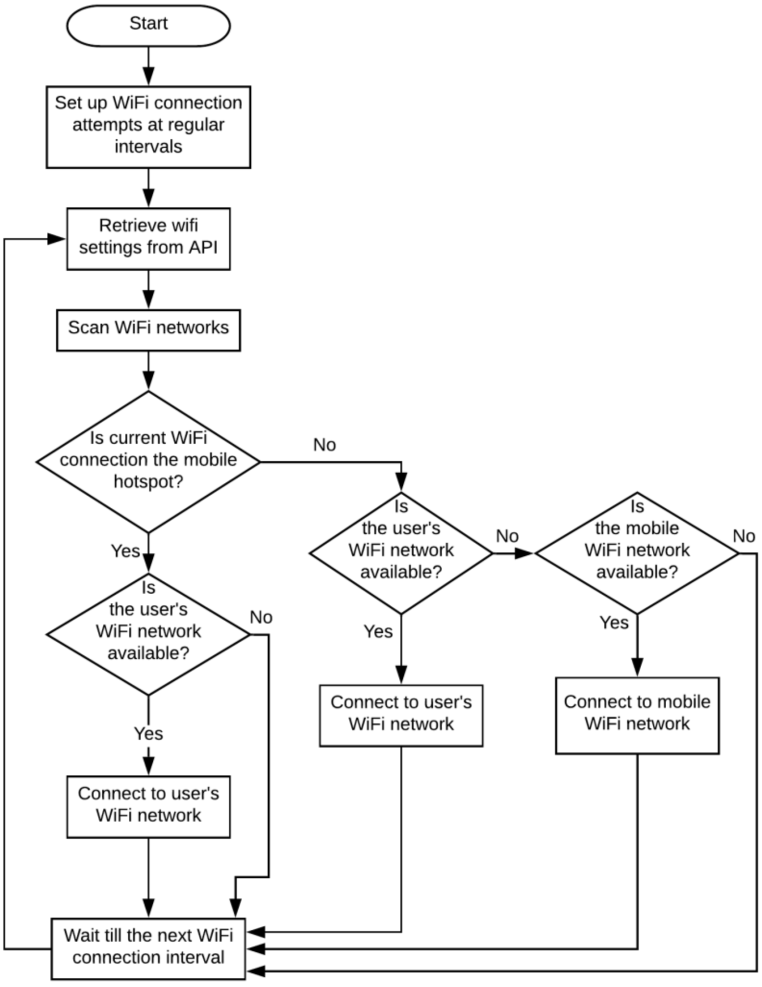

Start
Set up WiFi connection attempts at regular intervals
Retrieve wifi settings from API
Scan WiFi networks
Is current WiFi connection the mobile hotspot?
No
Yes
Is the user's WiFi network available?
No
Yes
Connect to user's WiFi network
Is the user's WiFi network available?
No
Yes
Connect to user's WiFi network
Is the mobile WiFi network available?
No
Yes
Connect to mobile WiFi network
Wait till the next WiFi connection interval

Figure 7. Flowchart of the Wi-Fi manager

5.5 Radio Protocol

5.5.1 Protocol Requirements

A protocol was designed to facilitate packet-based radio communication as wireless communication is required between the base station and the sensor nodes. This protocol is needed to ensure the correct handling of handshaking, collisions, and data integrity errors.

5.5.2 Packet Structure

To improve implementation efficiency, the LowPowerLab RFM69 radio library (LowPowerLab, 2020) is used to provide an existing packet structure. This structure has a total packet payload size of 65 bytes, with 4 of these being used by the header. This leaves 61 bytes for the main data, this is due to the limit for AES based encryption, which is supported by the library, the packet structure is shown in figure 8.

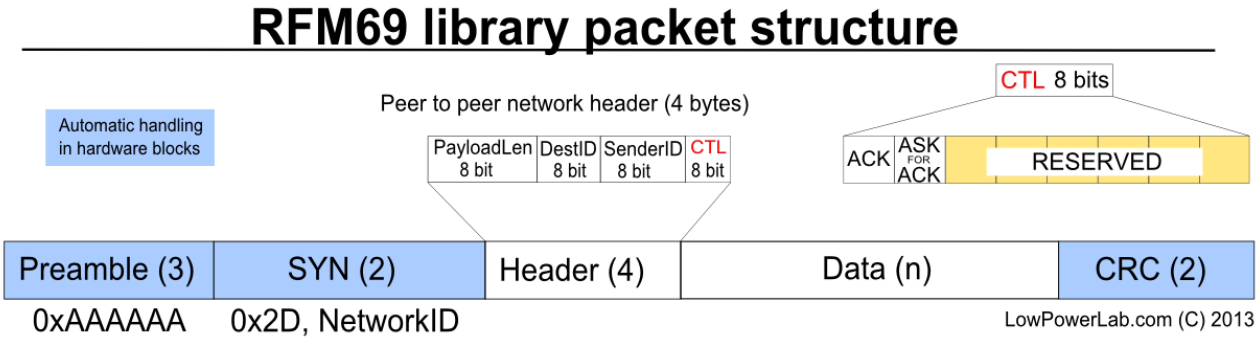


Figure 8. LowPowerLab RFM69 packet structure (LowPowerLab, 2013)

5.5.3 Error detection

The protocol will use error detection to ensure that packets are valid, as corrupted data could cause issues with software in the sensor nodes and base station, additionally, inaccurate measurements could cause misunderstandings in the weather trends. The LowPowerLab provides a cyclic redundancy check (CRC) which will be used to verify the integrity of the packet data. If a CRC fails or an acknowledgement has not been received the protocol will attempt to retry 3 times to see if a successful communication can be completed.

5.5.4 Packet Collisions Handling

As the system handles multiple sensor nodes, the protocol is designed to avoid packet collisions as the base station can only receive one packet at a time due to the limitations of radio communication. Multiple sensor nodes simultaneously communicating with the base station would cause interference and corrupt packets. Due to this a time period allocation system has been designed so that sensor nodes shouldn’t send packets at the same time. This uses time periods allocated to each sensor node so that they only send their climate data during their time period, thus collisions are avoided. This was selected due to the ease of implementation and scalability while being suited for the quick transmission times and limited packet numbers for the climate data. The nodes are assigned time periods by the base station once an initialisation request has been received from the node, this allows the base station to track allocated time periods. The base station then uses the time periods to generate the number of seconds the microcontroller will sleep for until it wakes and measures the climate. The sensor nodes will consider the time taken to complete other actions such as retrying sending data and retrying sensor data collection when using the milliseconds till the time period before sleeping.

5.5.5 Protocol Data Units

A set of protocol data units (PDU) were designed to standardise the packets across different devices, these have been listed individually with their packet structure.

5.5.6 Ack PDU

The acknowledgement PDU is used to send acknowledgements when a packet has been received to ensure that the sender is aware the packet has been received, thus the packet includes no main payload data.

5.5.7 Climate Data PDU

This PDU is used to send the climate data and thus requires most of the packet to be reserved for the measurement types and values, the structure is shown in table 2.

Packet section	Battery Voltage	Packet Type	Climate Data
Section size	7 Bytes	4 Bytes	50 Bytes

Table 2. Packet structure of the climate data protocol data unit

5.5.8 Initialisation PDU

This PDU is sent to the base station by a sensor node to request the data required for the node to start its measurements, this only requires the packet type as shown in table 3.

Packet section	Packet Type	Empty Padding Data
Section size	4 Bytes	58 Bytes

Table 3. Packet structure of the initialisation protocol data unit

5.5.9 Time-Period PDU

As time-periods are used to inform sensor nodes when they should send their next measurements, this PDU is used by the base station to send the milliseconds till the measurement time, this is required as the microcontrollers have no date or time synchronisation and must rely on the base station. The packet structure is shown in table 4.

Packet section	Packet Type	Start Time
Section size	4 Bytes	9 to 16 Bytes

Table 4. Packet structure of the time-period data unit

5.5.10 Node ID and Time Period PDU

This PDU is used to assign the sensor node a unique identifier and its initial measurement time once an initialisation request has been received, the structure is shown in table 5.

Packet section	Packet Type	Start Time	Node ID
Section size	4 Bytes	9 to 16 Bytes	4 Bytes

Table 5. Packet structure of the node ID and time period data unit

5.5.11 Exchange sequences

Exchange sequences have been described to ensure that specific packet sequences are followed between the devices, these are shown in figure 9.

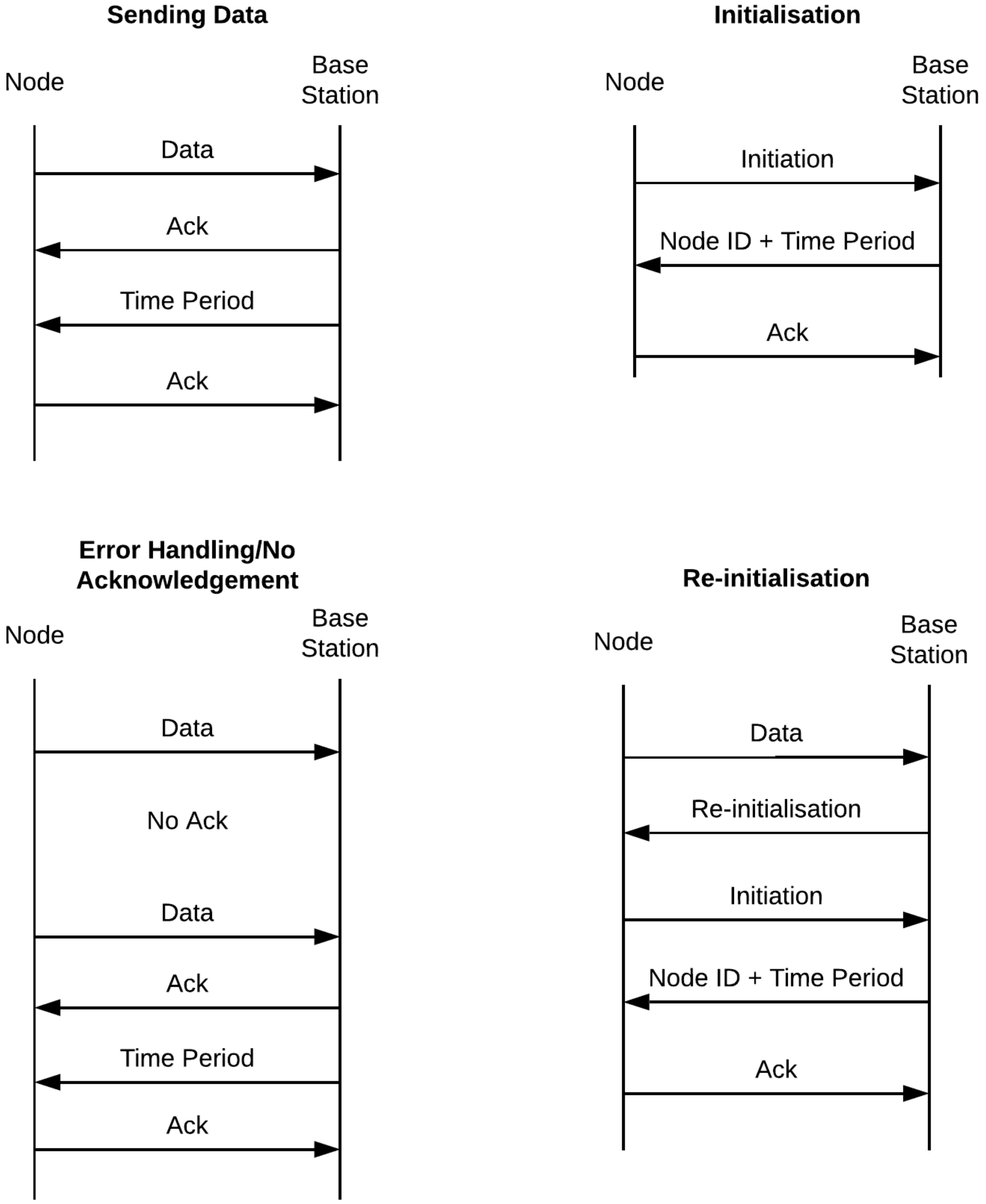


Figure 9. Radio packet sequences

5.6 API

5.6.1 Design approach

The API must interface with the database and act as a layer above the database by being able to create, read, update, and delete data. Based on the requirements and overall design it is required to handle requests internally from the base station's radio and Wi-Fi manager programs, while also handling requests resulting from users interacting with the website and requests from any connected automation services. While this could be provided by independent APIs each with limited functionality for their use case, it is more effective to use a single API to reduce complexity and processing overheads.

5.6.2 API architecture

The API endpoint design follows the REST architecture as it provides a consistent structure and constraints that ensure good performance and logic. This was selected as state is not required for any of the endpoints to achieve their functionality with most performing CRUD operations, additionally, it helps reduce the memory requirements. Due to the REST design constraints, the endpoints will need to provide a uniform interface, act as client-server communication, while being a layered system that is cacheable.

5.6.3 API Data Format

The API will receive and return JSON data across all endpoints as it provides easy to access hierarchical data structures that are similar in structure to the data in the database. JSON data can be easily constructed and read by programming languages using existing libraries.

5.6.4 Endpoint Structure

The endpoints have each been designed with a detailed documentation style to ensure that each endpoint can be implemented correctly due to the number of success and error possibilities. An overview of the endpoints as has been provided in table 6, with the full documentation in Appendix E.

Use of the RESTFUL architecture means that multiple endpoints will exist for some URLs as different types of requests can be performed based on the HTTP request type used. The post endpoints require valid JSON data to be sent as the requests body, while the get, patch, delete endpoints will use URL query strings to receive additional request configurations. Additionally, some endpoints require data to be placed in the URL such as the sensor ID when using the get sensor endpoint, these have been displayed in braces along with a variable name representing the required data.

Name	**Type**	**Authentication Method**	**URL**
Authenticate user	Post	None	/api/login
Logout access token	Post	JWT	/api/logout/access
Logout refresh token	Post	JWT	/api/logout/refresh
Refresh access token	Post	JWT	/api/token/refresh
Add new sensor	Post	API key	/api/sensors
Get all sensors	Get	JWT	/api/sensors
Delete all sensors	Delete	JWT	/api/sensors
Update sensor	Patch	JWT	/api/sensors/{sensor_id}

Get sensor	Get	JWT	/api/sensors/{sensor_id}
Delete sensor	Delete	JWT	/api/sensors/{sensor_id}
Sensor Data Setting	Post	API key	/api/sensors/{sensor_id}/climate-data
Sensor Climate Data Retrieval	Get	JWT	/api/sensors/{sensor_id}/climate-data
Sensor Climate Data Deletion	Delete	JWT	/api/sensors/{sensor_id}/climate-data
Account Creation	Post	JWT	/api//account
Account Updating	Patch	JWT	/api/account
Account Details Retrieval	Get	JWT	/api/account
Password Reset	Get	JWT	/api/account/actions/change-password
Next Available Sensor ID	Get	JWT	/api/sensors/actions/next-available-sensor-id
Settings Retrieval	Get	API key	/api/base-station-settings

Table 6. Overview of API endpoints

5.6.5 Authentication

The API requires two methods of endpoint user authentication, the first is use of an API key and will be used in the endpoints used by the internal subsystems, the second is JSON web tokens(JWT) which will be used by endpoints used by the website and connectable services. This is required so that the radio and Wi-Fi manager programs can access the API without knowing the user's credentials. The JWT authentication method was used as it provides users with tokens that give them access to the endpoints, these tokens can be configured to have limited time and be manually revoked. These tokens are provided once the user has provided the valid email and password to the login endpoint, they are then given an access token that provides access to the restricted endpoints. These tokens are then configured to last 30 days or will be revoked when a user chooses to logout. This method is stateless as all tokens are stored in the database and thus compatible with the RESTFUL architecture of the API and provides good security as the tokens can be safely stored client-side. The alternative method was Cookie-based authentication, but this was not suitable as it requires the API to use state which is not possible in a RESTFUL compliant API.

5.7 Database

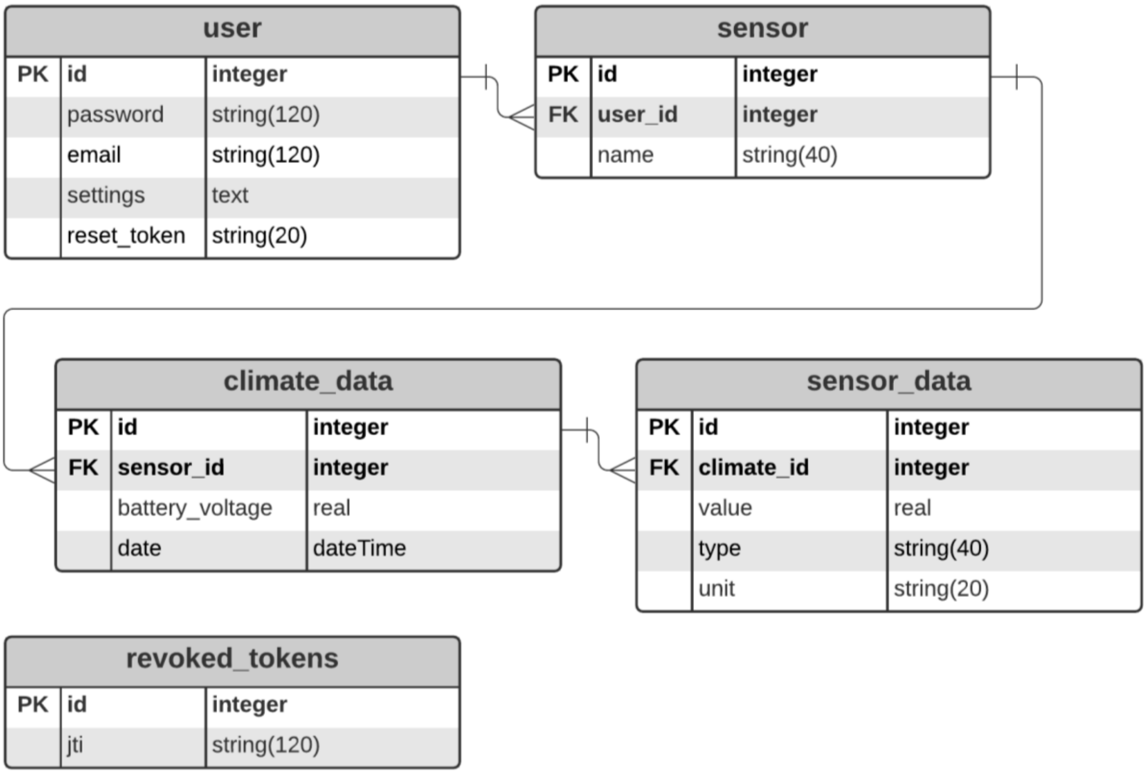
5.7.1 Database Architecture

The database provides non-volatile storage of the climate, sensor node and user data as specified by the requirements**.** A SQL or NoSQL database could store this data but an SQL database has been selected to ensure compatibility with the Raspberry Pi and

Raspbian OS used in the base station as none of the compatible NoSQL databases researched provided the date support and complex queries required.

5.7.2 Database Structure

The database structure consists of 5 tables with one to many relationships as shown in figure 10.



user		
PK	id	integer
	password	string(120)
	email	string(120)
	settings	text
	reset_token	string(20)

sensor		
PK	id	integer
FK	user_id	integer
	name	string(40)

climate_data		
PK	id	integer
FK	sensor_id	integer
	battery_voltage	real
	date	dateTime

sensor_data		
PK	id	integer
FK	climate_id	integer
	value	real
	type	string(40)
	unit	string(20)

revoked_tokens		
PK	id	integer
	jti	string(120)

Figure 10. Database Entity Relationship Diagram

The data types and sizes used by each of the tables were selected based on expected values that could be stored within it during normal use of the system. The main choices involved using short strings to limit the size of the databases and using float values for sensor data measurements as all potential measurement types found in the requirements background research could be represented by float values. The full choice analysis and justification have been included in Appendix F**.**

5.7.3 Password Hashing and Salting

The passwords are hashed using the bcrypt algorithm, this ensures that the only the user knows the plain text of the password with only the hashed password being stored. In the login API endpoint, the user’s input password is hashed and compared to the hashed password in the database. Alternative hashing algorithms including PBKDF2, Scrypt and WHIRLPOOL were considered, while older and bypassed algorithms such as MD-5 and SHA-3 were not.

Additionally, salting is then used with the hashing function as a string is added to the password string before being put into the database. This increases the complexity of the hashed password string, this increases system security. Only one salt string is used as only one user account exists, this will be generated by the hashing library used in the implementation.

5.8 Website

5.8.1 Website Architecture

The website provides the sensor node management, climate data viewing, and configuration options as per the requirements. This uses a modern client-side framework to provide single-page-app (SPA) and model-view-controller (MVC) functionality. This helps ensure good usability as page navigation is short as it is completed locally with the Web-Server only providing the root application code to the browser. This was selected when compared to plain JavaScript client-side rendering and server-side rendering. Using plain JavaScript would significantly increase the development time required and add unnecessary complexity. Server-side rendering would have required the web-pages to refresh when users complete interactions such as changing the date range for the historical climate data graphs, these require new data from the API once initiated. This is not user-friendly, as in the SPA all data is loaded from the API and can be displayed without refreshing. All website technologies provide mobile, desktop and laptop support, with SPA having the closest experience to mobile applications. Additionally, the use of MVC is suited to the data-focused use of this website as the UI can be easily updated once new data is received from the API.

5.8.2 Sitemap and Page Hierarchy

Based on the website requirements the following pages and functionalities are needed: login page, register page, settings page, reset password, sensor node management list, recent climate data displaying, and historical climate data visualisation.
While the functionality could exist in individual pages, a dashboard-style UI has been designed to provide multiple functionalities while reducing the number of pages required, and the time and complexity of implementation.

A sitemap was designed to show the page hierarchy, this reflects the low number of individual pages required by the design.

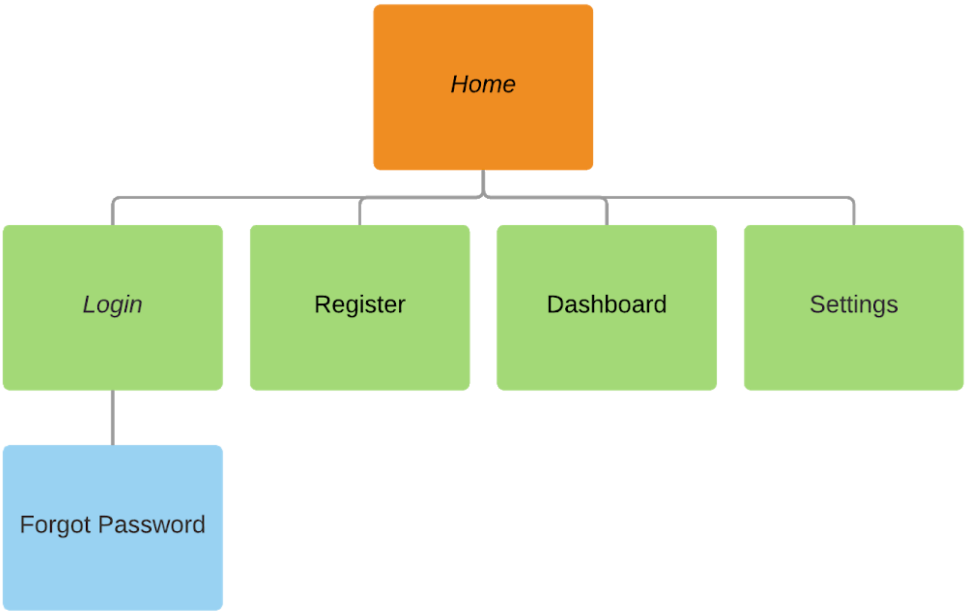
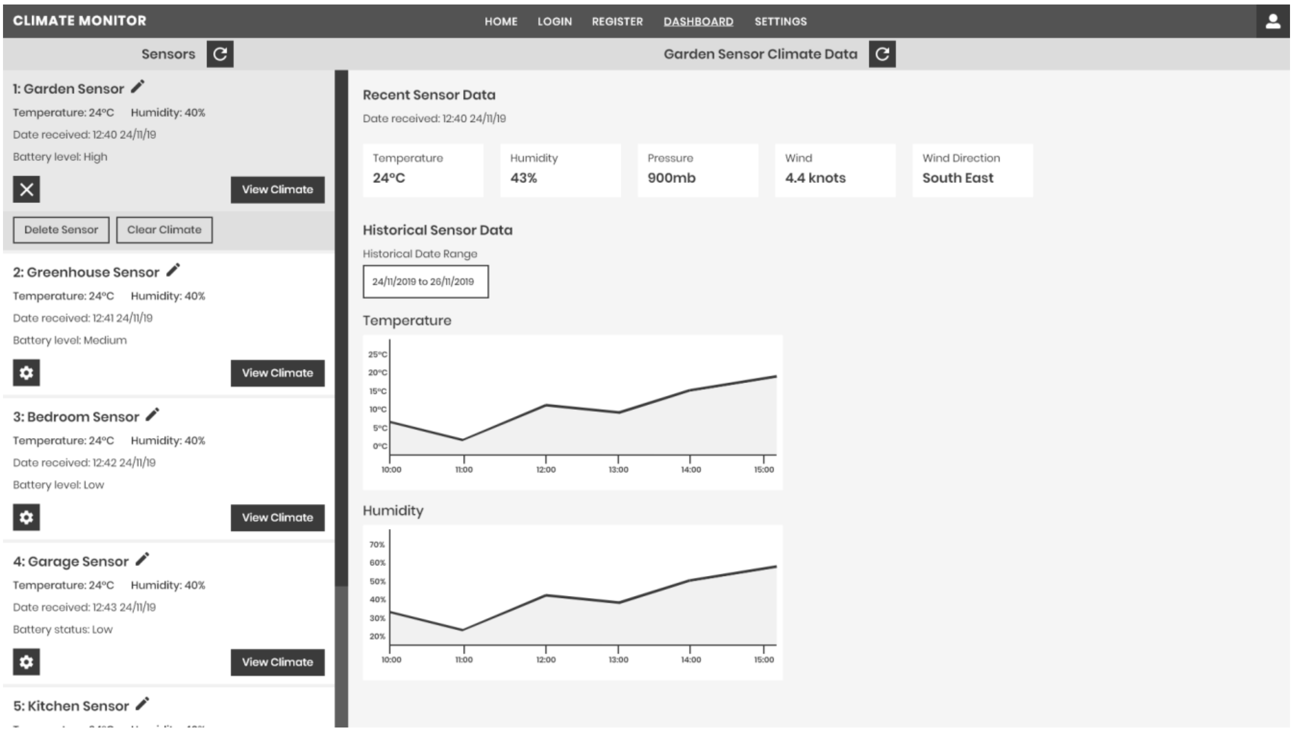


Figure 11. Website sitemap with page hierarchies

5.8.3 UI Design

Website's user interfaces were designed based on the pages identified in the sitemap by using Adobe Experience Design. This tool provides user interface designing functionality with features including reusable components and smart resizing providing increased efficiency when designing UI components that are used in different contexts. Mobile and desktop designs were completed to ensure good usability in the mobile user interface with larger inputs and increased element spacing to avoid miss-presses. These designs were based on the UI elements established within wireframe and high-level style sheets, this detailed the text styles, colours, shadows, and the different states for the interactive elements.

Initially, wireframes were designed for each page to give an outline of the page's structure and content. This facilitated iteration of the interfaces to ensure that all required functionality was designed with good usability. These wireframes only considered the layout, text content, interactable elements and grayscale shades. Figures 12 and 13 give an example of the created wireframes, with the full wireframe designs available in Appendix F.



1: Garden Sensor
Temperature: 24°C Humidity: 40%
Date received: 12:40 24/11/19
Battery level: High

2: Greenhouse Sensor
Temperature: 24°C Humidity: 40%
Date received: 12:41 24/11/19
Battery level: Medium

3: Bedroom Sensor
Temperature: 24°C Humidity: 40%
Date received: 12:42 24/11/19
Battery level: Low

4: Garage Sensor
Temperature: 24°C Humidity: 40%
Date received: 12:43 24/11/19
Battery status: Low

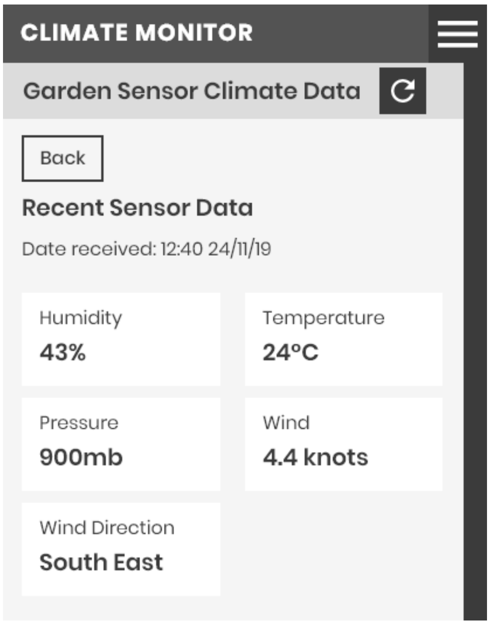
5: Kitchen Sensor

Date received: 12:40 24/11/19

Temperature	Humidity	Pressure	Wind	Wind Direction
24°C	43%	900mb	4.4 knots	South East

Historical Date Range
24/11/2019 to 26/11/2019

Figure 12. Desktop dashboard wireframe UI design



Recent Sensor Data

Date received: 12:40 24/11/19

Humidity
43%

Temperature
24°C

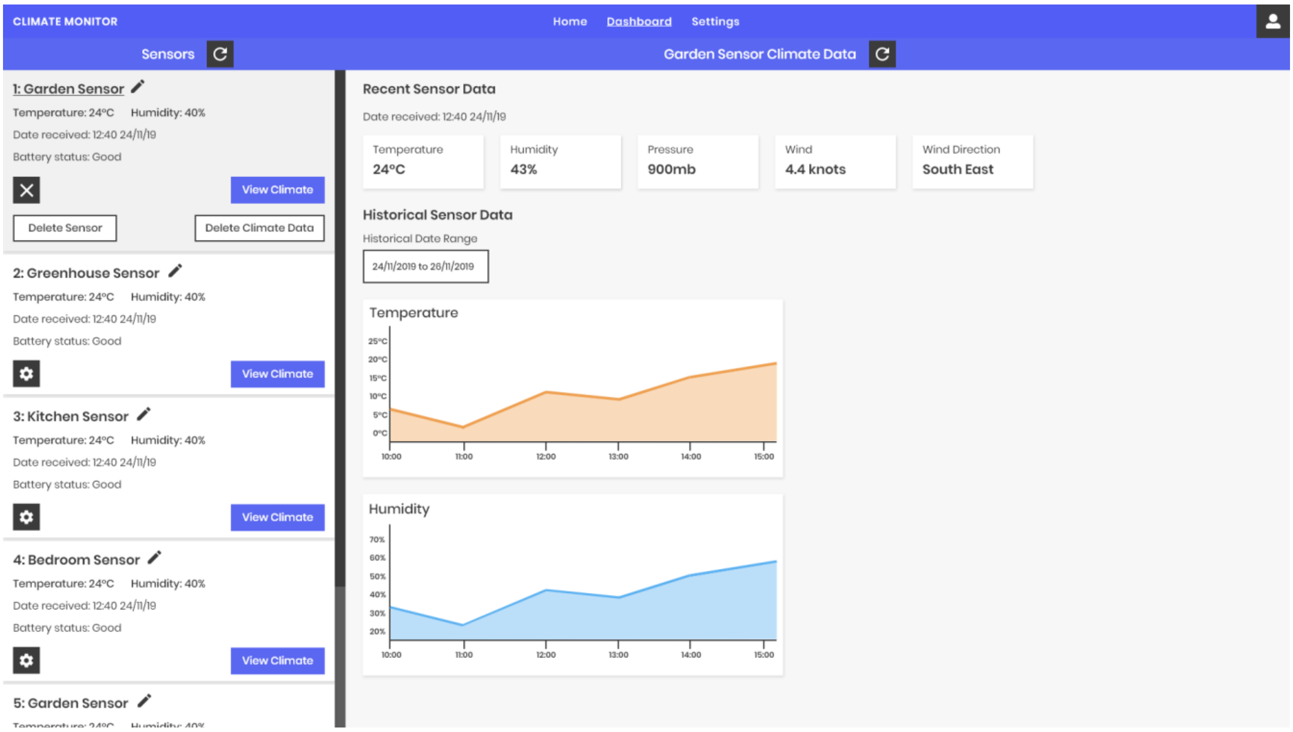
Pressure
900mb

Wind
4.4 knots

Wind Direction
South East

Figure 13. Mobile dashboard wireframe I UI design

High-level user interface designs were created from the wireframes, these expanded the designs to contain the final colours, icons, images, and complex details such as shadows. Figures 14 and 15 are examples high-level design with the all designs available in Appendix G



1: Garden Sensor
Temperature: 24°C Humidity: 40%
Date received: 12:40 24/11/19
Battery status: Good

2: Greenhouse Sensor
Temperature: 24°C Humidity: 40%
Date received: 12:40 24/11/19
Battery status: Good

3: Kitchen Sensor
Temperature: 24°C Humidity: 40%
Date received: 12:40 24/11/19
Battery status: Good

4: Bedroom Sensor
Temperature: 24°C Humidity: 40%
Date received: 12:40 24/11/19
Battery status: Good

5: Garden Sensor
Temperature: 24°C Humidity: 40%

Recent Sensor Data

Date received: 12:40 24/11/19

Temperature
24°C

Humidity
43%

Pressure
900mb

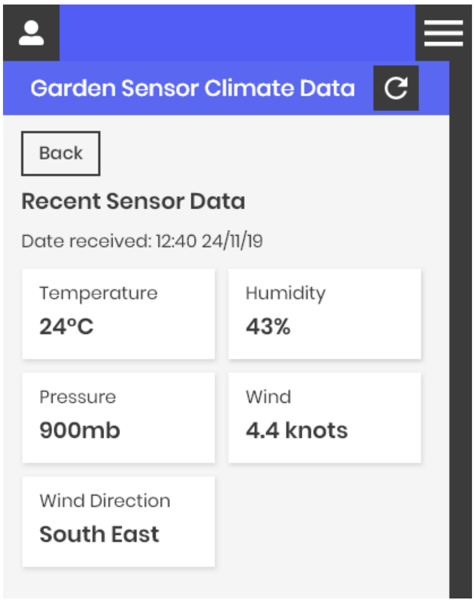
Wind
4.4 knots

Wind Direction
South East

Historical Sensor Data

Historical Date Range
24/11/2019 to 26/11/2019

Figure 14. Desktop dashboard high-level UI design



Recent Sensor Data

Date received: 12:40 24/11/19

Temperature
24°C

Humidity
43%

Pressure
900mb

Wind
4.4 knots

Wind Direction
South East

Figure 15. Mobile dashboard high-level UI design

6. Implementation

This section describes the implementation decisions and the reasoning for choosing them, including the implementation environments, algorithms, and data structures along with other considerations such as usability. The source code for the system is available in Appendix C.

6.1 Sensor Nodes

The first step of the implementation was creating the sensor nodes, their program was written in C with the Arduino IDE (Arduino Software, 2020), this was selected over other languages such as MicroPython due to a large number of libraries available within the Arduino. This included the LowPower (LowPowerLab, 2018) and RFM69 (LowPowerLab, 2020) libraries that were used to minimise battery usage while the nodes are sleeping between climate measurements. Additionally, the radio library provided a packet structure as described in the design document along with useful processing functions. The automatic power control provided by the radio library was used to reduce the radio transmission power when the node is closer to the base station. The Adafruit Unified Sensor Driver (Adafruit, 2020) and Adafruit BME280 (Adafruit, 2020) libraries were used to configure and read data from the BME280 sensor, as well as providing a forced measurement mode to reduce power usage while the node is sleeping. EEPROM was used to store the node ID when the node is powered off, this gives permanent storage with a limited number of reads and writes, but it supports enough that restarting the node is insignificant.

6.2 Base Station

The next step involved creating the radio communication manager, API, and database. This started with setting up the Raspberry Pi, along with configuring the Raspbian operating system that was selected for compatibility with the required hardware and software. The radio module was connected via GPIO pins and is controlled by a Python program that handles messages received from the sensor nodes, it also tracks the active nodes and manages the node spacing as designed in the radio protocol. The algorithm used to assign the measurement time-periods for the sensor nodes ensures that the nodes are spread out across the measurement time-period to reduce the likelihood of radio packet collisions. All python code within the implementation was written using PyCharm (JetBrains, 2019), with pip (Python Packaging Authority, 2020) to install Python libraries, and the venv module was used to create virtual environments so that the libraries are only available to each of the python programs. Next, the Apache web-server was implemented to serve the API and website, this involved setting up the Web Server Gateway Interface (WSGI) program to forward web-server requests to the API application. Apache was used as the Flask development server is not intended for use in production environments. Additionally, a static IP was configured to ensure that the API and website IP address is consistent regardless of the connected Wi-Fi network.

The API was built using Python 3 with Flask (Ronacher, 2020) providing a straightforward program structure and integrations. The use of JSON data structures to represent objects once received from the database facilitated straightforward API endpoint consumption by other system components. The database was implemented using SQLite as it is an in-file database with good performance and low memory usage. The SQL database enabled the use of the flask-sqlalchemy library for object-relational mapping of the required database data structures, this allowed for the quick creation and management of objects within the database with straightforward interfacing within the API. Additionally, the flask-restful (Bayer, 2019) library was used to provide a Restful API structure without manually ensuring restful characteristics. To ensure the correct structure of data being sent to the API the Marshmallow (Loria, 2019) library was used to validate the keys and values of the JSON data, this also provided errors generated directly from the validation failures. The user settings were stored as JSON strings as per the design, this allowed for multiple settings values to be introduced over the implementation of the system's functionality and support new settings in the future. Additionally, this provided easy transport of the user's settings to the other components. The algorithms used in the API endpoints were optimised to reduce the response time, this was a concern for the deletion endpoints which perform a batch deletion on many objects and different tables at once.

The PassLib (Collins, 2019) library provided an implementation of bcrypt to hash the user's passwords and provide salt generation. To limit the access of the API endpoints to authorised users the Flask JWT Extended (Gilbert-Bland, 2019) library provided an implementation of JWT. The Flask-cors (Dolphin, 2019) library allows cross-origin requests so that the website could consume the API. To parse dates the dateutil (Ganssle, 2019) library was used, this provided management of timezone-naive and timezone-aware date objects. Postman (Postman Inc, 2020) was used to test the API's responses during implementation, with the storing and configuration of many API requests.

The Wi-Fi manager used Node.JS to manage the base stations connection to the local Wi-Fi network. This involves an algorithm based on the available networks and current system status to select the correct network. This required changing the Rasbian's operating system to use network manager, this provides an interface to connect and disconnect to networks. The node-wifi (Friedrich, 2019) library was used to implement this interface.

The Apscheduler (Grönholm, 2019) library was used in the API and radio program, and Node intervals in the Wi-Fi manager to schedule the running of functions at intervals in the background. The wifi manager and radio program both retrieve the user's settings at regular intervals via HTTP requests to the API; this ensures that they stay up to date with the settings used in the API, database, and website. Though this results in unnecessary communication when the settings have not changed, instead event-based communication should have been used to ensure that the updated settings are received by the other components immediately after they have been changed. This could have been achieved using web sockets.

6.3 Website

The final step was creating the website, this involved using Vue.JS (You, 2019) to create a SPA with vue-router (vuejs, 2020) providing local page routing without requiring the web-server to provide multiple pages. Vuex (vuejs, 2019) was used to ensure that local and global data structures remained separate while being easily accessible when required. The user interface was built according to the final designs, though some changes were taken when the existing design elements were inadequate such as the battery status indicators. The website used the node package manager (npm Inc, 2020) to manage the libraries used by the website. The initial files, folders, and plugins were set up using Vue CLI (vuejs, 2019) which provides scaffolding functionality to speed up implementation, this included the Webpack (webpack, 2019) config, development server, and code linting. Additionally, the hot-reloading functionality was enabled allowing for the quicker testing of components without requiring full page reloads to see code changes.

Many open-source libraries were used to speed up development. This included vue-chart.js (Juszczak, 2019) to provide the climate data charts, Vcalendar (Reyes, 2019) for the date picker component, Webpack for building JS, HTML, and CSS files based on dependencies, Autoprefixer (Tidelift, 2019) to additional CSS rules to ensure compatibility and usability across modern and older browsers, Date-fns (Koss, 2019) for date parsing and date generation, Vue-Axios (Nguyen, 2019) for its interface to complex HTTP requests, and Vue-toasted (Sadikeen, 2019) for creating and managing toast notification components.

Additional implementation was carried out to ensure the usability of the mobile interface, by following the greater element spacing and font sizes required to create easy to use mobile web applications.

6.4 Difficulties Encountered

Some difficulties were encountered during the implementation, these were mostly minor bugs that resulted in slower performance, unexpected data, or connection issues. Issues were encountered during the apache web-server installation due to the specific configurations required to host the API and static website files correctly. This included the WSGI application requiring version configuration changes to run the correct language versions, solving this required viewing the apache logs to identify the cause. Another issue involved being unable to use the API endpoints that require user authentication, this was due to the apache configuration stripping the auth header before it reached the WSGI application, this was fixed by fixing the configuration. Changes were required to make the web-server host the website on base station’s network IP address instead of a local or specified IP, once the changes were completed the website and API were always accessible from the base station’s static IP address. Another problem caused by IP address was that the wifi manager and radio program required an additional endpoint to be created that wasn’t in the designs, this endpoint provides the user’s settings if the correct API key is specified, This changed resulted in two endpoints for the settings with one requiring JWT authentication and the other using the API key, as the independent programs cannot directly access the settings from the database or perform the JWT authentication by proving the user’s email and password.

Performance difficulties were encountered during the API implementation, these required fixing as the website interactions became delayed when deletion requests were made to the API due to the time taken for a response to be retrieved. The largest issue was the bad performance of the climate data and sensor deletion endpoints, as these would both delete very large batches of climate data objects. This was fixed by batching the deletion requests into a single call and ensuring that deletion cascaded to child objects within the object-relational-mapping instead of manually deleting the parents and children. Another difficulty involved bad performance with the historical climate data retrieval as all data points were retrieved between the two dates, this caused very delayed requests of over 5 to 10 seconds due to thousands of nested objects being retrieved. This was fixed by only retrieving every n rows of the climate data, this ensures that the same amount of data points was retrieved for all date ranges and that spacing was even between the selected intervals.

The database was originally designed to be a NoSQL database such as MongoDB (MongoDB Inc, 2020), it was not until the database implementation that it was discovered that there are no recent versions of MongoDB supported by the Raspberry Pi 3 A+. This resulted in going back and changing the database design and plans for the database interfacing with the API. Though this initially caused delays in the implementation, the later implementation of an SQLite database was likely a better design decision.

An issue was encountered once the time zone of the Raspberry Pi changed from GMT to BST, this caused inconsistencies with dates in the base station. The dates of the climate data recordings were 1 hour behind the dates used in other parts of the radio program. This was due to the open-source radio library providing packet dates that used the UTC time zone without any time zone offsets. This was fixed by using time zone conversion of the dates outputted by the radio library through the tzlocal library (Regebro, 2019), this demonstrates the unexpected issues of using open-source libraries where there is limited documentation and the source code has not been viewed.

6.5 Reflection on Implementation

In reflection, the implementation stage would have been more efficient if the Test Driven Development (TDD) methodology had been followed as this would have ensured that components successfully completed tests during development rather than waiting till the testing stage to create the tests. The Storybook library would have reduced the time spent refactoring poor code within the website into reusable and standardised components, as some of the implemented components are too specific and not fully reusable without modification.

7. Testing

This section describes the approaches taken to validate each stage of the project.

7.1 Requirements Validation

As the requirements were indirectly elicited, they were not verified or validated during the requirements stage. But the questionnaire completed during the evaluation stage provides some validation of the requirement importance for the system's features, with this being completed in sections 8.4 and 8.5.

7.2 Design Validation

The designs were not validated during the design stage, this meant that no user suggestions were received and thus the designs and the implemented system reflect only the author's design opinions. The evaluation questionnaire was completed after the implementation stage but did provide very favourable usability and suitability opinions on the main website designs. Ideally, a full usability evaluation should have been completed during the design stage to ensure that the designs were iterated and improved based on user opinions. The results of the UI usability section questionnaire have been evaluated in section 8.4.

7.3 Implementation Validation Approach

The approach for testing and debugging the system's code involved using automated test tools for unit and end-to-end tests across all the system's components. This means that the components tested were the sensor node, base station, API, database, and website. For each component, the tests were designed based on the features provided by the component and the requirements relevant to those features. These tests were separated into unit and end-to-end test sections that each had their own test plans. The test plans for all components include the test name, the description of what should occur for the test to be successful and whether the test was successful. The test plans and descriptions of the methods used to execute the test plans have been included below for each component, finally, an analysis of the test results was completed to understand any limitations of the methods used and the overall test results.

7.4 Sensor Node Testing

7.4.1 Approach

As the sensor node is a microcontroller with embedded code automated tests tools weren't possible. This meant that tests were completed with the base station active and a single sensor node being used to initiate the test plans. The test plans created include all features specified by the requirements that are provided by the sensor node, additionally some non-functional requirements that provided tests.

While most tests simply required setting up the initial state required by the test plan, performing an action then reviewing the result, some tests required different methods. This included the calculation of the battery life of the sensor node for each measurement interval. To do this a USB ammeter was used to measure the average amperes used by the node over an hour for each device state (sleep, normal, radio sending). Using the power consumption in each state and the time spent in each state the milliampere hour (mAh) was calculated using an equation. The battery capacity was then divided by the mAh giving the number of hours the device can be powered by the battery. This provided the battery life results shown in table 7**,** with the calculation spreadsheet in Appendix H.

Measurement interval	**Battery Life (hours)**	**Battery Life (days)**
5 Minutes	5330	222
10 Minutes	6236.17	259.84
30 Minutes	7618.83	317.45
60 Minutes	8065.92	336.08

Table 7. Sensor node battery life

Another complex test was the maximum radio range test, this was completed in an open area with no obstructions with a single node communicating with the base station. A program was created for the sensor node that uses the same radio configuration as the actual program to ensure accurate results. The program sent an initialisation packet every 5 seconds to the base station and flashed a built-in LED when an acknowledgement packet was received, indicating whether the communication was successful. The results for the test have been separated into different ranges from 1 metre to 100 metres as they represent different use cases, additionally, wall penetration tests were completed to ensure that there isn't significant range or communication lost due to obstructions that will occur when the system is used in a building.

7.4.2 End-to-end Testing

All tests apart from the 75 and 100-metre communication ranges tests were successful. The sensor node test plans are in Appendix I Table 1.

7.5 Base Station Testing

7.5.1 Approach

The base station's sensor node communication manager program was tested using the pytest library with automated tests to confirm that each function returns the correct result for various inputs. Automated tests were not completed on the wifi manager as the Wi-Fi networks could not be mocked within the pytest code, instead manual tests were used to confirm the functionality.

7.5.2 Unit Testing

All tests were successful, with the test plans for the sensor node communication and wifi manager available in Appendix I Table 2.

7.6 API Testing

7.6.1 Approach

The API was validated through automated unit tests using the pytest library, this was selected as it provided straightforward implementation into the flask endpoint structure while providing more features and better support than the flask specific testing libraries. The tests were created for each of the success and error states described in the API design documentation. These tests evaluated whether the correct responses were returned and the data in the database had been correctly modified according to endpoints functionality. While all documented error states were tested, the API generates dynamic errors for post requests that could not all be tested due to the potential number of tests for each endpoint. This means that each missing field, incorrect data type, or incorrect value range is tested, but combinations of each field are not tested as this results in too many tests that cannot be feasibility implemented.

7.6.2 Unit Testing

All tests were successful, though some required fixes to succeed these were mainly due to slight differences in the status messages provided by the endpoints when compared to the documentation. The test plans are available in Appendix I tables 3 to 22.

7.7 Web-server Testing

7.7.1 Approach

The web-server hosts the API and static website files, the correct hosting of the API is confirmed by the website's end-to-end tests in section 7.9 as the website consumes the API via the web-server. Thus, the only test for the web-server confirms that the static files can be served by the web-server when the files URL is accessed by HTTP requests.

7.7.2 Unit Testing

The web-serving tests were all successful, with the test plans available in Appendix I Table 21.

7.8 Database Testing

7.8.1 Validation Approach

The database was tested using pytest (Krekel, 2020) as the Flask-SQLAlchemy library was used to provide a python database interface. This meant that the correct creation, updating, and deletion of rows in the database were tested through the models and the methods provided by the interface.

7.8.2 Unit Testing

All tests were successful with the test plans available in Appendix I tables 23 to 27.

7.9 Website

7.9.1 Approach

The website was tested using automated unit tests to confirm the correct functionalities are performed by each Vue component and helper function, additionally, automated end-to-end tests confirmed that the website and API interact correctly.

The unit tests were implemented using Jest (Facebook, 2020) and vue-test-utils (vuejs, 2019), this allowed for the mocking of services and the shallow mounting of Vue components to ensure that each component was tested in isolation. These unit tests covered various component functionality including the correct displaying of data, correct handling of component props and global state, and the correct handling of user interactions. Alongside the unit tests, snapshot tests were created for each component test scenario, these confirmed that the component outputs the correct HTML.

The end-to-end tests were created using the Cypress library (Cypress, 2020). This uses a chromium browser to load the website, mimic user interactions and then confirm that various page elements such as the HTML and page title matched the specified values.

This method ensures that components work together correctly once combined and that the data provided by the API and database is successfully communicated. Initially, the nightwatch library was used to implement the first few tests but this required more complex test code to be written, thus the existing tests were converted to work with Cypress which replaced nightwatch. As Cypress only supports Chromium the website was not automatically tested with other browsers such as Firefox and Safari. Instead, manual testing using the Browserstack (https://www.browserstack.com) website was used to emulate the other browsers and confirm that the website worked correctly, this is a major limitation of the method used.

7.9.2 Unit Testing

All unit tests were successful though some required changes to the component's code to pass. The full test plans for each component are available in Appendix I tables 28 to

7.9.3 End-to-end tests

The end-to-end tests were successful and did not require any code changes to make the tests pass; this was due to the earlier completion of the API and website unit-tests. The test plans are in Appendix I tables 48 to 55.

7.10 Overall results

Overall, the tests designed based on the functional requirements were completely successful. With most of the tests representing non-functional requirements were successful apart from the sensor node communication range which was halfway to providing the required range.

This means that not all requirements will be successfully satisfied by the system as detailed in the evaluation section 8**.** Though the system’s code did require changes to ensure test success, this demonstrates that using TDD during to write the tests before code implementation would have been a better decision as rewriting code after the implementation was completed was avoidable and may have resulted in bad quality code. Additionally, there are several additional tests that were not designed or completed. Tests of the accuracy and range of the measurements provided by the BME280 sensor compared to trusted measurements conducted at the same time and location should have been completed. As this was conducted by some of the papers found during the literature research. The lack of automated tests in the website's compatibility with non Chromium-based browsers should have been completed to ensure identical visuals and interactions across modern browsers. This could have been completed using the Browserstack automated testing tool which interacts with the website in a similar manner to the end-to-end tests but with the advantage of being able to emulate popular browsers on desktop and mobile devices.

8. Evaluation

This section describes the system's evaluation against the requirements including the methods used and the outcomes.

8.1 Method

To evaluate the created system against the requirements, a questionnaire was used to elicit user opinions on the features, usability, and suitability of the system for its aim. The evaluation was then completed by comparing the requirements specification to the questionnaire and test outcomes. This enabled evaluation of the more subjective requirements such as the usability of the system, and the object requirements such as the features provided by the system. The limitations of this method are that it relies on the questionnaire being conducted well with no bias in the questions and answers available, and on the tests being completed on all of the aspect's components relevant to the requirements and that these tests are designed well to be strict on when they succeed. The complete analysis of the suitability and results of this method has been included in the overall evaluation section 8.6**.**

8.2 Questionnaire

8.2.1 Questionnaire Design

A questionnaire was created to acquire the opinions of potential users of the system. This questionnaire contained questions relating to the requirements of the system. The questionnaire design has been included in Appendix J.

The questions were grouped by their context, resulting in questions on functional requirements followed questions on non-functional requirements, then website UI usability and suitable, and finishing with the overall suitability of the system. The types of the answers used depended on the content and context of the question, this meant that yes/no questions were used to provide exact answers, these were used for some website usability questions, checkboxes were used to let the user provide any answers that apply for the feature importance questions, Likert-scale questions were used to understand the usability and suitability of the website UI, radio inputs were used for questions that had multiple fixed answers, finally raw number inputs were used in the cost and node questions to elicit the raw answers from the user instead of using number ranges. The website UI questions displayed a screenshot or short video of the UI before the question and answers were shown, this was used to not influence how the individual looks at the UI as showing the UI after the question text may have introduced bias.

The questionnaire was provided to various potential users, this included individuals that had no knowledge of weather systems, others that had heard but not used weather station products, and individuals that currently or previously used weather station products.

8.2.1 Questionnaire Results

To analyse the questionnaire results, the responses of each question have been analysed for trends and what these trends mean in the context of evaluation. The responses to the questionnaire have been included in Appendix K. The question analysis was then included in the evaluation of the requirement relevant to the question. To better understand the response trends, the responses were grouped based on the response to the first question. This resulted in three groups, group N for individuals that answered that they haven't heard of or used a weather station product, group K for individuals that had prior knowledge of at least one weather station product, and group U for those that have used a weather station product. In total 15 responses were given to the questionnaire, with 4 responses in group U, 4 in group K, and 7 in group N.

8.3 Evaluation Structure

Each requirement from the requirements specification has been listed along with the evaluation of whether it has been satisfied by the system. These are grouped into functional and non-functional sections. Finally, an overall evaluation has been created based on the evaluation results along with the limitations of this evaluation method.

8.4 Functional Requirements Evaluation

8.4.1 Indoor Sensor Node Requirement

This requirement is satisfied as a multiple purpose node was implemented for indoor and outdoor use as it uses a weather-proof case suitable for both purposes, this measures the surrounding climate based on the connected sensor modules.

8.4.2 Outdoor Sensor Requirement

This requirement is satisfied as the multiple purpose node in “4.5.1 Indoor Sensor Node” was intended for outdoor use as it uses an IP55 rated weatherproof case, this measures the surrounding climate based on the connected sensor modules.

8.4.3 Sensor Node Temperature Recording Requirement

This requirement is satisfied as the sensor nodes use the BME280 sensor to measure temperatures, with the tests confirming the functionality.

8.4.4 Sensor Node Temperature Units Requirement

This requirement is satisfied as the sensor nodes do record temperature measurements in Celsius with the data stored as Celsius in the database, with tests confirming the functionality. A setting is provided on the website that allows for the viewing of the temperature data in Fahrenheit.

8.4.5 Sensor Node Humidity Recording Requirement

This requirement is satisfied as the sensor nodes use the BME280 sensor to measure the atmospheric pressure of the surrounding climate, with tests confirming this functionality.

8.4.6 Sensor Node Recorded Data Requirement

This requirement is satisfied as though the sensor nodes do not directly record the time and date along with the climate data measurements, the base station instead adds the additional metadata including a datetime string to the climate data and sends it to the API for storage in the database. The website retrieves the climate data from the database via the API and displays the recent and historical data for users to view, with tests confirming this functionality.

8.4.7 Sensor Node Identifier Requirement

This requirement is satisfied as the sensor nodes are assigned a unique ID by the base station with this ID being generated by the API when a new node is added, with tests confirming this functionality. This ID is then used during all communication and storage of climate data to ensure the correct linking of the climate data to the sensor.

8.4.8 Additional Sensor Types Requirement

This requirement is satisfied as the nodes, base station, API, and database have been implemented to handle any type of climate data being collected. This is demonstrated by the BME280 providing pressure data which was not initially planned for. Though only one sensor module is used per node, which means that multiple sensor modules per node were not tested, though it should be completed in the future.

8.4.9 Multiple Sensor Nodes Requirement

This requirement is satisfied as the system was implemented to support up to 20 concurrent nodes and could be further modified to increase this limit, two nodes were implemented and tested to ensure this requirement was met.

8.4.10 Sensor Battery Power Requirement

This requirement is satisfied as the sensor nodes were implemented with batteries to provide power to the microcontroller and sensor modules.

8.4.11 Sensor Names Requirement

This requirement is satisfied as the sensor nodes are assigned user-defined names by the website which provides a renaming functionality. Initially, the API generates placeholder names with the sensor's ID, though the user can change this at any time, with tests confirming this functionality.

8.4.12 Climate Data Accessible by External Services Requirement

This requirement is satisfied as the API is accessible over the internet once port forwarding has been completed, the connectable service can then use JWT or API key authorisation with HTTP requests to consume the API endpoints. While the use of the API by external services wasn't tested or any external services implemented with the API, there shouldn't be any reason why this won't work.

8.4.13 Website Requirement

This requirement is satisfied as the website provided a dashboard with climate data, sensor management, and configuration options, with tests confirming this functionality. This is initially available over the local Wi-Fi network until port forwarding is completed this makes it available over the internet. But is reliant on the user understanding and researching how to port forward using their router.

8.4.14 Website Authorised Data Access Requirement

This requirement is satisfied as the API uses JWT authorisation for all endpoints meant for external use or that concern user and climate data, API key authorisation is used for the internal component communication though the endpoints that use this do not retrieve sensitive information. This ensures that data can only be accessed if the user has logged in with valid credentials, with tests confirming this functionality.

8.4.15 Website Desktop/Laptop Access Requirement

This requirement is satisfied as the website was accessed and tested with an automated testing tool on a Chromium browser with desktop screen sizes, additional manual testing was completed on Firefox and Safari to ensure the website is accessible on modern browsers. The website will work on any modern browser due to polyfills that ensure JavaScript backward compatibility though there may be some graphical differences due to the differences in browser implementations. The questionnaire results identified that most responders found the text easy to read on the desktop dashboard UI with all saying that the purpose and content of it were easy to understand demonstrating the usability of this UI.

8.4.16 Website Mobile Access Requirement

This requirement is satisfied as the website is accessible on mobile devices with the base station's IP address identically to the desktop version**.** A mobile interface was designed to suit the smaller screen size and large spacing used to suit fingers. This interface makes some interactions longer, such as the dashboard sensor selection, this is required to fit the smaller screen. The mobile interface had automated tests completed with the automated testing tool on a Chromium browser with mobile screen sizes, additional manual testing was completed on Firefox and Safari to ensure the website is accessible on modern browsers on real mobile devices. All questionnaire responders said that the purpose and content of the mobile dashboard page and settings page were easy to understand. Though the results were more mixed for mobile dashboard interactions and mobile navigation functionality. The questionnaire comments identified that a more modern mobile navigation method such as a bottom tab bar would be easier to use than the hamburger menu implemented.

8.4.17 Sensor Node List Requirement

This requirement is mostly satisfied as the website shows the user's nodes with their name, battery status, view button, deletion buttons, and recent temperature and humidity data, with tests confirming this functionality. As it does not provide a summary of all the most recent climate data, users instead must view each node's climate data to view recent climate data with other data types such as the atmospheric pressure.

8.4.18 Historical Climate Data Visualisation Requirement

This requirement is satisfied as the website displays a graph for each of the climate data types and the battery voltage over time, with tests confirming this functionality. This graph is based on a user-selected date and time range with the default being 1-day, pre-set date periods are also provided. A date range input is provided that allows users to select any date range. Most questionnaire responders said that the data selection functionality was very suitable though a minority said it mostly or partially suitable. This may have been due to the mix of bespoke pre-set period buttons and the date range input from an open-source library that is inconsistent in design though some configuration was used, this was clearly not enough to ensure consistency in user interaction.

8.4.19 Recent Climate Data Requirement

This requirement is satisfied as the website displays the temperature and humidity for each node in the list, with tests confirming this functionality. When a node is selected a box displays the most recent values for each type along with the daily highs and lows, and the trend compared to the previous value. The high, lows and trends were not required but provide additional analysis that helps the user understand the data being displayed rather than the raw measured values.

8.4.20 Website Temperature Units Requirement

This requirement is satisfied as the website has a configuration to choose whether to display the recorded temperature in Celsius or Fahrenheit, this only affects the displaying on the website and not stored data, with tests confirming this functionality.

8.4.21 Climate Data Deletion Requirement

This requirement is mostly satisfied as the website provides a button in the sensor node list to delete the climate data for each node individually, with tests confirming this functionality. But the questionnaire results identified that users were split on the usability of this feature as only half answered that it was very suitable with other selected mostly and partially suitable demonstrating that future work is required in improving the usability of the UI used for this feature. Additionally, 3 responders identified that a deletion confirmation functionality would help fix the usability problems.

8.4.22 Sensor Deletion Requirement

This requirement is mostly satisfied as the website provides a button in the sensor node list to delete each node individually, with tests confirming this functionality. But similarly, to "4.5.21 Climate Data Deletion" the questionnaire results identified an identical split on the feature usability with all responders providing the same answers to the other question. The deletion confirmation functionality would also help fix the usability problems of this functionality.

8.4.23 Website Sensor Naming Requirement

This requirement is satisfied as the website provides a togglable text box for each sensor in the node list that allows for the renaming of sensors, with tests confirming this functionality. The questionnaire identified that the vast majority thought this method of implementation of the feature was suitable, though 26.7% responded mostly suitable, thus there is additional work required to improve the UI of this feature. One responder commented that better distinguishing between the editable node name and the node ID displayed next to it would make it clearer to understand.

8.4.24 Login Requirement

This requirement is satisfied as the website requires users to provide the valid email and password that matches the stored credentials to be able to access any pages other than the register, login, and home pages. A login page is provided that allows users to login via a form with email and password inputs, with tests confirming this functionality. The questionnaire results identified that all responders found the login page suitable for the login functionality.

8.4.25 Register Requirement

This requirement is satisfied as the website requires that the user registers an account before they can access the rest of the website other than the login and home pages. A register page is provided that takes in the email, password and confirm password via a form which registers the user once completed, with tests confirming this functionality. The questionnaire results identified that all responders found the register page suitable for the register functionality.

8.4.26 Forgot Password Requirement

This requirement is satisfied as the website provides a page that allows users to reset their password using a reset token provided on registration or when already logged on and the new password, with tests confirming this functionality in both scenarios. This was chosen as the independent system design meant than an email account for sending the reset link could not be safely kept in the base station. Thus, the reset token method was used, this is overall less effective, but a necessary sacrifice given the security and privacy requirements. The questionnaire results identified that all responders found the reset password page suitable for the forgot password functionality.

8.4.27 Add Sensor Requirement

This requirement is not satisfied as the website does not handle the additions; this is instead completed when the sensor node sends the initialisation request to the base station. This was chosen as a website interface for the addition would have required knowing the node ID before it had been generated requiring additional functionality to link the new sensor node to the node created on the website.

8.4.28 Sensor Status Requirement
This requirement is satisfied as the website displays the battery voltage in a graph over time, with tests confirming this functionality. Additionally, the most recent status based on voltage ranges is shown for each sensor in the node list. Though one comment on the questionnaire said that the battery level should be displayed in a graphical format such as battery indicator icons.

8.5 Non-functional Requirements Evaluation

8.5.1 Recent Climate Data Requirement
This requirement is satisfied as the climate data processing after measurements is minimal and thus is done in under 5 seconds depending on if radio communication retries are required. The website dashboard retrieves the most recent climate data from the API every minute, this ensures that the data is always available after one minute.

8.5.2 Temperature Climate Data Accuracy Requirement
This requirement is not satisfied as the BME280 records to ±1.0°C temperature accuracy based on its hardware specifications rather than the ±0.5°C accuracy required. This was deemed a necessary sacrifice for affordability in each node as the sensors providing ±0.5°C or lower were more expensive and did not provide the humidity measurements as required. Though temperature accuracy tests were not completed to ensure that the specifications provided were correct.

8.5.3 Temperature Climate Data Range Requirement
This requirement is satisfied as the BME280 provides temperature measurements between -40°C to 85°C based on the hardware specifications provided, this is slightly better than the range of -40°C to 80°C required. Though temperature range tests were not completed to ensure that the specifications provided were correct.

8.5.4 Humidity Climate Data Accuracy Requirement
This requirement is satisfied as the BME280 provides humidity measurements in the range of ±3% based on the hardware specifications, this is more accurate than the ±5% accuracy required. Similarly, to the temperature data, accuracy tests were not completed to ensure the specifications were correct.

8.5.5 Humidity Climate Data Range Requirement
This requirement is satisfied as the BME280 provides humidity measurements in the range of 0% to 100% based on the hardware specifications provided, this is slightly identical to the range of 0% to 100% required. Though humidity range tests were not completed to ensure that the specifications provided were correct.

8.5.6 Concurrent Climate Sensor Nodes Requirement
This requirement is satisfied as the system can handle between 1 to 20 nodes per individual installation which is more than the required 1 to 10 nodes. Though only two nodes were implemented due to cost limitations. The number of nodes is only currently constrained by the amount of time given per node in their time periods provided by the radio protocol. Thus, the concurrent nodes limit can be increased if the time periods are shortened though this may result in a higher chance of radio packet collisions.

8.5.7 Climate Data Storage Duration Requirement

This requirement is satisfied as the database can store permanently depending on the storage space available, though the API deletes any climate data older than 6 months each day. This meets the 6 months of storage required and provides the potential for the limit to be increased.

8.5.8 Sensor Recording Frequency Requirement

This requirement is satisfied as the sensor nodes can record measurements at 5, 10, 30, or 60-minute intervals as selected by the user. This meets the minimum 5-minute intervals required.

8.5.9 Sensor Recording Frequency Consistency Requirement

This requirement is mostly satisfied as the sensor nodes will record within the 20 seconds following their assigned time period due to the node sleeping not being time perfect. This is not completely consistent with equal lengths of times between measurements as required. But the user should not notice the data being recorded seconds after expected. Though the message used does ensure that the nodes are synchronised with a new sleep duration after each measurement to stop any long-term drifts in measurement frequency consistency.

8.5.10 Sensor Battery Life Requirement

This requirement is satisfied as the nodes are estimated to last up to 222 days on battery for the 5-minute interval with the 60-minute interval providing up to 336 days of battery life. This is considerably better than the 2 months of battery life required. Though these numbers were calculated based on the node power consumption rather than being tested months as this was not possible in the time frame available.

8.5.11 System Affordability Requirement

This requirement is not satisfied as the final system with two sensor nodes and the base station costing £161.55 to be implemented as of December 2019. This is considerably more than the £100 cost required. Though implementation of a single sensor node and a base station costs only £102.62 which is closer but does not reflect the modularity aim which is the key objective of the project.

8.5.12 Installation Usability Requirement

This requirement is mostly satisfied as only one technical step is required to install the system rather than the no technical knowledge required for installation. This involves port-forwarding the base station's IP address which will require different levels of technical knowledge depending on the user interface provided by the user's Wi-Fi router. Though the results of the questionnaire indicated that only 40% of the responders said that this was a suitable step for installation this was mainly due to the majority of group U answering that it was suitable with the majority of the other groups answering that it wasn’t.

8.5.13 Maintenance Usability Requirement

This requirement is satisfied as the system does not require technical knowledge to use after installation during normal system use. As all configuration is provided by the website and all programs are automatically run once powered on. But any bug fixes and new features cannot be added to the website without having the user add the updated files onto the base station's PI via an SSH connection, this concern for future improvements but wouldn’t be required for the current normal system use.

8.5.14 Sensor Node Distance Requirement

This requirement is not satisfied as the maximum range tests identified that the max communication distance is 56.5 metres instead of the 100 metres required. This was completed using the current hardware and configuration in an open area with no obstructions. This may be improved in future work through additional radio antenna hardware or different radio configurations. Though the results of the questionnaire resulted in an average of 44 metres when users indicated what they thought was a suitable max communication range, thus the current range may be suitable for many users.

8.5.15 User Data Security Requirement

This requirement is satisfied as all data is stored locally in the base station, with this data only be accessible by directly accessing the Raspberry Pi or successfully completing the JWT authentication used by the API endpoints that concern user and climate data. The JWT authentication requires the valid password and email to be input, with the password being hashed and salted in the database so that if a malicious actor accessed the database they would not be able to retrieve the raw text password. Thus the user data is secure from access by individuals other than the user, with the only concern being that a malicious individual with access to the user’s browser which has logged in, this would give them access to the JWT access token required for the API authentication though this is unlikely to occur.

8.6 Overall evaluation

The evaluation identified that the system’s functionality was successfully implemented according to the functional requirement specification, though some changes how they were implemented did occur without affecting the functionalities purpose as evaluated by the tests. The majority of non-functional requirements were satisfied apart from the maximum node range, affordability, and temperature accuracy though these are important requirements the results of the questionnaire indicated that most potential users would be satisfied by the node range and accuracy provided by the system. But the system’s cost and node costs were still higher than the specification and questionnaire results, thus this is a major focus for future work.

Additionally, there were issues in which the questionnaire was designed that may have affected the usefulness of the results for the evaluation. The questionnaire only received 4 responses from users of existing weather station products, additional responses should have been retrieved by interacting with weather station communities. This should have provided a better understanding of their priorities and issues with existing products and how this affects their opinions on the system. The requirements used in the evaluation were not validated during the requirements stage with only some of the questions providing importance and value validation of the requirements. Additionally, responders had not been given hands-on experience with the hardware and website, with their only experience of the system coming from the screenshots and videos in the questionnaire. Hands-on user evaluation sessions should have been completed, as the website is accessible over the internet and the credentials could have been provided. This would have given better results as users may not be able to give full opinions on UI usability until they have directly interacted with the website. After the questionnaire, some respondents noted that additional background knowledge should have been provided as they did not know the features of existing products such as the costs. A product overview could have been provided but this may have influenced their answers once they found out the system was designed to improve upon the limitations of existing products.

9. Conclusion

9.1 Conclusion

Overall, the project was mostly successful based upon the evaluation against requirements, but additional work is required for improving the system and evaluating its use within integration with smart home systems. While the system has the required features and most of the non-functional requirements, it lacks the polishing of features and currently acts more as a prototype of a potential system. Based on the aim described in the introduction it is a more modular and flexible system than the alternative weather station products with the same core functionality though the affordability aim was not met due to higher than expected costs. The future recommendations have been analysed based upon each of the project's stages.

The requirements stage did provide many detailed requirements, but they were not elicited directly from users. Instead, they should have been more effective using more direct requirements elicitation from potential end-users of the system. This could have included interviews with users of existing weather station products to understand their needs from a system, and from non-users to understand why they have not yet used a weather station product. This would have likely resulted in more useful requirements that directly represent a user's requirements than those indirectly elicited from background research on the market.

The design stage provided good results across all aspects of the system, but it should have had more user evaluation mainly focused on the usability of the website UI designs and the use of connectable services so that the smart home use case could be better fulfilled by the system. Instead, the website design was purely based on the features offered by existing systems and the concept of dashboard UI layouts and styles. While the questionnaire responses showed that users did find the website UI easy to understand, they had limited interactions with the actual website though letting each potential experience the system for long periods would not have been practical regardless of the better usability improvements it would have provided. By understanding what users wanted from connectable services, the API would have been better designed to integrate with the most common home automation services used by users. The design stage also suffered from a large error due to the initial selection of a MongoDB database which was not supported by the Raspberry Pi hardware until the implementation stage, this should have been researched before it was selected for the database design and wouldn't have resulted in any later difficulties**.**

The testing stage identified the success of the system’s implementation but did identify that the radio range was lower than expected, this is important to ensure the project's purpose and could be improved through better hardware or radio configuration. Additionally, unit tests should have been written during the implementation through TDD rather than being completed after the system’s implementation. This would have improved the code structure and saved time caused by fixing code responsible for failing tests.

The evaluation stage demonstrated that the functionality was implemented correctly, but that the affordability and node range requirements were not satisfied. While these were both important for the purpose of improving upon existing weather station devices, they can be implemented in future work with lower-cost hardware and better radio hardware. It also identified that the accuracy of the measurements was not tested, thus the current system may not be providing accurate measurements, this will need to be completed in future work.

Additionally, there are some flaws identified from extended personal use of the system. These include no radio protocol support for multiple linked climate data packets when a single packet cannot support the amount of measured data collected. This means that the current system can only support a certain amount of sensor modules at the same time. Updating the sensor node and base station with new code requires manual updating, this is unsuitable for general consumer use. Instead, it should be done automatically from a remote code repository.

Based upon the issues with each step of the project the recommendations for system improvements and research have been included in the future work section 9.3, with personal recommendations in section 9.2.

9.2 Reflection

To improve future projects several personal recommendations have been made based on the successes and failures of the project.

9.2.1 More Flexible Project Plan

Use a more flexible development life cycle that limits the number of overlapping tasks so that delayed tasks have less of an impact on later tasks.

9.2.2 Feature Prioritisation

Prioritization based on primary and secondary features, as the whole system with primary features could have been created first then with the secondary features added afterwards. This would minimize the impacts of missing deadlines on the system’s core functionality ensuring its main aims are satisfied.

9.2.3 Technology Compatibility

Ensure that full research is completed on the compatibility of different technologies when required. This is due to the issues in the implementation where there was no MongoDB support for Raspbian as planned for in the initial design. This resulted in delays due to searching for another noSQL database as an alternative. When none were found, it required changing the design to an SQL database and then continuing with the implementation, resulting in lost development time.

9.2.4 Methodology Selection

More time should be used in selecting appropriate methodologies for each stage of the project. This would likely result in the more efficient completion of each stage. As the steps required by the methodology would likely improve results if followed correctly.

9.3 Future Work

There is plenty of future work to be completed with this system due to its flexibility and potential expandability.

9.3.1 Additional Sensor Modules

Implement additional sensor modules such as lighting, rainfall, and wind detectors; so that the system can be evaluated with multiple sensor modules per node.

9.3.2 Sensor Node Communication Range

The sensor nodes should have their radio range improved through antenna hardware or using radio modules with greater range such as LoRa, this could help in scenarios covering large geographic areas such as farms. This is especially important as while one of the papers researched did use LoRa in a weather station system it did not include testing of the maximum range. Improved communication range could also be explored through node mesh networks and the repeating of packets towards the base station.

9.3.3 Sensor Module Interface Standardisation

There is future research into the standardisation of sensor interfaces which would remove the need for custom sensor interface code within the sensor nodes.

9.3.4 Base Station Code Updating

The base station needs better code updating functionality that doesn’t require manual updating files, instead, it should use the source files on the current master branch of the GitHub repository so that the base station programs always remain up to date automatically, this is especially important if multiple implementations of the system are used.

9.3.5 Improved Radio Protocol

The radio protocol needs improvements to support multiple linked climate data packets. The improvement would facilitate nodes sending multiple packets per climate recording and combine them in the base station when the climate data is larger than a single packet.

9.3.6 Website Secondary Features

The website should be expanded with additional secondary features such as the searching and organisation of sensors, more mobile-specific changes to improve usability and performance. The climate data analysis could be improved with greater trends being generated such as means, averages, and comparisons of historical data such as comparing different weeks and months of climate data.

9.3.7 Sensor Accuracy Testing

Additionally, tests should be completed on the measurement accuracy of the system's sensors when compared to local trusted sources of climate data, so that any measurement biases can be understood.

9.3.8 Alternative Low-cost Hardware

Research and experimentation should be completed on what lower-cost hardware could be used to make the system more affordable while remaining suitable for the rest of the requirements.

10. References

AcuRite. (2016). *AcuRite 01008M Atlas Weather Station.* Retrieved from https://www.acurite.com/shop-all/weather-instruments/weather-stations/atlas-weather-station-with-access-remote-monitoring.html

Adafruit. (2020). Adafruit BME280 Library [Computer software]. Retrieved from https://github.com/adafruit/Adafruit_BME280_Library

Adafruit. (2020). Adafruit Unified Sensor Driver [Computer software]. Retrieved from https://github.com/adafruit/Adafruit_Sensor

Adityawarman, Y., & Matondang, J. (2018). *Development of Micro Weather Station Based on Long Range Radio Using Automatic Packet Reporting System Protocol*. 2018 International Conference on Information Technology Systems and Innovation, ICITSI 2018 - Proceedings, 221–224. https://doi.org/10.1109/ICITSI.2018.8696081

Alex Grönholm. A. G. (2019). Advanced Python Scheduler [Computer software]. (Version 3.6.3). Retrieved from https://apscheduler.readthedocs.io/en/stable/

Ambient Weather. (2016). *Ambient Weather WS-2902B Smart Weather Station.* Retrieved from https://www.ambientweather.com/amws2902.html

Arduino Software. (2020). Arduino IDE [Computer software]. (Version 1.8.10). Retrieved from https://www.arduino.cc/en/main/software

Armin Ronacher. A. N. (2019). Flask [Computer software]. (Version 1.1.1). Retrieved from https://flask.palletsprojects.com/en/1.1.x/

Bell, S., Cornford, D. and Bastin, L. (2013), *The state of automated amateur weather observations. Weather,* 68: 36-41. https://doi.org/10.1002/wea.1980

BloomSky. (2016). *SKY2 Weather Camera Station.* Retrieved from http://shop.bloomsky.com/products-list/sky2

Bresser. (2019). *BRESSER WIFI professional weather station*. Retrieved from https://www.bresseruk.com/weather-station/bresser-wifi-professional-weather-station.html

Brush, A. J. B., Lee, B., Mahajan, R., Agarwal, S., Saroiu, S., & Dixon, C. (2011). *Home Automation in the Wild: Challenges and Opportunities.* https://doi.org/10.1145/1978942.1979249

Business Insider. (2020). Smart Farming in 2020: *How IoT sensors are creating a more efficient precision agriculture industry*. Retrieved from https://www.businessinsider.com/smart-farming-iot-agriculture?r=US&IR=T

Citizen Weather Observer Program. (n.d.). *Citizen Weather Observer Program*. Retrieved from http://www.wxqa.com/

Cory Dolphin. C. D. (2019). Flask-CORS [Computer software]. (Version 3.0.8). Retrieved https://github.com/corydolphin/flask-cors

Cypress. (2020). Cypress [Computer software]. (Version 4.0.2) Retrieved from https://github.com/cypress-io/cypress

Davis Instruments. (2009). *Vantage Vue Wireless Weather Station.* Retrieved from https://www.davisinstruments.com/product/vantage-vue-wireless-weather-station/

De Silva, L. C., Morikawa, C., & Petra, I. M. (2012). *State of the art of smart homes.* Engineering Applications of Artificial Intelligence, 25(7), 1313–1321. https://doi.org/10.1016/j.engappai.2012.05.002

Dri-Box. (2011). Dri-Box FL-1859-200 IP55 Weatherproof Box. Retrieved from http://dri-box.com/

Eli Collins. E. C. (2019). Passlib [Computer software]. (Version 1.7.2). Retrieved from https://passlib.readthedocs.io/en/stable/

Evan You. E. Y. (2019). Vue [Computer software]. (Version 2.6.10). Retrieved from https://github.com/vuejs/vue

Facebook. (2020). Jest [Computer software]. (Version 25.1.0) Retrieved from https://github.com/facebook/jest

Gomez, C., & Paradells, J. (2010). *Wireless home automation networks: A survey of architectures and technologies*. IEEE Communications Magazine, 48(6), 92–101. https://doi.org/10.1109/MCOM.2010.5473869

Gomez, C., & Paradells, J. (2010). *Wireless home automation networks: A survey of architectures and technologie*s. IEEE Communications Magazine, 48(6), 92–101. https://doi.org/10.1109/MCOM.2010.5473869

Greenough, J. (2016). *SMART HOME MARKET: Adoption forecasts, top products, cost & fragmentation problems*. Retrieved from https://www.businessinsider.com/the-us-smart-home-market-report-adoption-forecasts-top-products-and-the-cost-and-fragmentation-problems-that-could-hinder-growth-2015-9

Holger Krekel. H. K. (2020). pytest [Computer software]. (Version 5.3.5) Retrieved from https://github.com/pytest-dev/pytest

Internet Engineering Task Force. (2014a). *Terminology for Constrained-Node Networks*. Retrieved from https://tools.ietf.org/html/rfc7228

Internet Engineering Task Force. (2019). *Internet of Things (IoT) Security: State of the Art and Challenges*. Retrieved from https://tools.ietf.org/html/rfc8576

Jakub Juszczak. J. J. (2019). vue-chartjs [Computer software]. (Version 3.5.0). Retrieved from https://github.com/apertureless/vue-chartjs

JetBrains. (2019). PyCharm Community Edition [Computer software]. (Version 2019.3). Retrieved from https://www.jetbrains.com/pycharm/

Kanagaraj, E., Kamarudin, L. M., Zakaria, A., Gunasagaran, R., & Shakaff, A. Y. M. (2015). *Cloud-based remote environmental monitoring system with distributed WSN weather stations*. 2015 IEEE SENSORS - Proceedings, 1–4. https://doi.org/10.1109/ICSENS.2015.7370449

Kapoor, P., & Barbhuiya, F. A. (2019). *Cloud Based Weather Station using IoT Devices*. IEEE Region 10 Annual International Conference, Proceedings/TENCON, 2019-October, 2357–2362. https://doi.org/10.1109/TENCON.2019.8929528

Kusriyanto, M., & Putra, A. A. (2019). *Weather Station Design Using IoT Platform Based On Arduino Mega*. ISESD 2018 - International Symposium on Electronics and Smart Devices: Smart Devices for Big Data Analytic and Machine Learning, 1–4. https://doi.org/10.1109/ISESD.2018.8605456

Landon Gilbert-Bland. L. G. (2019). Flask-JWT-Extended [Computer software]. (Version 3.24.1). Retrieved from https://github.com/vimalloc/flask-jwt-extended

Lennart Regebro. L. R. (2019). tzlocal [Computer software]. (Version 2.0.0) Retrieved from https://github.com/regebro/tzlocal

LowPowerLab. (2013). *RFM69 library and Moteino R3*. Retrieved from https://lowpowerlab.com/2013/06/20/rfm69-library/

LowPowerLab. (2013). *LowPowerLab RFM69 packet structure.* Retrieved from https://lowpowerlab.com/wp-content/uploads/2013/06/RFM69_library_packet_structure.png

LowPowerLab. (2020). Low-Power [Computer software]. (Version 1.7). Retrieved from https://github.com/lowpowerlab/lowpower

LowPowerLab. (2020). RFM69 Library [Computer software]. Retrieved from https://github.com/LowPowerLab/RFM69

Michael Bayer. M. B. (2019). flask-sqlalchemy [Computer software]. (Version 2.4.1). Retrieved from https://www.sqlalchemy.org/

MongoDB Inc. (2020). MongoDB [Computer software]. (Version 4.2). Retrieved from https://www.mongodb.com/

Netatmo. (2012). *Smart Home Weather Station.* Retrieved from https://www.netatmo.com/en-gb/weather/weatherstation

Nathan Reyes. N. R. (2020). v-calendar [Computer software]. (Version 1.0.1). Retrieved from https://github.com/nathanreyes/v-calendar npm, Inc. (2020). npm [Computer software]. (Version 6.13.4). Retrieved from https://www.npmjs.com/

Paul Ganssle. P. G. (2019). dateutil [Computer software]. (Version 2.8.1). Retrieved from https://github.com/dateutil/dateutil/

Postman Inc. (2020). Postman [Computer software]. (Version 7.22.1). Retrieved from https://www.postman.com/

Python Packaging Authority. (2020). pip [Computer software]. (Version 20.0.2). Retrieved from https://pip.pypa.io/en/stable/

Quoc-Anh Nguyen. Q. N. (2019). Vue-axios [Computer software]. (Version 2.1.5). Retrieved from https://github.com/imcvampire/vue-axios

Rahman, M., Hossen, M., & Rahama, T. (2017). *Raspberry Pi as Sensor Node and Hardware of the Internet of Things (Iot) for Smart Home*. International Journal of Innovative Research in Electronics and Communications, 4(1), 12–19. https://doi.org/10.20431/2349-4050.0401003

Saini, H., Thakur, A., Ahuja, S., Sabharwal, N., & Kumar, N. (2016). *Arduino based automatic wireless weather station with remote graphical application and alerts.* 3rd International Conference on Signal Processing and Integrated Networks, SPIN 2016, 605–609. https://doi.org/10.1109/SPIN.2016.7566768

Sasha Koss. S. K. (2019). Date-fns [Computer software]. (Version 2.9.0). Retrieved from https://github.com/date-fns/date-fns

Savic, T., & Radonjic, M. (2016). *One approach to weather station design based on Raspberry Pi platform*. 2015 23rd Telecommunications Forum, TELFOR 2015, 623–626. https://doi.org/10.1109/TELFOR.2015.7377544

Shakeeb Sadikeen. S. S. (2019). vue-toasted [Computer software]. (Version 1.1.27). Retrieved from https://github.com/shakee93/vue-toasted

Statista. (2019). *Smart Home.* Retrieved from https://www.statista.com/outlook/279/100/smart-home/worldwide#market-age

Steven Loria. S. L. (2019). marshmallow [Computer software]. (Version 3.3.0). Retrieved from https://github.com/marshmallow-code/marshmallow

Tenzin, S., Siyang, S., Pobkrut, T., & Kerdcharoen, T. (2017). *Low cost weather station for climate-smart agriculture*. 2017 9th International Conference on Knowledge and Smart Technology: Crunching Information of Everything, KST 2017, 172–177. https://doi.org/10.1109/KST.2017.7886085

Therpin. (2017). Therpin DIY Waterproof Electronic ABS Plastic Project Junction Box Enclosure 200mm x 120mm x 75mm. Retrieved from https://www.amazon.co.uk/Therpin-Waterproof-Electronic-Junction-Enclosure/dp/B071GRDPZG

Thibault Friedrich. T. F. (2019). node-wifi [Computer software]. (Version 2.0.12). Retrieved from https://github.com/friedrith/node-wifi

Tidelift. (2019). Autoprefixer [Computer software]. (Version 9.7.4). Retrieved from https://github.com/postcss/autoprefixer

vuejs. (2019). Vue CLI [Computer software]. (Version 4.1.2) Retrieved from https://github.com/vuejs/vue-cli

vuejs. (2019). Vue-test-utils [Computer software]. (Version 1.0.0-beta.29). Retrieved from https://github.com/vuejs/vue-test-utils

vuejs. (2019). Vuex [Computer software]. (Version 3.1.2). Retrieved from https://github.com/vuejs/vuex

vuejs. (2020). vue-router [Computer software]. (Version 3.1.5). Retrieved from https://github.com/vuejs/vue-router

Vujović, V., & Maksimović, M. (2015). *Raspberry Pi as a Sensor Web node for home automation. Computers and Electrical Engineering*, 44, 153–171. https://doi.org/10.1016/j.compeleceng.2015.01.019

WeatherFlow. (2016). *WeatherFlow Smart Home Weather Station.* Retrieved from https://weatherflow.com/tempest-weather-system/

webpack. (2019). webpack [Computer software]. Retrieved from https://github.com/webpack/webpack

Williams, M., Cornford, D., Bastin, L., Jones, R., & Parker, S. (2011). *Automatic processing, quality assurance and serving of real-time weather data.* Computers and Geosciences, 37(3), 353–362. https://doi.org/10.1016/j.cageo.2010.05.010

World Meteorological Organization. (2018). *Guide to Meteorological Instruments and Methods of Observation* (Issue 8). Retrieved from https://www.weather.gov/media/epz/mesonet/CWOP-WMO8.pdf

11. Appendices

11.1 Appendix A: Project Initiation Document

UNIVERSITY OF
PORTSMOUTH

School of Computing Final Year Project

Daniel Hearn

PJE40

Project Initiation Document

Networked Climate Monitor

Project Initiation Document

1. Basic details

Student name:	Daniel Hearn
Draft project title:	Networked Climate Monitor
Course:	BSC (Hons) Computer Science
Client organisation:	N/A
Client contact name:	N/A
Project supervisor:	Rinat Khusainov

2. Degree suitability

The project is suitable for a Computer Science final year project as it involves producing an IT system through software design and development, web development, networking design, database design and development, and Internet-of-Things device development.

3. Outline of the project environment and problem to be solved

The problem that this project aims to solve is that it is difficult to get accurate and real-time information about indoor and outdoor climates in areas that you live in or own such as your house or office. This climate information includes temperature, humidity, air quality, rainfall, and wind speed. While some commercial products exist that attempt to solve the problem, they are expensive and often limited by the number of separate climate sensors they can track and the types of information that these sensors can track. Due to this problem and the limited existing solutions, a cheaper and more modular solution is required to solve the problem.

4. Project aim and objectives

The project aims to solve the problem by developing a system that can record the local climate around a sensor. The recorded climate data is then accessible via a website that enables users to see real-time and historical data from the climate sensor. This aim will be achieved by researching, designing, implementing and testing a system to solve the problem. This system will contain at least one Internet-of-Things device with a temperature sensor, a database to store the sensor data, and a web-based user interface to remotely view temperatures collected by the sensor over the internet. This system will be designed to be modular so that additional types of data or the climate of a different area could be collected by additional sensors.

Objectives in meeting the project's aim:

1. Obtain, analyse and specify the requirements of the system.
2. Research available climate monitoring solutions to understand existing commercial and research projects that design or implement a system that solves a similar problem to the one proposed.
3. Research available Internet-of-Things technology and compare them to find the most suitable technology to be used in the project while considering the technology used in similar systems.

4. Design a system that will meet the aim of the project by solving the problem.
5. Implement the system based on the design.
6. Test the system to ensure it fulfils the project’s aim and correctly solves the problem.
7. Evaluate the test results to establish the project’s suitability as a viable solution to the problem.

The initial requirements below have been extracted from analysis of the problem and the initial research on the features of existing commercial systems within the problem’s environment. These requirements will be expanded further during the full requirements analysis and specification, along with the relative importance of the individual requirements.

Functional Requirements:

- The system should have a device with a temperature sensor that can measure the temperature in the surrounding environment.
- The system should have a device with a humidity sensor that can measure the relative humidity in the surrounding environment.
- The sensors devices should function in indoor and outdoor environments.
- The device with the sensors should have communication capabilities to send the sensor data to the database via the internet.
- The system should have a database to store the sensor data.
- The system should have an API to handle setting and retrieving the sensor data to and from the database over the internet.
- The system should have a web-based interface accessed over the internet allowing for the viewing of the data collected by the temperature and humidity sensors.
- The system should have the capability to implement additional sensors to the project that collect additional data.
- The sensor devices should be battery-powered, with the capability to also be powered through a mains connection.

Non-Functional Requirements:

- Temperature sensor data should be recorded within ±0.5°C accuracy.
- The temperature sensor should be able to record within a range of -40°C to 80°C.
- Humidity sensor data should be recorded within ±0.5°C accuracy.
- The humidity sensor should be able to record within a range of -40°C to 80°C.
- The system should be able to handle between 1 to 5 climate sensor devices within or around a single location or house that are all recording data for the same user.
- The system should be able to handle multiple climate sensor devices each in different geographic locations that are all recording data for the same user.
- The system should be able to store a user’s climate data for at least 6 months from the recording of that data.
- The frequency that the sensors record at should be as often every 5 minutes.
- The sensor devices should have a battery life of at least 2 months before having to have their batteries replaced or recharged.
- Climate sensor data on the web-based interface should be the most recent data within one minute of the data being recorded on the sensor device.
- The system should be more affordable than existing solutions with the final system costing under £100 to implement.
- Once the system has been built it should be easy to implement in a different location without technical changes or technical knowledge of the system.

- The system should be easy to maintain once the system has been built and implemented, without technical changes to the system other than replacing/recharging the batteries used in the sensor device.
- The user data shall be kept secure from malicious or accidental access by individuals other than the user that the data belongs to.

5. Project deliverables

Information system artefacts to be developed:

- Internet-of-Things device with climate sensing capabilities.
- Software for the climate sensor device.
- Server software with an API for handling climate data and web application serving capabilities.
- Database for sensor data storage.
- Web application for viewing the climate sensor data.

Documents to be produced:

- Research results.
- Requirement specifications.
- Design specifications.
- Test strategies and results.
- Final report.

6. Project constraints

Possible constraints on the project:

- Acquiring the hardware required to build the sensor device as they will have financial constraints and will be subject to availability from retailers.
- Testing the solution within a realistic environment that accurately reflects the problem so that the solution can be accurately evaluated.

7. Project approach

The project approach will use the waterfall methodology so that the project’s steps are sequentially approached, this provides significant time to be allocated at each step. This approach will allow for detailed planning at the start of each step so that risks and issues can be identified before work is undertaken on each step. But this approach is inflexible and could result in later milestones being delayed if a task is not completed on time due to unexpected issues. These issues should be mitigated by the plans to review and mitigate risks, as well as the backup plans if the risks are not successfully mitigated.

Major steps of the approach:

1. Requirements elicitation and analysis.
2. Research into existing systems.
3. System design.
4. System implementation.
5. System testing.
6. System evaluation.
7. Writing of the final report.

Starting background research:

- Existing commercial and research systems for climate monitoring.
    - How they handle communication to the Internet.
    - How they handle communication between climate sensing devices.

    - How they handle powering the climate sensor devices.
    - What data they can collect and if they require additional sensors for additional types of data collection.
    - The cost of implementing these systems.
    - Whether these systems work indoors and/or outdoors.

Required skills to produce the system:

- Web-app development.
- Web-server development.
- API development.
- Database design and development.
- Internet-of-Things device development.
- Network design.

To acquire the required skills that I don’t know, I will need to learn more about implementing Internet-of-Things device communication and how to install and configure specific hardware with Internet-of-Things devices.

8. Facilities and resources

The project will require a computer/laptop to develop the system’s software and database. This computer, another computer or a cloud-based service will also be required to host the database and server software.

The project will need resources in the form of the technology required to build an ideal temperature sensor device based on the results of Internet-of-Things technology research. This technology includes the main device (e.g Raspberry Pi, Arduino, microcontrollers) and the electronics hardware (e.g temperature sensors, wiring). Acquiring this technology will be subject to financial and availability constraints as they will need to be bought from retailers.

9. Log of risks

Possible project risks:

- Running out of time to produce the project due to the complexity of producing a system comprising of multiple parts, where each part is required to deliver an effective system that will solve the problem.
- Difficulties in obtaining the technology that is required in the building of the temperature sensing device.
- Difficulties in designing and implementing the temperature sensing device with compatible software and hardware.

Backup plans:

- Reducing the complexity and number of features within the project to provide a partial solution to the problem.
- Changing technologies if the technologies established in the design become unsuitable during the project’s implementation due to unforeseen complications.

Plan for reviewing risks:

Risk description and type	**Risk impact**	**Risk probability**	**Mitigation**	**First indicator**
Running out of time to implement and connect all parts of the system.	High, could result in an incomplete system with limited usability.	Medium	Detailed project planning so that time is allocated for each part of the system or reducing the number of features in the system.	Consistently not meeting deadlines within the design and implementation of the system.
Problems obtaining the technology for the sensor device.	High, unable to build the sensor device or could result in significant delays.	Low	Detailed research into what technology will be required and where they can be purchased or changing the technology being used.	Not meeting deadlines when designing and implementing the sensor device.
Problems building the sensor device.	High, the sensor device is required for the rest of the system to be usable and problem to be solved.	Medium	Detailed research early in the project into what technology is required and how to implement them into the device.	Not meeting deadlines for building the sensor device.

10. Starting points for research

There are various starting points for research on Internet-of-Things home automation systems, including systems that use sensors to detect the local climate and control other devices based on the climate data.

Commercial Solutions:

- Netatmo
- WeatherFlow
- BloomSky SKY2 Weather Station
- BRESSER WIFI Professional Weather Station
- AcuRite Atlas
- BloomSky
- Davis Vantage Vue

Academic Research:

- Bluetooth based home automation system
- Bluetooth Remote Home Automation System Using Android Application
- Cloud based low-cost Home Monitoring and Automation System
- Raspberry Pi as Sensor Node and Hardware of the Internet of Things (Iot) for Smart

Home

- An Internet based wireless home automation system for multifunctional devices
- The state of automated amateur weather observations
- How good are citizen weather stations? Addressing a biased opinion

11. Breakdown of tasks

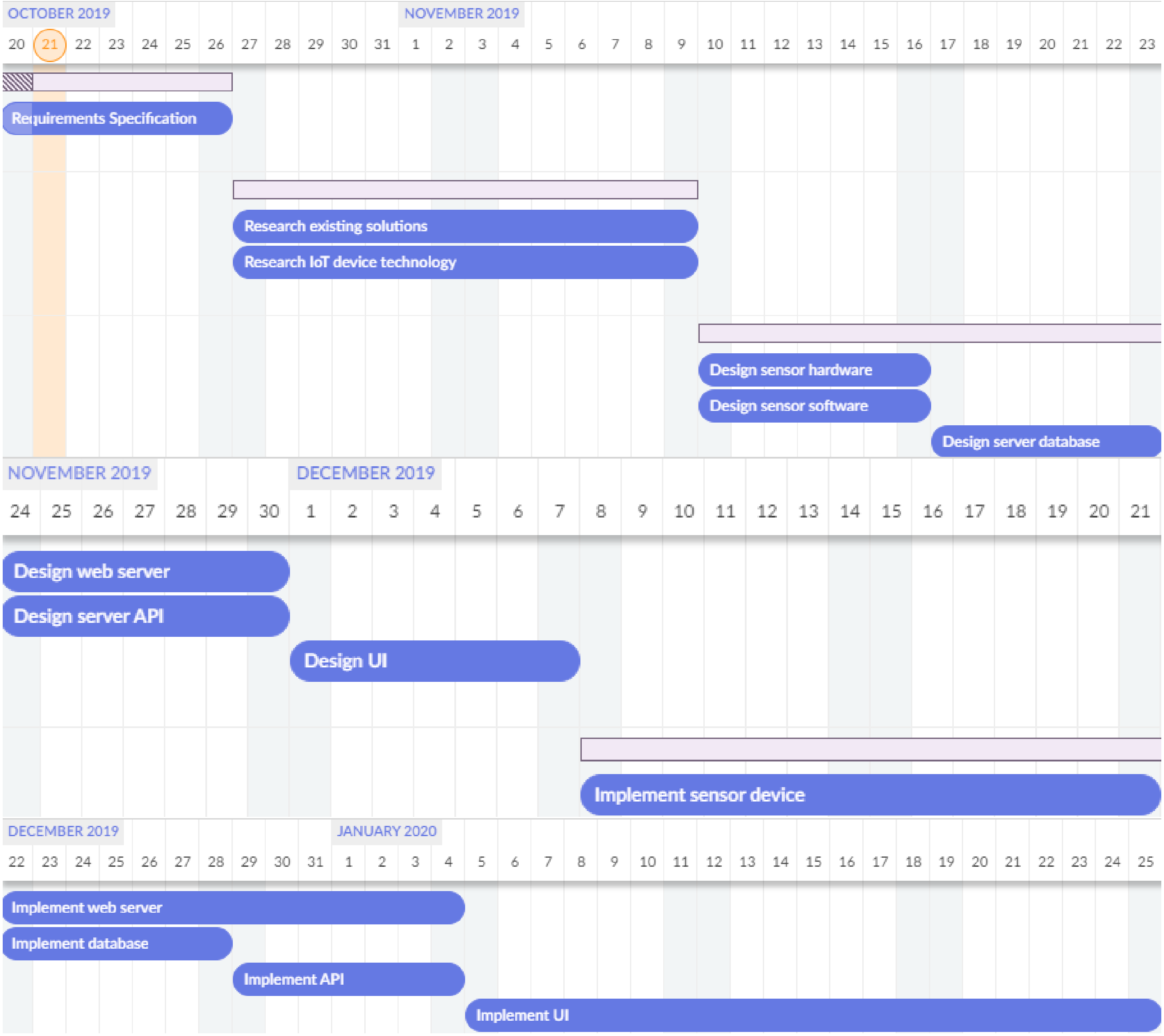
Walkthrough of approach:

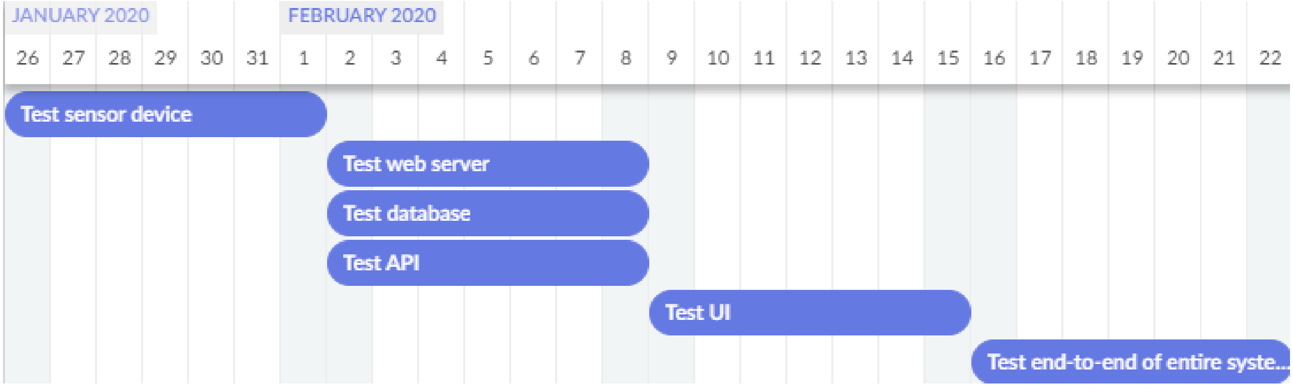
To create the system, the approach described in point 7 will be followed which results in the tasks being completed sequentially. Due to this the requirements elicitation is completed first, following with the research, design, implementation, testing, evaluation, and report writing steps. Overlap on some sub-tasks is required as some parts of the system will rely on each other to be implemented.

Project tasks and sub-tasks:

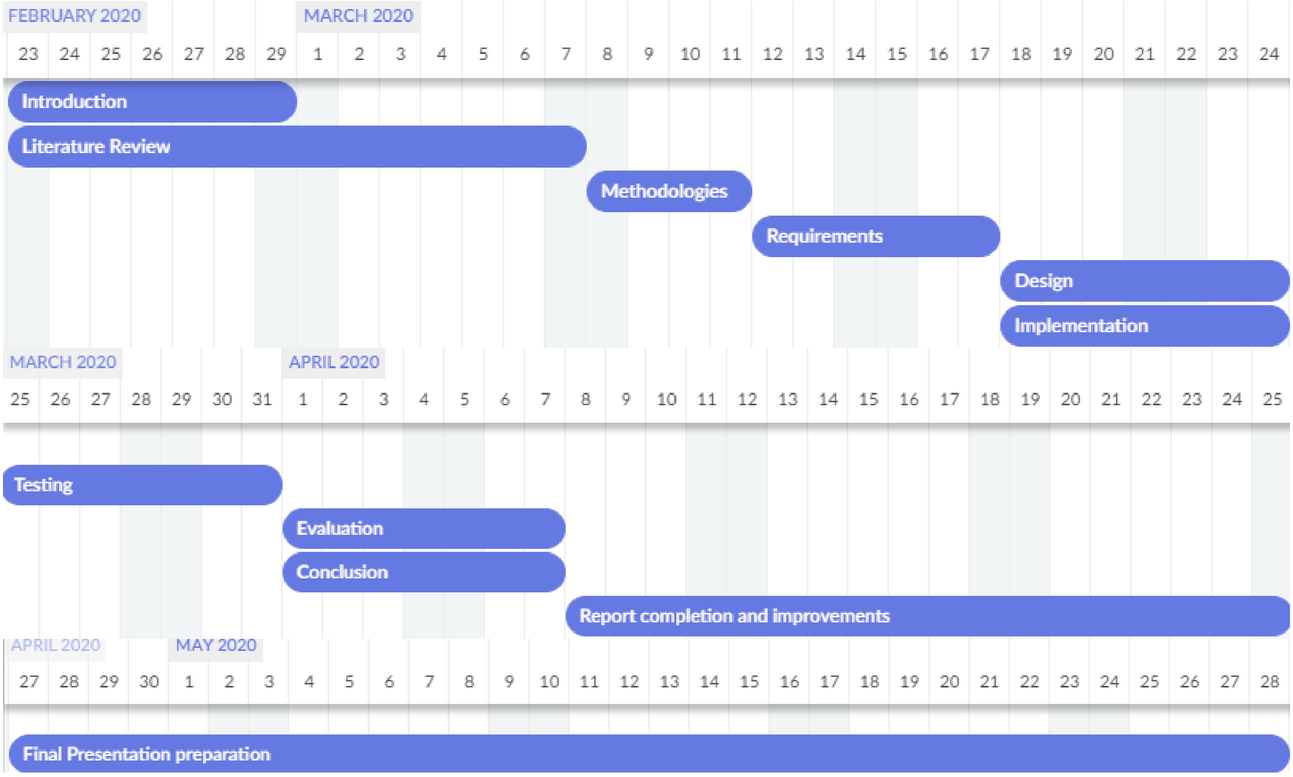
1. Requirements specification.
2. Research into existing commercial and research projects
3. Research into available IoT technology for the sensor device
4. System design
   - 4.1. Design sensor device hardware and software
   - 4.2. Design web-server
   - 4.3. Design API
   - 4.4. Design database
   - 4.5. Design user interface
5. System implementation
   - 5.1. Implement sensor device hardware and software
   - 5.2. Implement web-server
   - 5.3. Implement API
   - 5.4. Implement database
   - 5.5. Implement user interface
6. System testing
   - 6.1. Test sensor device
   - 6.2. Test database
   - 6.3. Test web server
   - 6.4. Test API
   - 6.5. Test user interface
   - 6.6. Test end-to-end of the entire system
7. Project evaluation
8. Writing of the final report
   - 8.1. Introduction
   - 8.2. Literature Review
   - 8.3. Methodologies and Project management
   - 8.4. Requirements and Analysis
   - 8.5. Design
   - 8.6. Implementation
   - 8.7. Testing
   - 8.8. Evaluation
   - 8.9. Conclusion

12. Project plan





Final Report Plan:



Risks to the success of the project:
The amount of time required to research, design, and implement each part of the system is a risk, as the complexity of the system requires all parts to be operational for the system to correctly solve the proposed problem.

There is a risk that obtaining the technology required to build a suitable sensor device could have unforeseen delays as technology will be subject to financial constraints and availability at retailers.

Steps to minimise the risks:

- Following a detailed plan with milestones for each task, so that project progress can be easily understood and that potential delays and missed milestones can act as a warning that overall progress is not on track.
- Regular supervisor-student meetings to ensure that regular meaningful work is produced and that progress is maintained across the duration of the project.

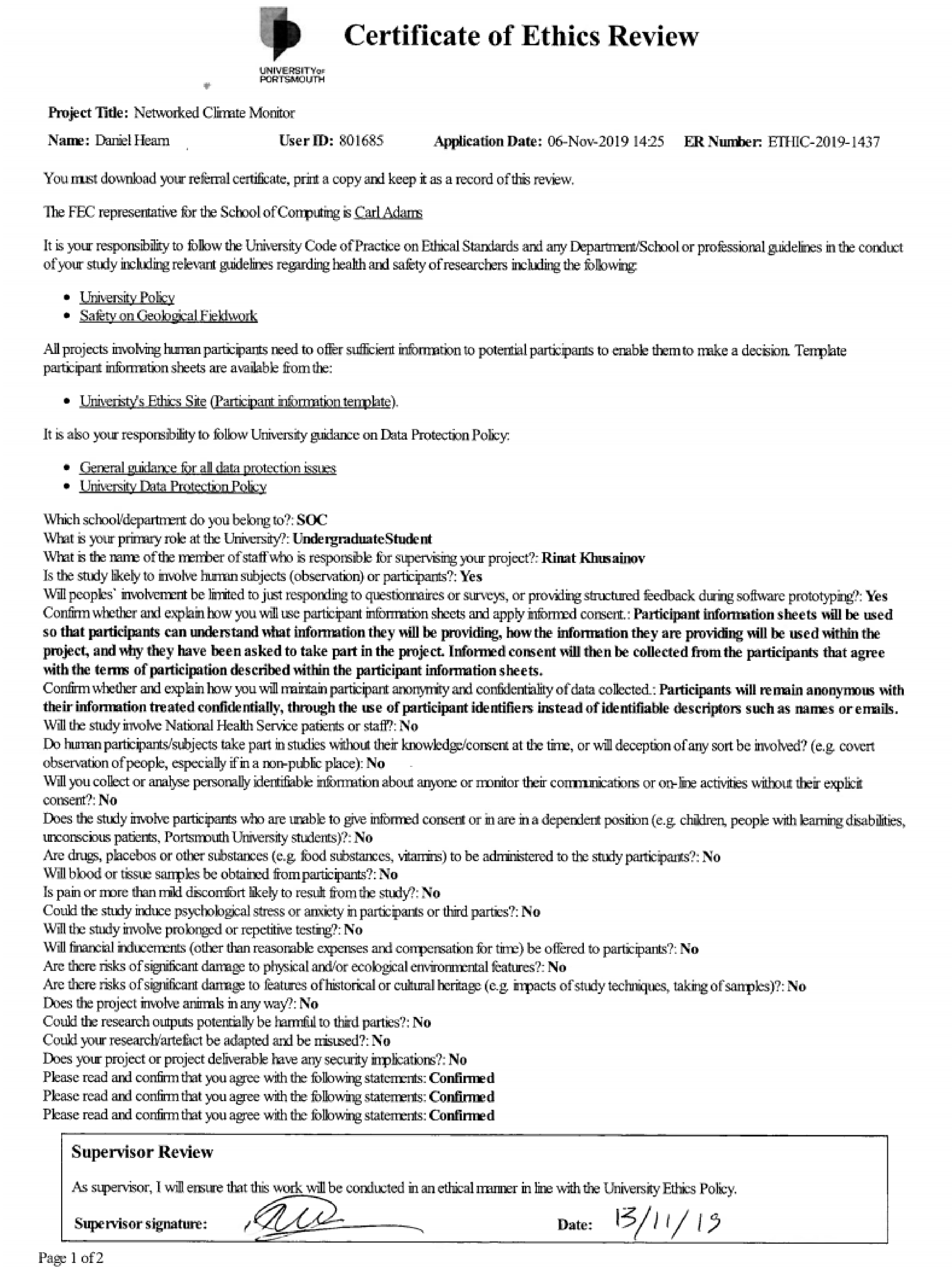
13. Legal, ethical, professional, social issues

There are no legal/ethical/professional/social issues that may impose constraints on the project.

Potential security implications:
There is potential for the system to store a user’s email and password in a database, but this will depend on how the system is designed, how users are described within the system, and whether user’s are given access to the system for testing.

Ethics approval:
Ethics approval will not be required for the project as it will not be tested by participants or be publicly released or accessible, so potential security concerns for user data can be disregarded.

11.2 Appendix B: Ethics certificate



UNIVERSITY OF PORTSMOUTH

Certificate of Ethics Review

Project Title: Networked Climate Monitor

Name: Daniel Hearn **User ID:** 801685 **Application Date:** 06-Nov-2019 14:25 **ER Number:** ETHIC-2019-1437

You must download your referral certificate, print a copy and keep it as a record of this review.

The FEC representative for the School of Computing is Carl Adams

It is your responsibility to follow the University Code of Practice on Ethical Standards and any Department/School or professional guidelines in the conduct of your study including relevant guidelines regarding health and safety of researchers including the following:

- University Policy
- Safety on Geological Fieldwork

All projects involving human participants need to offer sufficient information to potential participants to enable them to make a decision. Template participant information sheets are available from the:

- Univeristy's Ethics Site (Participant information template).

It is also your responsibility to follow University guidance on Data Protection Policy:

- General guidance for all data protection issues
- University Data Protection Policy

Which school/department do you belong to?: **SOC**
What is your primary role at the University?: **UndergraduateStudent**
What is the name of the member of staff who is responsible for supervising your project?: **Rinat Khusainov**
Is the study likely to involve human subjects (observation) or participants?: **Yes**
Will peoples' involvement be limited to just responding to questionnaires or surveys, or providing structured feedback during software prototyping?: **Yes**
Confirm whether and explain how you will use participant information sheets and apply informed consent.: **Participant information sheets will be used so that participants can understand what information they will be providing, how the information they are providing will be used within the project, and why they have been asked to take part in the project. Informed consent will then be collected from the participants that agree with the terms of participation described within the participant information sheets.**
Confirm whether and explain how you will maintain participant anonymity and confidentiality of data collected.: **Participants will remain anonymous with their information treated confidentially, through the use of participant identifiers instead of identifiable descriptors such as names or emails.**
Will the study involve National Health Service patients or staff?: **No**
Do human participants/subjects take part in studies without their knowledge/consent at the time, or will deception of any sort be involved? (e.g. covert observation of people, especially if in a non-public place): **No**
Will you collect or analyse personally identifiable information about anyone or monitor their communications or on-line activities without their explicit consent?: **No**
Does the study involve participants who are unable to give informed consent or in are in a dependent position (e.g. children, people with learning disabilities, unconscious patients, Portsmouth University students)?: **No**
Are drugs, placebos or other substances (e.g. food substances, vitamins) to be administered to the study participants?: **No**
Will blood or tissue samples be obtained from participants?: **No**
Is pain or more than mild discomfort likely to result from the study?: **No**
Could the study induce psychological stress or anxiety in participants or third parties?: **No**
Will the study involve prolonged or repetitive testing?: **No**
Will financial inducements (other than reasonable expenses and compensation for time) be offered to participants?: **No**
Are there risks of significant damage to physical and/or ecological environmental features?: **No**
Are there risks of significant damage to features of historical or cultural heritage (e.g. impacts of study techniques, taking of samples)?: **No**
Does the project involve animals in any way?: **No**
Could the research outputs potentially be harmful to third parties?: **No**
Could your research/artefact be adapted and be misused?: **No**
Does your project or project deliverable have any security implications?: **No**
Please read and confirm that you agree with the following statements: **Confirmed**
Please read and confirm that you agree with the following statements: **Confirmed**
Please read and confirm that you agree with the following statements: **Confirmed**

Supervisor Review

As supervisor, I will ensure that this work will be conducted in an ethical manner in line with the University Ethics Policy.

Supervisor signature: **Date:** 13/11/19

Page 1 of 2

11.3 Appendix C: Source code link

https://github.com/DanielHearn/networked_climate_monitor

11.4 Appendix D: Requirements background research

Name	WeatherFlow
Source Link	https://shop.weatherflow.com/products/smart-home-weather-stations
System cost	$300 USD (£234 converted as of 25/10/2019)
Modularity	Partly modular as it includes two outdoor sensors that capture different data. But there is no option to buy sensors individually only to buy the two sensors and a base station as a set.
Inside/Outside sensors	Two outdoor sensors with an indoor base station.
Types of data recorded	Air sensor detects: ambient light, UV index, solar radiation, wind speed, wind direction, and rain intensity. Sky sensor detects: temperature, humidity, atmospheric pressure, lightning.
Features of the website and/or application	Proprietary website and mobile application, with both displaying the same data. Features: ● Latest recording for all types of sensor data ● Historical data recording for each day with high, low and average values for each data type ● Historical data displayed in a chart for each data type from the last hour, last day, last week, and last month. ● Sensor battery status ● Weekly forecast for the sensor location ● Community sensor map
Method of sensor communication	*"AIR and SKY connect via long range telemetry to an indoor receiving hub."*
Method of sensor powering	Air sensor powered by 8 AA batteries, with the Sky sensor powered by 4 AA batteries or solar power.
Sensor battery life	Up to one year for AIR and SKY sensors.
Method of base station powering	Mains connection.
Connectable services	Weather Underground, Google Assistant, IFTTT, Amazon Alexa
Temperature accuracy	±0.4°C
Temperature range	-38°C to 60°C
Humidity accuracy	±4%.

Humidity range	0 to 100%
Maximum sensor distance from the base station	Up to 100 metres.
Sensor recording frequency	Not specified
Maximum number of devices per user	Not specified
Storage duration	Not specified

Table 1. WeatherFlow feature research

Name	Ambient Weather WS-2902 Osprey Weather Station
Source Link	https://www.ambientweather.com/amws2902.html
System cost	$129.99 (£100 converted as of 25/10/2019)
Modularity	No, one outdoor sensor and an indoor base station.
Inside/Outside sensors	Outside sensor with an indoor base station.
Types of data recorded	Outdoor Sensor detects: temperature, humidity, atmospheric pressure, wind speed, wind direction, rainfall, solar radiation, UV index, dew point. Indoor base station detects: temperature, humidity, atmospheric pressure.
Features of the website and/or application	Proprietary website and mobile application, with both displaying the same data. Features: ● Latest recording for all types of sensor data. ● Trends between latest sensor data and the previous day's sensor data. ● Historical data recording for each day with high, low and average values for each data type ● Historical data displayed in a chart for each data type from any specified date range. ● Community sensor map.
Method of sensor communication	915 MHz wireless radio transmission.
Method of sensor powering	Powered by 2 AA batteries or solar power
Sensor battery life	Not specified.

Method of base station powering	Mains
Connectable services	Weather Underground, Weathercloud, Google assistant, IFTTT, Amazon Alexa
Temperature accuracy	Indoor Temperature Accuracy: ± 0.5°C Outdoor Temperature Accuracy: ± 0.5°C
Temperature range	Indoor Temperature Range: -10 to 60°C Outdoor Temperature Sensor Range: -40 to 65°C
Humidity accuracy	Indoor Humidity Accuracy: ± 5% Outdoor Humidity Accuracy: ± 5%
Humidity range	Indoor Humidity Range: 10 to 99% Outdoor Humidity Range: 10 to 99%
Maximum sensor distance from the base station	25 metres.
Sensor recording frequency	Every minute.
Maximum number of devices per user	Up to 10 sensors per account.
Storage duration	Not specified

Table 2. Ambient Weather feature research

Name	AcuRite Atlas
Source Link	https://www.amazon.com/AcuRite-01009M-Weather-Definition-Touchscreen/dp/B074XK4BSN
System cost	$186 (£145 converted as of 25/10/2019)
Modularity	Main sensor isn't but a few additional sensor attachments are available including a lightning detection module, wind vane extension, sensor battery pack, and sensor mains power adapter. Up to 7 sensors per base station.
Inside/Outside sensors	Outside sensor with an indoor base station.
Types of data recorded	Outside sensor detects: temperatures, humidity, atmospheric pressure, UV index, wind speed, wind direction, rainfall, light intensity.
Features of the website and/or	Proprietary website and mobile application, both displaying the same data.

application	Features: ● Latest recording for all types of sensor data ● Historical data displayed in a chart for each data type from the last hour, last day, last week, and last month. ● Weekly forecast for the sensor location. ● Custom alerts triggered by thresholds for types of sensor data.
Method of sensor communication	433 MHz wireless radio transmission.
Method of sensor powering	Powered by solar power or 4 AA batteries.
Sensor battery life	Not specified.
Method of base station powering	Micro USB with a battery backup used to temporarily store data until the USB power is restored.
Connectable services	Weather Underground, Weathercloud, Amazon Alexa
Temperature accuracy	± 1°C
Temperature range	-40°C to 70°C
Humidity accuracy	± 2%
Humidity range	1% to 99%
Maximum sensor distance from the base station	Up to 100 metres.
Sensor recording frequency	Records every 30 seconds but is only stored online every 5 minutes.
Maximum number of devices per user	Not specified.
Storage duration	1 month

Table 3. AcuRite Atlas feature research

Name	Davis Vantage Vue
Source Link	https://www.davisinstruments.com/product_documents/weather/spec_sheets/6250_6351_57_SS.pdf https://www.davisinstruments.com/product/vantage-vue-wireless-weather-station/
System cost	£349.99
Modularity	No, one outdoor sensor and an indoor base station.

Inside/Outside sensors	Outside sensor with an indoor base station.
Types of data recorded	Outdoor sensor detects: temperature, humidity, wind speed, wind direction, rainfall Base station detects: temperature, humidity
Features of the website and/or application	Proprietary website and mobile application. Features: ● Latest recording for all types of sensor data ● Historical data displayed in a chart for each data type. ● Weekly forecast for the sensor location. ● Community sensor map Sending the recorded measurements to the website requires a £250 adapter to provide the internet connection. It also requires a paid monthly subscription to access the historical data and chart features on the website.
Method of sensor communication	868.0 - 868.6 MHz wireless radio transmission.
Method of sensor powering	Powered by battery and solar panel
Sensor battery life	Up to 8 months without recharging via solar power
Method of base station powering	Three C batteries.
Connectable services	None available.
Temperature accuracy	± 1°C
Temperature range	-40° to 65°C
Humidity accuracy	± 2%
Humidity range	1% to 100%
Maximum sensor distance from the base station	Up to 300 metres.
Sensor recording frequency	Every 10 seconds
Maximum number of devices per user	Not specified.
Storage duration	Not specified

Table 4. Davis Vantage Vue feature research

Name	BloomSky

Source Link	https://bloomsky.force.com/support/s/article/sky-2-specifications http://shop.bloomsky.com/products-list/sky2
System cost	£295
Modularity	No it only has an outside sensor.
Inside/Outside sensors	Outside sensor
Types of data recorded	Outside sensor detects: temperature, humidity, atmospheric pressure, rainfall. The outside sensor also regularly takes images of the sky from an onboard camera.
Features of the website and/or application	Proprietary website and mobile application. Features: ● Latest recording for all types of sensor data ● Historical data displayed in a chart for each data type. ● Weekly forecast for the sensor location. ● Community sensor map. ● Daily timelapse of the sky images
Method of sensor communication	WiFi 2.4GHz Bluetooth used to pair with a phone during the installation.
Method of sensor powering	Battery recharged by solar power or a main connection.
Sensor battery life	Two weeks without being recharged by solar power or mains connection.
Method of base station powering	N/A as the outside sensor acts as a base station
Connectable services	IFTTT
Temperature accuracy	±0.3°C
Temperature range	0°C to 65°C
Humidity accuracy	± 3%
Humidity range	10% to 90%
Maximum sensor distance from the base station	Not specified.
Sensor recording frequency	5 minutes unless a 2c temperature difference is detected
Maximum number of devices per user	Not specified.

Storage duration	Not specified

Table 5. BloomSky feature research

Name	BRESSER WiFi Professional Weather Station
Source Link	https://www.bresseruk.com/weather-station/bresser-wifi-professional-weather-station.html https://www.bresser.de/out/media/b5c6a05f16c23d43124fa62889a2edb8.pdf
System cost	£197
Modularity	No, one outdoor sensor and an indoor base station.
Inside/Outside sensors	Outdoor sensor and indoor base station.
Types of data recorded	Outdoor sensor detects: temperature, wind speed, atmospheric pressure, UV, humidity, rainfall.
Features of the website and/or application	Does not have its own system but instead uses weather underground to view the data on its website or mobile application.
Method of sensor communication	868 MHz wireless radio transmission.
Method of sensor powering	Powered by 3 AA batteries Has backup solar power
Sensor battery life	Not specified.
Method of base station powering	Mains
Connectable services	Weather underground (Required to view sensor data via the internet)
Temperature accuracy	Not specified.
Temperature range	-40°C to 60°C
Humidity accuracy	± 1%
Humidity range	1% to 99%
Maximum sensor distance from the base station	Up to 150 metres.
Sensor recording frequency	Not specified.
Maximum number of	Not specified.

devices per user	
Storage duration	Not specified

Table 6. BRESSER feature research

Name	Netatmo
Source Link	https://www.netatmo.com/en-gb/weather/weatherstation
System cost	£149.99
Modularity	Partly comes with an outside sensor and an inside sensor. Additional sensors can be added to the same system, these are up to 3 additional indoor modules, 1 rain gauge, and 1 outdoor sensor.
Inside/Outside sensors	Outside and inside sensors.
Types of data recorded	Outside sensor detects: temperature, humidity, wind speed, wind direction, sound levels, Co2 levels, Inside sensor detects: temperature, humidity, air quality.
Features of the website and/or application	Proprietary website and mobile application. Features: ● Latest recording for all types of sensor data ● List of available sensors for that account. ● Sensor battery level, signal. ● Historical data displayed in a chart for each data type. ○ Can change day, week, month, year for graph ○ See previous day, week, month, year ● Weekly forecast for the sensor location. ● Community sensor map.
Method of sensor communication	WiFi 2.4GHz
Method of sensor powering	Outside sensor 4 AAA batteries Outside sensor 4 AAA batteries
Sensor battery life	Inside sensor claims 1-year life Outside sensor claims 2-year life
Method of base station powering	Inside sensor acts as a base station, so 4 AAA batteries.
Connectable services	Not available.
Temperature accuracy	± 0.3°C
Temperature range	Inside sensor: 0°C to 50°C Outside sensor: -40°C to 65°C

Humidity accuracy	± 3%
Humidity range	0 to 100%
Maximum sensor distance from the base station	Up to 100 metres.
Sensor recording frequency	Every 5 minutes.
Maximum number of devices per user	Unlimited per account.
Storage duration	Not specified

Table 7. Netatmo feature research

11.5 Appendix E: API Endpoint Documentation

Name	Authenticate user from email and password
Description	Authenticates that the input email and password match the stored email and password in the database and returns the authentication token.
Url	/login
Method	Post
Header Parameters	None
Path Parameters	None
Query Parameters	None
Body Parameters	<table><tr><th>Field</th><th>Type</th><th>Required</th><th>Description</th></tr><tr><td>password</td><td>String</td><td>Yes</td><td>Password.</td></tr><tr><td>email</td><td>String</td><td>Yes</td><td>Email.</td></tr></table>

Success Response	Code: 200
	Body: {
	status: “Successful login”,
	access_token: <string>,
	refresh_token: <string>
	}

Error Response	**Incorrect Credentials**
	Code: 401
	Body: {
	status: “Error”,
	errors: [“Email or password is incorrect”]
	}

Table 1. Login Endpoint Design

Name	Logout access token
Description	Blacklists the access token so that the access token can no longer be used for authentication.

<table>
<tr><td><b>Url</b></td><td>/logout/access</td></tr>
<tr><td><b>Method</b></td><td>Post</td></tr>
<tr><td><b>Header Parameters</b></td><td><table>
<tr><th>Field</th><th>Type</th><th>Required</th><th>Description</th></tr>
<tr><td>Authorization</td><td>String</td><td>Yes</td><td>Authentication token that has been received from successful login or registration.</td></tr>
</table></td></tr>
<tr><td><b>Path Parameters</b></td><td>None</td></tr>
<tr><td><b>Query Parameters</b></td><td>None</td></tr>
<tr><td><b>Body Parameters</b></td><td>None</td></tr>
<tr><td><b>Success Response</b></td><td>Code: 200
Body: {
  status: “Access token has been revoked”,
}</td></tr>
<tr><td><b>Error Response</b></td><td><b>Incorrect Credentials</b>
Code: 401
Body: {
  status: “Error”,
  errors: [“Access token invalid”]
}

<b>Missing Auth Token</b>
Code: 401
Body: {
  status: “Error”,
  errors: [“Missing Authorization Header”]
}</td></tr>
</table>

Table 2. Logout Access Token Endpoint Design

Name	Logout refresh token
Description	Blacklists the refresh token so that the refresh token can no longer be used for obtaining a new access token.
Url	/logout/refresh
Method	Post

<table>
<tr><td><b>Header Parameters</b></td><td><table><tr><th>Field</th><th>Type</th><th>Required</th><th>Description</th></tr><tr><td>Authorization</td><td>String</td><td>Yes</td><td>Refresh token that has been received from successful login or registration.</td></tr></table></td></tr>
<tr><td><b>Path Parameters</b></td><td>None</td></tr>
<tr><td><b>Query Parameters</b></td><td>None</td></tr>
<tr><td><b>Body Parameters</b></td><td>None</td></tr>
<tr><td><b>Success Response</b></td><td>Code: 200
Body: {
  status: “Refresh token has been revoked”,
}</td></tr>
<tr><td><b>Error Response</b></td><td><b>Incorrect Credentials</b>
Code: 401
Body: {
  status: “Error”,
  errors: [“Refresh token invalid”]
}

<b>Missing Auth Token</b>
Code: 401
Body: {
  status: “Error”,
  errors: [“Missing Authorization Header”]
}</td></tr>
</table>

Table 3. Logout Refresh Token Endpoint Design

<table>
<tr><td><b>Name</b></td><td>Refresh access token</td></tr>
<tr><td><b>Description</b></td><td>Provides a new access token once provided a valid refresh token.</td></tr>
<tr><td><b>Url</b></td><td>/token/refresh</td></tr>
<tr><td><b>Method</b></td><td>Post</td></tr>
<tr><td><b>Header Parameters</b></td><td><table><tr><th>Field</th><th>Type</th><th>Required</th><th>Description</th></tr></table></td></tr>
</table>

<table>
<tr><td></td><td><table><tr><td>Refresh</td><td>String</td><td>Yes</td><td>Refresh token that has been received from successful login or registration.</td></tr></table></td></tr>
<tr><td><b>Path Parameters</b></td><td>None</td></tr>
<tr><td><b>Query Parameters</b></td><td>None</td></tr>
<tr><td><b>Body Parameters</b></td><td>None</td></tr>
<tr><td><b>Success Response</b></td><td>Code: 200
Body: {
  status: “Successful refresh”,
  access_token: <string>,
}</td></tr>
<tr><td><b>Error Response</b></td><td><b>Incorrect Credentials</b>
Code: 401
Body: {
  status: “Error”,
  errors: [“Refresh token invalid”]
}</td></tr>
</table>

Table 4. Refresh Access Token Endpoint Design

<table>
<tr><td><b>Name</b></td><td>Add new sensor</td></tr>
<tr><td><b>Description</b></td><td>Adds a new sensor to the collection if the user is authorised.</td></tr>
<tr><td><b>Url</b></td><td>/sensors</td></tr>
<tr><td><b>Method</b></td><td>Post</td></tr>
<tr><td><b>Header Parameters</b></td><td>None</td></tr>
<tr><td><b>Path Parameters</b></td><td>None</td></tr>
<tr><td><b>Query Parameters</b></td><td><table><tr><th>Field</th><th>Type</th><th>Required</th><th>Description</th></tr><tr><td>api_key</td><td>String</td><td>Yes</td><td>Api key used only by the base station radio program.</td></tr></table></td></tr>
</table>

<table>
<tr><td><b>Body Parameters</b></td><td>
<table>
<tr><th>Field</th><th>Type</th><th>Required</th><th>Description</th></tr>
<tr><td>name</td><td>String</td><td>Yes</td><td>Name of the sensor.</td></tr>
<tr><td>sensor_id</td><td>Integer</td><td>Yes</td><td>ID of the sensor, this ID cannot already exist within the collection.</td></tr>
</table>
</td></tr>
<tr><td><b>Success Response</b></td><td>Code: 200
Body: {
status: “Sensor successfully created.”
}</td></tr>
<tr><td><b>Error Response</b></td><td><b>Missing name field</b>
Code: 422
Body: {
    "status": "Error",
    "errors": {
        "name": [
            "Missing data for required field."
        ]
    }
}
<b>Missing sensor_id field</b>
Code: 422
Body: {
    "status": "Error",
    "errors": {
        "sensor_id": [
            "Missing data for required field."
        ]
    }
}

<b>Missing user_id field</b>
Code: 422
Body: {
    "status": "Error",
    "errors": {
        "user_id": [
            "Missing data for required field."
        ]
    }
}

<b>Name field isn’t a string</b>
Code: 422
Body: {
    "status": "Error",</td></tr>
</table>

	"errors": {
	"name": [
	"Not a valid string."
	]
	}
	}
	
	sensor_id field isn’t an integer
	Code: 422
	Body: {
	"status": "Error",
	"errors": {
	"sensor_id": [
	"Not a valid integer."
	]
	}
	}
	
	user_id field isn’t an integer
	Code: 422
	Body: {
	"status": "Error",
	"errors": {
	"user_id": [
	"Not a valid integer."
	]
	}
	}
	
	Duplicate ID
	Code: 400
	Body: {
	status: “Error”,
	errors: [“'A sensor with that id already exists”]
	}
	
	Invalid or Missing API key
	Code: 401
	Body: {
	status: “Error”,
	errors: [“Invalid api key”]
	}

Table 5. Add Sensor Endpoint Design

Name	Get all sensors
Description	Gets all sensor data from the collection.
Url	/sensors

<table>
<tr><td><b>Method</b></td><td>Get</td></tr>
<tr><td><b>Header Parameters</b></td><td><table>
<tr><th>Field</th><th>Type</th><th>Required</th><th>Description</th></tr>
<tr><td>Authorization</td><td>String</td><td>Yes</td><td>Authentication token that has been received from successful login or registration.</td></tr>
</table></td></tr>
<tr><td><b>Path Parameters</b></td><td>None</td></tr>
<tr><td><b>Query Parameters</b></td><td>None</td></tr>
<tr><td><b>Body Parameters</b></td><td>None</td></tr>
<tr><td><b>Success Response</b></td><td>Code: 200
Body: {
  status: “Sensors successfully retrieved”
  sensors: [
    {
      name: <string>
      identifier: <string>
      recent_climate_data: {
          time_date: <string in ISO 8601 format>
          battery_voltage: <float>
          sensor_data: [
            {
               type: <string>
               value: <float>
               unit: <string>
            }
          ]
        }
    }
  ]
}</td></tr>
<tr><td><b>Error Response</b></td><td><b>Missing Authentication Token</b>
Code: 401
Body: {
  status: “Error”,
  errors: [“Missing authentication token”]
}

<b>Invalid Authentication Token</b>
Code: 401</td></tr>
</table>

	Body: {
	  status: “Error”,
	  errors: [“Invalid authentication token”]
	}

Table 6. Get Sensors Endpoint Design

Name	Delete all sensors
Description	Deletes all sensor data from the collection.
Url	/sensors
Method	Delete
Header Parameters	<table><tr><th>Field</th><th>Type</th><th>Required</th><th>Description</th></tr><tr><td>Authorization</td><td>String</td><td>Yes</td><td>Authentication token that has been received from successful login or registration.</td></tr></table>
Path Parameters	None
Query Parameters	None
Body Parameters	None

Success Response	Code: 200
	Body: {
	  status: “Sensors successfully deleted”
	}

Error Response	**Missing Authentication Token**
	Code: 401
	Body: {
	  status: “Error”,
	  errors: [“Missing authentication token”]
	}
	
	Invalid Authentication Token
	Code: 401
	Body: {
	  status: “Error”,
	  errors: [“Invalid authentication token”]

	}

Table 7. Delete Sensors Endpoint Design

<table>
<tr><td><b>Name</b></td><td>Update sensor</td></tr>
<tr><td><b>Description</b></td><td>Updates an existing sensor with changes to some or all of the keys.</td></tr>
<tr><td><b>Url</b></td><td>/sensors/{sensor_id}</td></tr>
<tr><td><b>Method</b></td><td>Patch</td></tr>
<tr><td><b>Header Parameters</b></td><td>
<table>
<tr><th>Field</th><th>Type</th><th>Required</th><th>Description</th></tr>
<tr><td>Authorization</td><td>String</td><td>Yes</td><td>Authentication token that has been received from successful login or registration.</td></tr>
</table>
</td></tr>
<tr><td><b>Path Parameters</b></td><td>
<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>sensor_id</td><td>ID of the sensor that will have its data updated.</td></tr>
</table>
</td></tr>
<tr><td><b>Query Parameters</b></td><td>None</td></tr>
<tr><td><b>Body Parameters</b></td><td>
<table>
<tr><th>Field</th><th>Type</th><th>Required</th><th>Description</th></tr>
<tr><td>name</td><td>String</td><td>No</td><td>Name of the sensor.</td></tr>
<tr><td>sensor_id</td><td>Integer</td><td>No</td><td>ID of the sensor, this ID cannot already exist within the collection.</td></tr>
</table>
</td></tr>
<tr><td><b>Success Response</b></td><td>Code: 200
Body: {
  status: “Sensor successfully updated”
}</td></tr>
<tr><td><b>Error Response</b></td><td><b>Name field isn’t a string</b>
Code: 422
Body: {
    "status": "Error",
    "errors": {
        "name": [
            "Not a valid string."</td></tr>
</table>

	]
	}
	}
	
	sensor_id field isn’t an integer
	Code: 422
	Body: {
	"status": "Error",
	"errors": {
	"sensor_id": [
	"Not a valid integer."
	]
	}
	}
	
	Sensor doesn’t exist
	Code: 400
	Body: {
	status: “Error”,
	errors: [“Sensor doesn’t exist”]
	}
	
	Missing Authentication Token
	Code: 401
	Body: {
	status: “Error”,
	errors: [“Missing authentication token”]
	}
	
	Invalid Authentication Token
	Code: 401
	Body: {
	status: “Error”,
	errors: [“Invalid authentication token”]
	}

Table 8. Update Sensor Endpoint Design

<table>
<tr><td><b>Name</b></td><td>Get sensor</td></tr>
<tr><td><b>Description</b></td><td>Gets sensor data for a specified sensor ID</td></tr>
<tr><td><b>Url</b></td><td>/sensors/{sensor_id}</td></tr>
<tr><td><b>Method</b></td><td>Get</td></tr>
<tr><td><b>Header Parameters</b></td><td>
<table>
<tr><th>Field</th><th>Type</th><th>Required</th><th>Description</th></tr>
<tr><td>Authorization</td><td>String</td><td>Yes</td><td>Authentication token that has been</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td></td><td><table><tr><td></td><td></td><td></td><td>received from successful login or registration.</td></tr></table></td></tr>
<tr><td><b>Path Parameters</b></td><td><table><tr><th>Field</th><th>Type</th><th>Required</th><th>Description</th></tr><tr><td>sensor_id</td><td>Integer</td><td>Yes</td><td>ID of the sensor that will have its data retrieved</td></tr></table></td></tr>
<tr><td><b>Query Parameters</b></td><td>None</td></tr>
<tr><td><b>Body Parameters</b></td><td>None</td></tr>
<tr><td><b>Success Response</b></td><td>Code: 200
Body: {
    status: “Sensor successfully retrieved”
    sensor: {
      name: <string>
      id: <integer>
      user_id: <integer>
      recent_climate_data: {
          time_date: <string in ISO 8601 format>
          battery_voltage: <float>
          sensor_data: [
            {
               type: <string>
               value: <float>
               unit: <string>
            }
          ]
       }
   }
}</td></tr>
<tr><td><b>Error Response</b></td><td><b>Sensor doesn’t exist</b>
Code: 400
Body: {
  status: “Error”,
  errors: [“Sensor doesn’t exist”]
}

<b>Missing Authentication Token</b>
Code: 401
Body: {
  status: “Error”,
  errors: [“Missing authentication token”]
}</td></tr>
</table>

	Invalid Authentication Token
	Code: 401
	Body: {
	  status: “Error”,
	  errors: [“Invalid authentication token”]
	}

Table 9. Get Sensor Endpoint Design

Name	Delete sensor
Description	Deletes the sensor that has the specified sensor ID
Url	/sensors/{sensor_id}
Method	Delete
Header Parameters	**Field** / **Type** / **Required** / **Description** Authorization / String / Yes / Authentication token that has been received from successful login or registration.
Path Parameters	**Field** / **Type** / **Required** / **Description** sensor_id / Integer / Yes / ID of the sensor that will be deleted.
Query Parameters	None
Body Parameters	None

Success Response	Code: 200
	Body: {
	  status: “Sensor successfully deleted”
	}

Error Response	**Sensor doesn’t exist**
	Code: 400
	Body: {
	  status: “Error”,
	  errors: [“Sensor doesn’t exist”]
	}
	
	Missing Authentication Token

	Code: 401
	Body: {
	  status: “Error”,
	  errors: [“Missing authentication token”]
	}
	
	Invalid Authentication Token
	Code: 401
	Body: {
	  status: “Error”,
	  errors: [“Invalid authentication token”]
	}

Table 10. Delete Sensor Endpoint Design

<table>
<tr><td><b>Name</b></td><td>Sensor Data Setting</td></tr>
<tr><td><b>Description</b></td><td>Adds new climate data for the specified sensor.</td></tr>
<tr><td><b>Url</b></td><td>/sensors/{sensor_id}/climate-data</td></tr>
<tr><td><b>Method</b></td><td>Post</td></tr>
<tr><td><b>Header Parameters</b></td><td>None</td></tr>
<tr><td><b>Path Parameters</b></td><td>
<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>sensor_id</td><td>ID of the sensor that collected the climate data.</td></tr>
</table>
</td></tr>
<tr><td><b>Query Parameters</b></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Required</th><th>Description</th></tr>
<tr><td>api_key</td><td>String</td><td>Yes</td><td>Api key used only by the base station radio program.</td></tr>
</table>
</td></tr>
<tr><td><b>Body Parameters</b></td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Required</th><th>Description</th></tr>
<tr><td>date</td><td>String(ISO 8601 format)</td><td>Yes</td><td>Date and time that the climate data was recorded at.</td></tr>
<tr><td>climate_data</td><td>Array</td><td>Yes</td><td>Array of objects in the following format:</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td></td><td>
<table>
<tr><td></td><td></td><td></td><td>{
data_type: <string>
measurement: <float>
unit: <string>
}</td></tr>
<tr><td>battery_voltage</td><td>Float</td><td>Yes</td><td>Battery voltage of the sensor node</td></tr>
</table>
</td></tr>
<tr><td><b>Success Response</b></td><td>Code: 200
Body: {
status: “Sensor data successfully created.”
}</td></tr>
<tr><td><b>Error Response</b></td><td><b>Climate_data isn’t a list</b>
Code: 422
Body: {
    "status": "Error",
    "errors": {
        "climate_data": [
            "Not a valid list."
        ]
    }
}

<b>Unit field in climate_data isn’t a string</b>
Code: 422
Body: {
    "status": "Error",
    "errors": {
        "climate_data": {
            "0": {
                "unit": [
                    "Not a valid string."
                ]
            }
        }
    }
}

<b>Value field in climate_data isn’t a number</b>
Code: 422
Body: {
    "status": "Error",
    "errors": {
        "climate_data": {
            "0": {
                "value": [
                    "Not a valid number."
                ]
            }</td></tr>
</table>

```
        }
    }
}
```

Type field in climate_data isn’t a string

```
Code: 422
Body: {
    "status": "Error",
    "errors": {
        "climate_data": {
            "0": {
                "type": [
                    "Not a valid string."
                ]
            }
        }
    }
}
```

Battery_voltage isn’t a float number

```
Code: 422
Body: {
    "status": "Error",
    "errors": {
        "battery_voltage": [
            "Not a valid number."
        ]
    }
}
```

Incorrect Date

```
Code: 400
Body: {
    "status": "Error",
    "errors": {
        "date": [
            "Not a valid datetime."
        ]
    }
}
```

Sensor Does Not Exist

```
Code: 400
Body: {
  status: “Error”,
  errors: [“Sensor doesn’t exist”]
}
```

Invalid or Missing API Key

```
Code: 401
Body: {
```

	status: “Error”, errors: [“Invalid API key”] }

Table 11. Set Climate Data Endpoint Design

<table>
<tr><td><b>Name</b></td><td>Sensor Climate Data Retrieval</td></tr>
<tr><td><b>Description</b></td><td>Gets the climate data for the specified sensor based on the quantity of recent data requested or a date and time range. By default it will return the quantity is 1 so the endpoint will always return the most recent climate data in the subcollection.</td></tr>
<tr><td><b>Url</b></td><td>/sensors/{sensor_id}/climate-data</td></tr>
<tr><td><b>Method</b></td><td>Get</td></tr>
<tr><td><b>Header Parameters</b></td><td>
<table>
<tr><th>Field</th><th>Type</th><th>Required</th><th>Description</th></tr>
<tr><td>Authorization</td><td>String</td><td>Yes</td><td>Authentication token that has been received from successful login or registration.</td></tr>
</table>
</td></tr>
<tr><td><b>Path Parameters</b></td><td>
<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>sensor_id</td><td>ID of the sensor that collected the climate data.</td></tr>
</table>
</td></tr>
<tr><td><b>Query Parameters</b></td><td>
<table>
<tr><th>Field</th><th>Type</th><th>Required</th><th>Description</th></tr>
<tr><td>quantity</td><td>Integer</td><td>No</td><td>Quantity of the climate data objects required, based on how recently they were added to the database.</td></tr>
<tr><td>range_start</td><td>String(ISO 8601 format)</td><td>No</td><td>Start of the time and date range.</td></tr>
<tr><td>range_end</td><td>String(ISO 8601 format)</td><td>No</td><td>End of the time and date range.</td></tr>
</table>
</td></tr>
<tr><td><b>Body Parameters</b></td><td>None</td></tr>
<tr><td><b>Success Response</b></td><td>Code: 200
Body: {</td></tr>
</table>

	status: “Climate data successfully retrieved”
	climate_data: [
	{
	time_date: <string in ISO 8601 format>
	battery_voltage: <float>
	sensor_data: [
	{
	type: <string>
	value: <float>
	unit: <string>
	}
	]
	}
	]
	}

Error Response	**Quantity isn’t an integer**
	Code: 422
	Body: {
	"status": "Error",
	"errors": [
	"Quantity must be an integer"
	]
	}
	
	Sensor does not Exist
	Code: 400
	Body: {
	status: “Error”,
	errors: [“Sensor doesn’t exist”]
	}
	
	Quantity is invalid
	Code: 422
	Body: {
	status: “Error”,
	errors: [“Quantity must be below or equal to 50”]
	}
	
	Date range from range_start to range_end is invalid
	Code: 422
	Body: {
	status: “Error”,
	errors: [“Invalid date range”]
	}
	
	Missing Authentication Token
	Code: 401
	Body: {
	status: “Error”,
	errors: [“Missing authentication token”]

	}
	
	Invalid Authentication Token
	Code: 401
	Body: {
	  status: “Error”,
	  errors: [“Invalid authentication token”]
	}

Table 12. Get Climate Data Endpoint Design

<table>
<tr><td><b>Name</b></td><td>Sensor Climate Data Deletion</td></tr>
<tr><td><b>Description</b></td><td>Deletes the climate data for the sensor that has the specified sensor ID.</td></tr>
<tr><td><b>Url</b></td><td>/sensors/{sensor_id}/climate-data</td></tr>
<tr><td><b>Method</b></td><td>Delete</td></tr>
<tr><td><b>Header Parameters</b></td><td><table><tr><th>Field</th><th>Type</th><th>Required</th><th>Description</th></tr><tr><td>Authorization</td><td>String</td><td>Yes</td><td>Authentication token that has been received from successful login or registration.</td></tr></table></td></tr>
<tr><td><b>Path Parameters</b></td><td><table><tr><th>Field</th><th>Description</th></tr><tr><td>sensor_id</td><td>ID of the sensor that collected the climate data.</td></tr></table></td></tr>
<tr><td><b>Query Parameters</b></td><td>None</td></tr>
<tr><td><b>Body Parameters</b></td><td>None</td></tr>
<tr><td><b>Success Response</b></td><td>Code: 200
Body: {
  Status: “Data successfully deleted”
}</td></tr>
<tr><td><b>Error Response</b></td><td><b>Sensor does not exist</b>
Code: 400
Body: {
  status: “Error”,
  errors: [“No climate data for the sensor ID”]
}</td></tr>
</table>

	Missing Authentication Token
	Code: 401
	Body: {
	  status: “Error”,
	  errors: [“Missing authentication token”]
	}
	
	Invalid Authentication Token
	Code: 401
	Body: {
	  status: “Error”,
	  errors: [“Invalid authentication token”]
	}

Table 13. Delete Climate Data Endpoint Design

<table>
<tr><td><b>Name</b></td><td>Account Creation</td></tr>
<tr><td><b>Description</b></td><td>Creates the account based on the input email and password and returns the authentication token.</td></tr>
<tr><td><b>Url</b></td><td>/account</td></tr>
<tr><td><b>Method</b></td><td>Post</td></tr>
<tr><td><b>Header Parameters</b></td><td>None</td></tr>
<tr><td><b>Path Parameters</b></td><td>None</td></tr>
<tr><td><b>Query Parameters</b></td><td>None</td></tr>
<tr><td><b>Body Parameters</b></td><td><table>
<tr><th>Field</th><th>Type</th><th>Required</th><th>Description</th></tr>
<tr><td>email</td><td>String</td><td>Yes</td><td>Email of the account</td></tr>
<tr><td>password</td><td>String</td><td>Yes</td><td>Password of the account</td></tr>
</table></td></tr>
<tr><td><b>Success Response</b></td><td>Code: 200
Body: {
  status: “Account was successfully created”,
  access_token: <string>,
  refresh_token: <string>,
  reset_token: <string>
}</td></tr>
<tr><td><b>Error</b></td><td><b>Email isn’t a string</b></td></tr>
</table>

Response	Code: 422
	Body: {
	"status": "Error",
	"errors": {
	"email": [
	"Not a valid email address."
	]
	}
	}
	
	Password isn’t a string
	Code: 422
	Body: {
	"status": "Error",
	"errors": {
	"password": [
	"Not a valid string."
	]
	}
	}
	
	Password isn’t between 8 to 40 characters long
	Code: 422
	Body: {
	"status": "Error",
	"errors": {
	"password": [
	"Length must be between 8 and 40."
	]
	}
	}

Table 14. Create Account Endpoint Design

<table>
<tr><td><b>Name</b></td><td>Account Updating</td></tr>
<tr><td><b>Description</b></td><td>Updates the account based on the optional body parameters email and password.</td></tr>
<tr><td><b>Url</b></td><td>/account</td></tr>
<tr><td><b>Method</b></td><td>Patch</td></tr>
<tr><td><b>Header Parameters</b></td><td>
<table>
<tr><th>Field</th><th>Type</th><th>Required</th><th>Description</th></tr>
<tr><td>Authorization</td><td>String</td><td>Yes</td><td>Authentication token that has been received from successful login or registration.</td></tr>
</table>
</td></tr>
<tr><td><b>Path</b></td><td>None</td></tr>
</table>

<table>
<tr><td><b>Parameters</b></td><td></td></tr>
<tr><td><b>Query Parameters</b></td><td>None</td></tr>
<tr><td><b>Body Parameters</b></td><td>
<table>
<tr><th>Field</th><th>Type</th><th>Required</th><th>Description</th></tr>
<tr><td>email</td><td>String</td><td>No</td><td>Email of the account</td></tr>
<tr><td>password</td><td>String</td><td>No</td><td>Password of the account</td></tr>
<tr><td>settings</td><td>String</td><td>No</td><td>Encoded JSON string of the account settings</td></tr>
</table>
</td></tr>
<tr><td><b>Success Response</b></td><td>Code: 200
Body: {
  status: “Account successfully updated”
}</td></tr>
<tr><td><b>Error Response</b></td><td><b>Email isn’t a string</b>
Code: 422
Body: {
    "status": "Error",
    "errors": {
        "email": [
            "Not a valid email address."
        ]
    }
}

<b>Password isn’t a string</b>
Code: 422
Body: {
    "status": "Error",
    "errors": {
        "password": [
            "Not a valid string."
        ]
    }
}

<b>Settings isn’t a string</b>
Code: 422
Body: {
    "status": "Error",
    "errors": {
        "settings": [
            "Not a valid string."
        ]</td></tr>
</table>

	}
	}
	
	Password isn’t between 8 to 40 characters long
	Code: 422
	Body: {
	"status": "Error",
	"errors": {
	"password": [
	"Length must be between 8 and 40."
	]
	}
	}
	
	Account not created
	Code: 400
	Body: {
	status: “Error”,
	errors: [“The account has not been created”]
	}
	
	Missing Authentication Token
	Code: 401
	Body: {
	status: “Error”,
	errors: [“Missing authentication token”]
	}
	
	Invalid Authentication Token
	Code: 401
	Body: {
	status: “Error”,
	errors: [“Invalid authentication token”]
	}

Table 15. Update Account Endpoint Design

<table>
<tr><td><b>Name</b></td><td>Account Details Retrieval</td></tr>
<tr><td><b>Description</b></td><td>Gets the account details.</td></tr>
<tr><td><b>Url</b></td><td>/account</td></tr>
<tr><td><b>Method</b></td><td>Get</td></tr>
<tr><td><b>Header Parameters</b></td><td>
<table>
<tr><th>Field</th><th>Type</th><th>Required</th><th>Description</th></tr>
<tr><td>Authorization</td><td>String</td><td>Yes</td><td>Authentication token that has been received from successful login or</td></tr>
</table>
</td></tr>
</table>

	registration.
Path Parameters	None
Query Parameters	None
Body Parameters	None

Success Response	Code: 200
	Body: {
	Status: ‘Account successfully retrieved’,
	account: {
	id: <integer>,
	email: <string>,
	settings: <string>,
	reset_token: <string>
	}
	}

Error Response	**Account not created**
	Code: 400
	Body: {
	status: “Error”,
	errors: [“The account has not been created”]
	}
	
	Missing Authentication Token
	Code: 401
	Body: {
	status: “Error”,
	errors: [“Missing authentication token”]
	}
	
	Invalid Authentication Token
	Code: 401
	Body: {
	status: “Error”,
	errors: [“Invalid authentication token”]
	}

Table 16. Get Account Endpoint Design

Name	Password Reset
Description	Changes the users password if the reset key is valid.

<table>
<tr><td><b>Url</b></td><td>/account/actions/change-password</td></tr>
<tr><td><b>Method</b></td><td>Get</td></tr>
<tr><td><b>Header Parameters</b></td><td>None</td></tr>
<tr><td><b>Path Parameters</b></td><td>None</td></tr>
<tr><td><b>Query Parameters</b></td><td>None</td></tr>
<tr><td><b>Body Parameters</b></td><td>
<table>
<tr><th>Field</th><th>Type</th><th>Required</th><th>Description</th></tr>
<tr><td>reset_token</td><td>String</td><td>Yes</td><td>Reset token obtained on login.</td></tr>
<tr><td>password</td><td>String</td><td>Yes</td><td>New password to replace the existing password.</td></tr>
</table>
</td></tr>
<tr><td><b>Success Response</b></td><td>Code: 200
Body: {
  status: “Successfully reset password”,
  new_reset_token: <string>
}</td></tr>
<tr><td><b>Error Response</b></td><td><b>Password isn’t a string</b>
Code: 422
Body: {
    "status": "Error",
    "errors": {
        "password": [
            "Not a valid string."
        ]
    }
}

<b>Password isn’t between 8 to 40 characters long</b>
Code: 422
Body: {
    "status": "Error",
    "errors": {
        "password": [
            "Length must be between 8 and 40."
        ]
    }
}</td></tr>
</table>

	Account Not Created
	Code: 400
	Body: {
	  status: “Error”,
	  errors: [“The account has not been created”]
	}
	
	Invalid Reset Token
	Code: 400
	Body: {
	  status: “Error”,
	  errors: [“Invalid password reset token”]
	}

Table 17. Reset Password Endpoint Design

<table>
<tr><td><b>Name</b></td><td>Next Available Sensor ID</td></tr>
<tr><td><b>Description</b></td><td>Returns the next unused sensor ID based on the currently used sensor IDs</td></tr>
<tr><td><b>Url</b></td><td>/api/sensors/actions/next-available-sensor-id</td></tr>
<tr><td><b>Method</b></td><td>Get</td></tr>
<tr><td><b>Header Parameters</b></td><td>None</td></tr>
<tr><td><b>Path Parameters</b></td><td>None</td></tr>
<tr><td><b>Query Parameters</b></td><td>
<table>
<tr><th>Field</th><th>Type</th><th>Required</th><th>Description</th></tr>
<tr><td>api_key</td><td>String</td><td>Yes</td><td>Api key used only by the base station radio program.</td></tr>
</table>
</td></tr>
<tr><td><b>Body Parameters</b></td><td>None</td></tr>
<tr><td><b>Success Response</b></td><td>Code: 200
Body: {
  status: “Next available ID found”,
  ID: <integer>
}</td></tr>
<tr><td><b>Error Response</b></td><td><b>Invalid or Missing API Key</b>
Code: 401</td></tr>
</table>

	Body: {
	  status: “Error”,
	  errors: [“Invalid API key”]
	}

Table 18. Get Next Available Node ID Endpoint Design

<table>
<tr><td><b>Name</b></td><td>Get Account Settings</td></tr>
<tr><td><b>Description</b></td><td>Returns the account settings</td></tr>
<tr><td><b>Url</b></td><td>/api/base-station-settings</td></tr>
<tr><td><b>Method</b></td><td>Get</td></tr>
<tr><td><b>Header Parameters</b></td><td>None</td></tr>
<tr><td><b>Path Parameters</b></td><td>None</td></tr>
<tr><td><b>Query Parameters</b></td><td>
<table>
<tr><th>Field</th><th>Type</th><th>Required</th><th>Description</th></tr>
<tr><td>api_key</td><td>String</td><td>Yes</td><td>Api key used only by the base station radio program.</td></tr>
</table>
</td></tr>
<tr><td><b>Body Parameters</b></td><td>None</td></tr>
<tr><td><b>Success Response</b></td><td>Code: 200
Body: {
  status: “Next available ID found”,
  settings: <string>
}</td></tr>
<tr><td><b>Error Response</b></td><td><b>Invalid or Missing API Key</b>
Code: 401
Body: {
  status: “Error”,
  errors: [“Invalid API key”]
}

<b>Account Not Created</b>
Code: 500
Body: {</td></tr>
</table>

	status: “Error”, errors: [“The account has not been created”] }

Table 19. Get Account Settings Endpoint Design

11.6 Appendix F: Database Data Types

Field	Reason for type and size
id	The integer type was selected for the primary key, as only one account is required though this can be incremented if required.
password	The password will be hashed so a long string limit of 120 characters is required while the actual password will only be 8 to 40 characters long.
email	A 120-character string field was selected to allow for longer email addresses.
settings	The settings are stored as a JSON string so an unlimited length text field is required as the string will be encoded and decoded on the API and front-end.
reset_token	A 20-character string field was selected as the user will need to write it down in a safe location thus it shouldn't be too long, but long enough to secure from being guessed.

Table 1. User table data type justification

Field	Reason for type and size
id	The integer type was selected for the primary key as it can be incremented when a new sensor is created.
user_id	The Integer type was chosen for the foreign key to link the sensor with the user according the user's primary key.
name	A 40-character string field was selected as the sensor names will not be too long as they won't need to be too descriptive, but long enough to contain the name of a room or location.

Table 2. Sensor table data type justification

Field	Reason for type and size
id	The integer type was selected for primary key as it can be incremented when a new climate data row is created. This ensures that up to 6 months of climate data can be stored and uniquely identified.
sensor_id	The integer type was selected as the foreign key to link the climate data with the sensor according the sensor's primary key.
battery_voltage	The real field was selected as the voltage is a float with 2 decimal points.
date	The date field was selected as the date and time of the sensor recording must be stored to link the climate and sensor data to the correct time of recording.

Table 3. Climate data table data type justification

Field	Reason for type and size
id	The integer type was selected for the primary key as it can be incremented when a new sensor data row is created.
climate_id	The integer foreign key was selected to link the sensor data with the climate data according to the climate data’s primary key.
value	The float field was selected as the sensor measurements will always be numbers with decimal points regardless of the sensor type.
type	A 40-character string describing the type of data collected by the sensor was selected as the product research indicated that all measurement type names are less than 40 characters. This will lower the storage required in the database as many sensor data rows are required.
unit	A 20-character string field was used as it will be used in formatting and describing the data recorded by the sensors, and the product research showed that the measurement units were all below 20 characters.

Table 4. Sensor data table data type justification

Field	Reason for type and size
id	The integer type was selected for the primary key as it can be incremented when a new revoked JWT token row is added.
jti	A 120-character string was selected to store the JWT token as all tokens will be shorter than the field limit.

Table 5. Revoked JWT tokens table data type justification

11.7 Appendix G: UI Designs

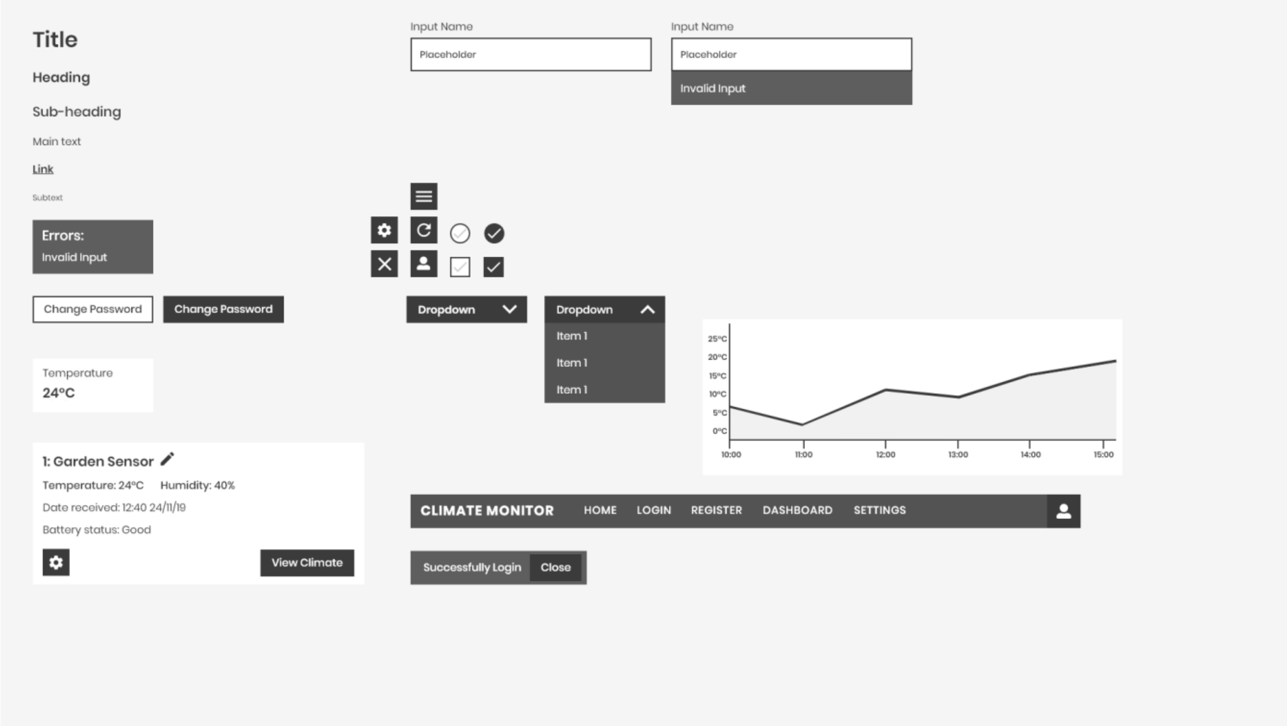
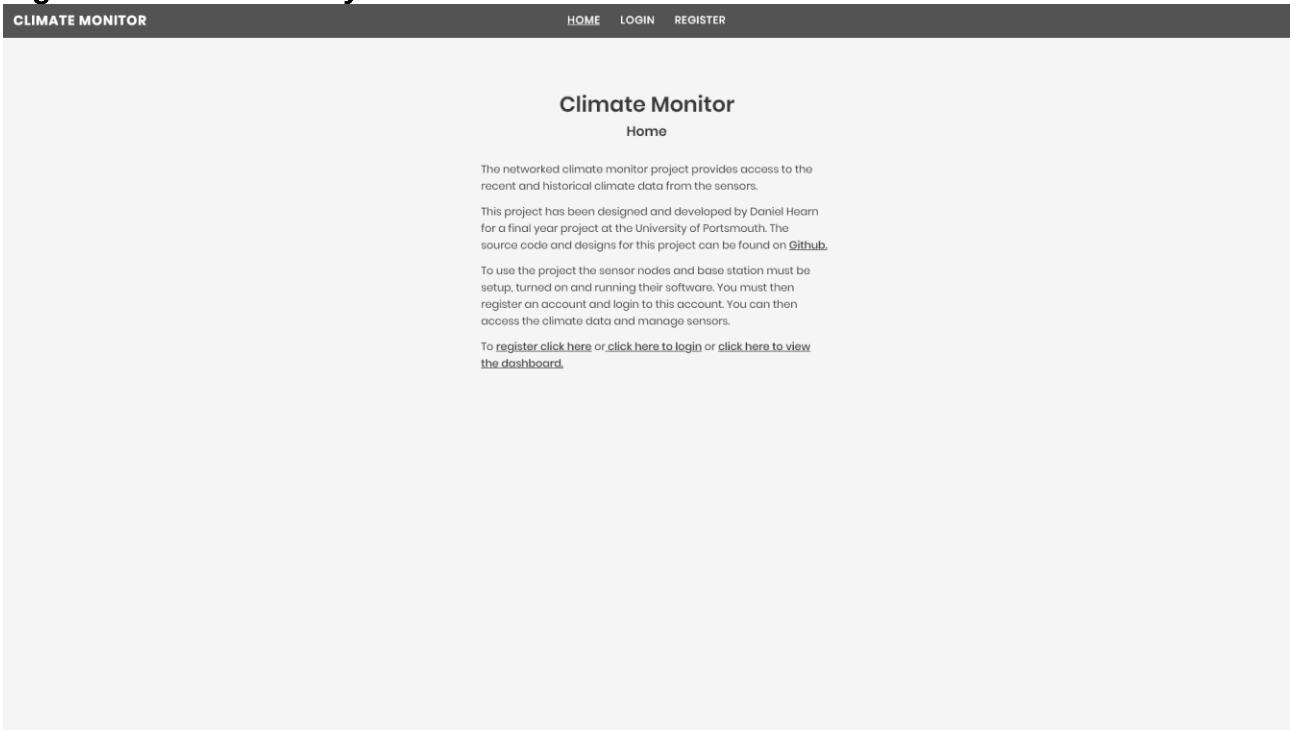


Figure 1. Wireframe Stylesheet



Climate Monitor

Home

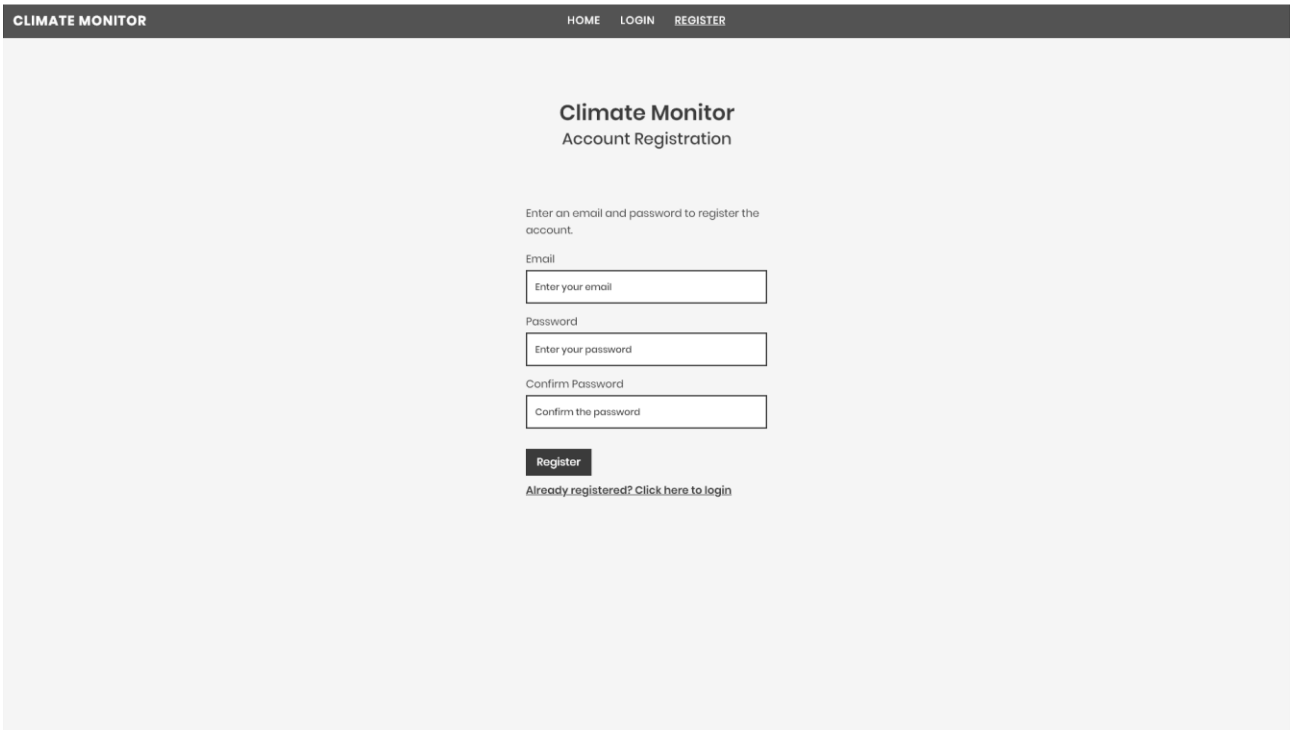
The networked climate monitor project provides access to the recent and historical climate data from the sensors.

This project has been designed and developed by Daniel Hearn for a final year project at the University of Portsmouth. The source code and designs for this project can be found on Github.

To use the project the sensor nodes and base station must be setup, turned on and running their software. You must then register an account and login to this account. You can then access the climate data and manage sensors.

To register click here or click here to login or click here to view the dashboard.

Figure 2. Wireframe Desktop Home Page



Climate Monitor

Account Registration

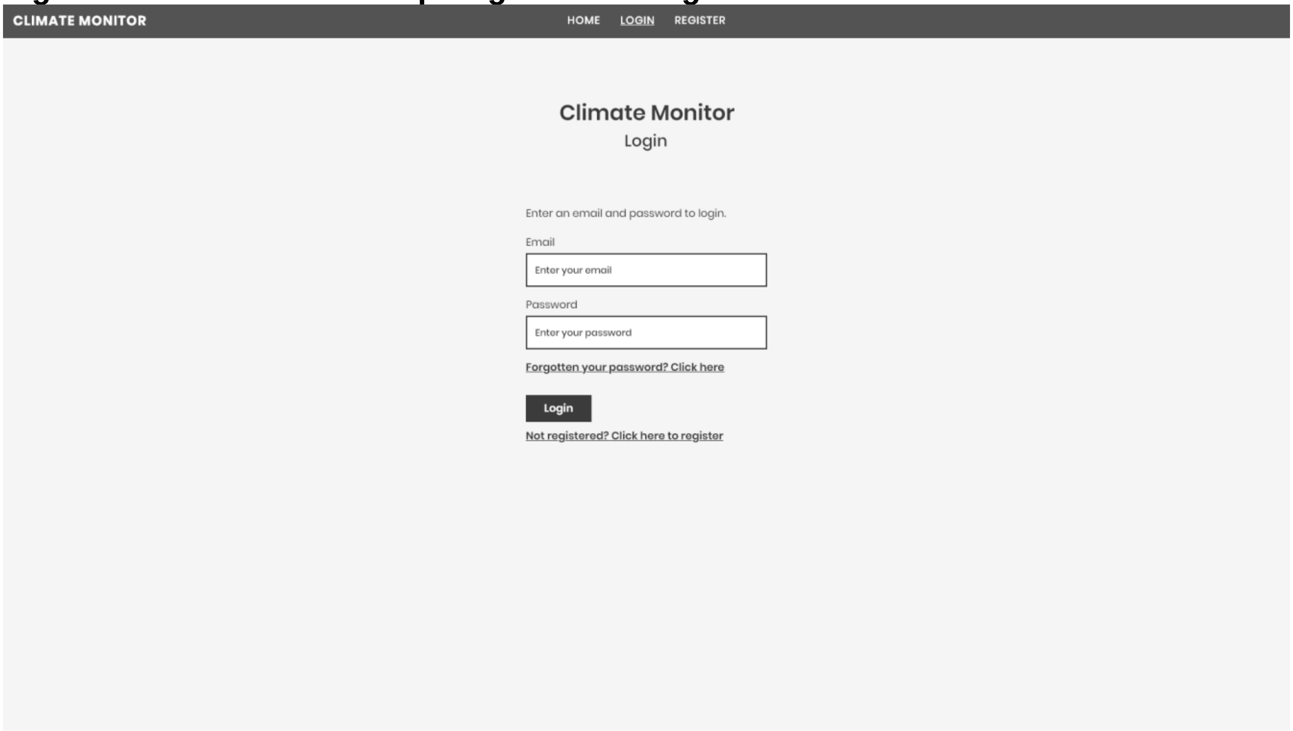
Enter an email and password to register the account.

Email: Enter your email

Password: Enter your password

Confirm Password: Confirm the password

Figure 3. Wireframe Desktop Registration Page


CLIMATE MONITOR
HOME
LOGIN
REGISTER

Climate Monitor

Login

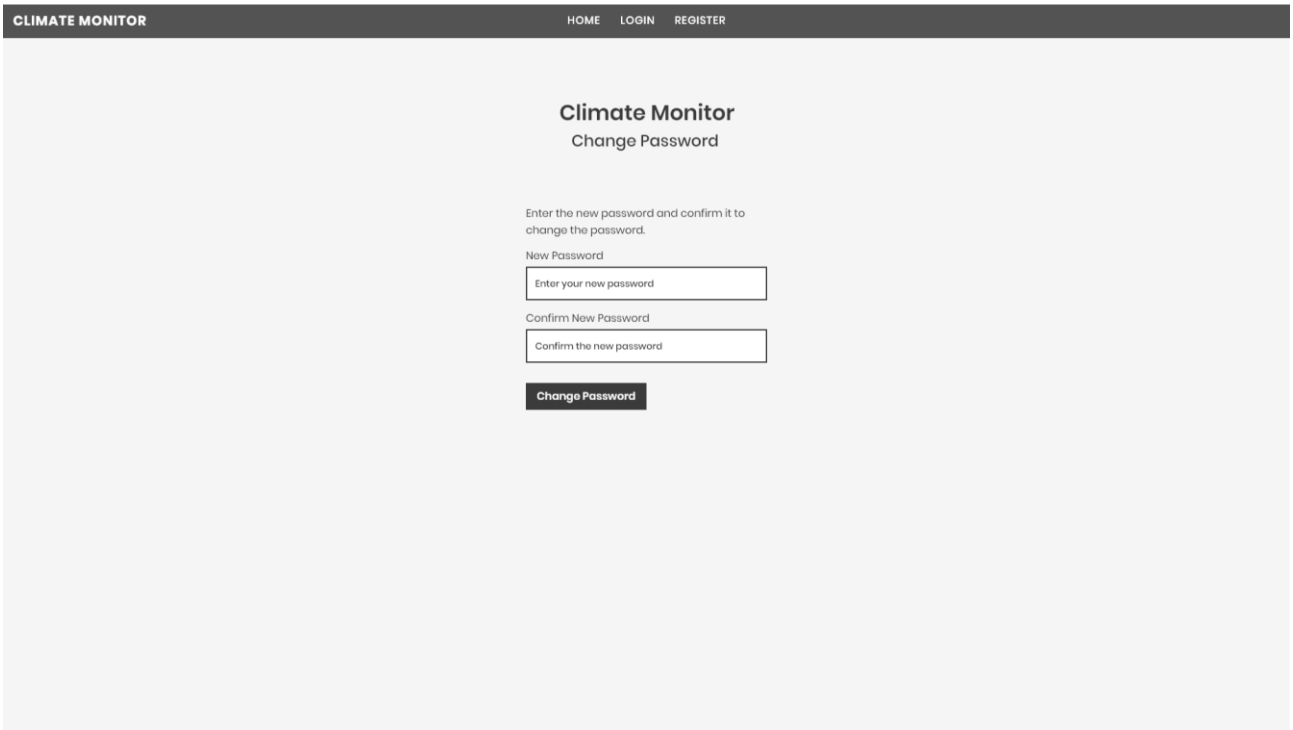
Enter an email and password to login.

Email: Enter your email

Password: Enter your password

Forgotten your password? Click here
Login
Not registered? Click here to register

Figure 4. Wireframe Desktop Login Page

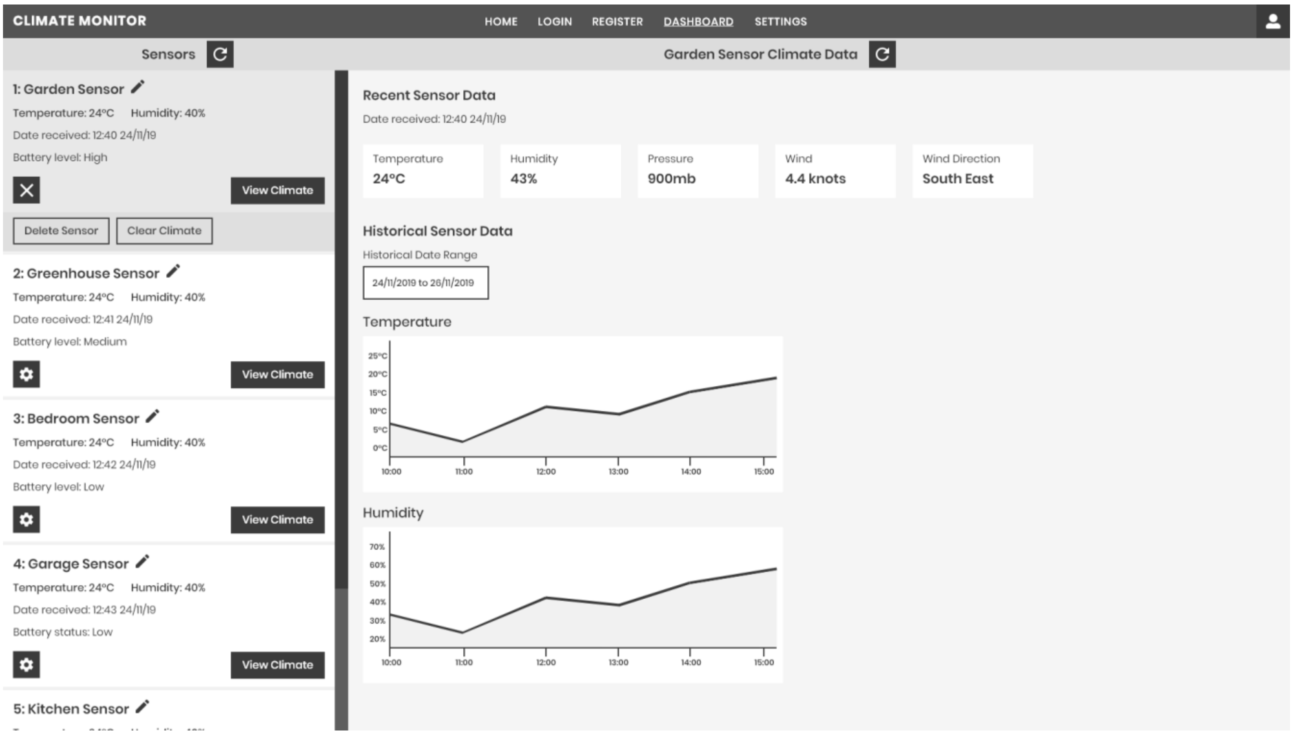


Enter the new password and confirm it to change the password.

New Password: Enter your new password

Confirm New Password: Confirm the new password

Figure 5. Wireframe Desktop Change Password Page



1: Garden Sensor

Temperature: 24°C Humidity: 40%

Date received: 12:40 24/11/19

Battery level: High

2: Greenhouse Sensor

Temperature: 24°C Humidity: 40%

Date received: 12:41 24/11/19

Battery level: Medium

3: Bedroom Sensor

Temperature: 24°C Humidity: 40%

Date received: 12:42 24/11/19

Battery level: Low

4: Garage Sensor

Temperature: 24°C Humidity: 40%

Date received: 12:43 24/11/19

Battery status: Low

5: Kitchen Sensor

Recent Sensor Data

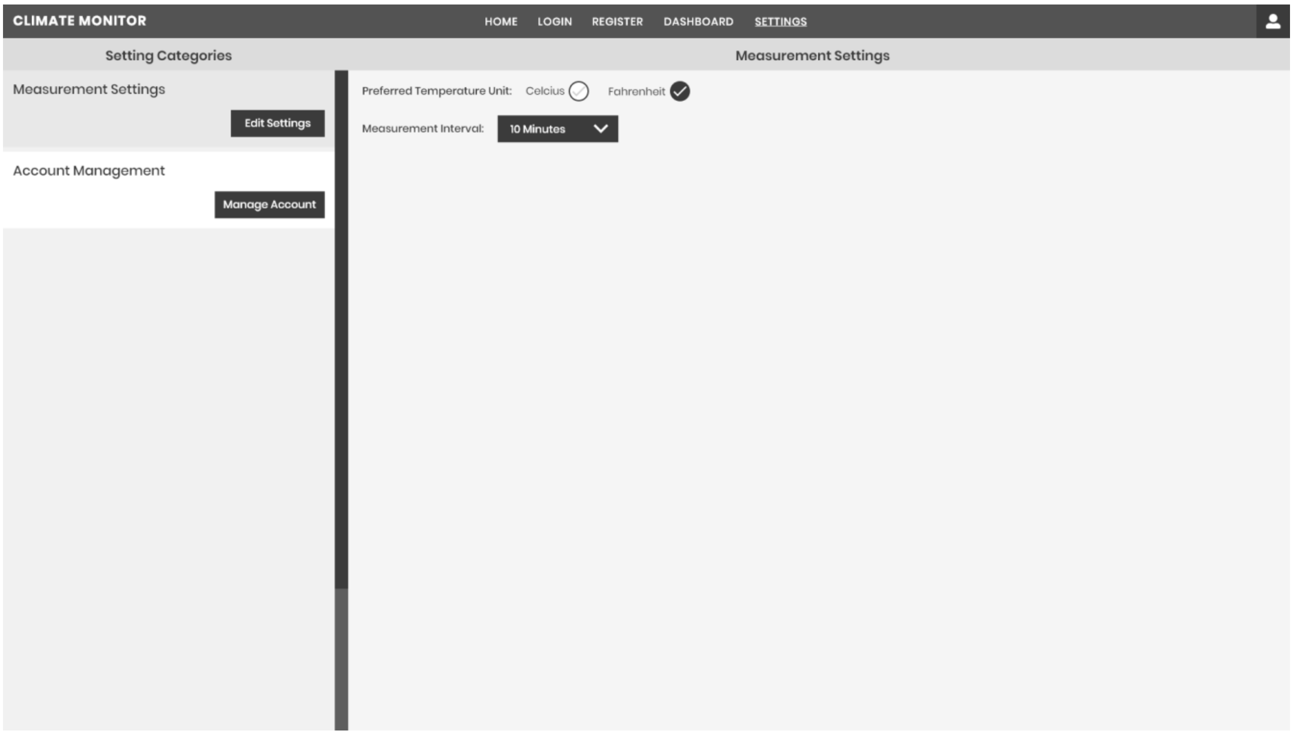
Date received: 12:40 24/11/19

Temperature	Humidity	Pressure	Wind	Wind Direction
24°C	43%	900mb	4.4 knots	South East

Historical Sensor Data

Historical Date Range: 24/11/2019 to 26/11/2019

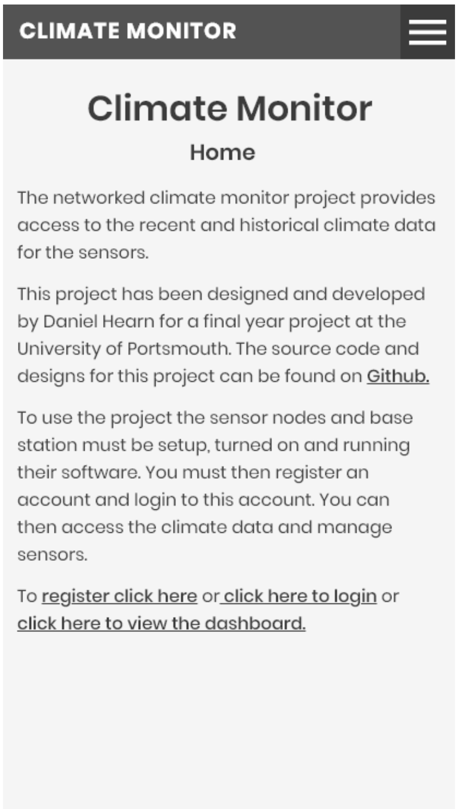
Figure 6. Wireframe Desktop Dashboard Page



Preferred Temperature Unit: Celcius ○ Fahrenheit ●

Measurement Interval: 10 Minutes

Figure 7. Wireframe Desktop Settings Page



Climate Monitor

Home

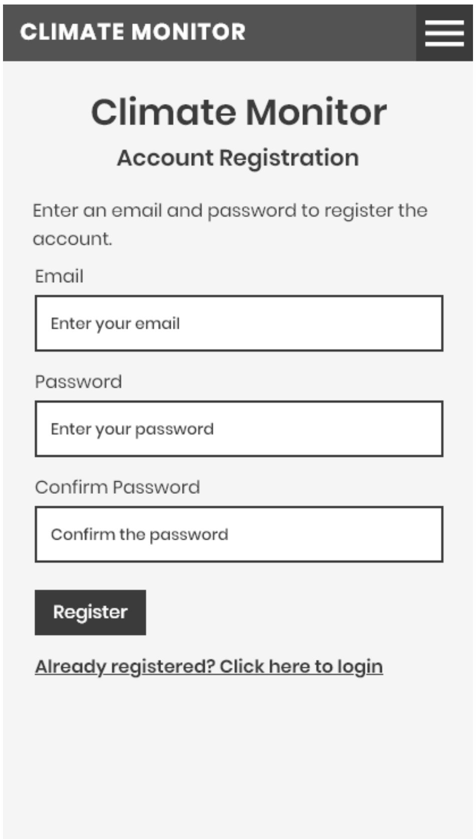
The networked climate monitor project provides access to the recent and historical climate data for the sensors.

This project has been designed and developed by Daniel Hearn for a final year project at the University of Portsmouth. The source code and designs for this project can be found on Github.

To use the project the sensor nodes and base station must be setup, turned on and running their software. You must then register an account and login to this account. You can then access the climate data and manage sensors.

To register click here or click here to login or click here to view the dashboard.

Figure 8. Wireframe Mobile Home Page



CLIMATE MONITOR

Climate Monitor

Account Registration

Enter an email and password to register the account.

Email

Enter your email

Password

Enter your password

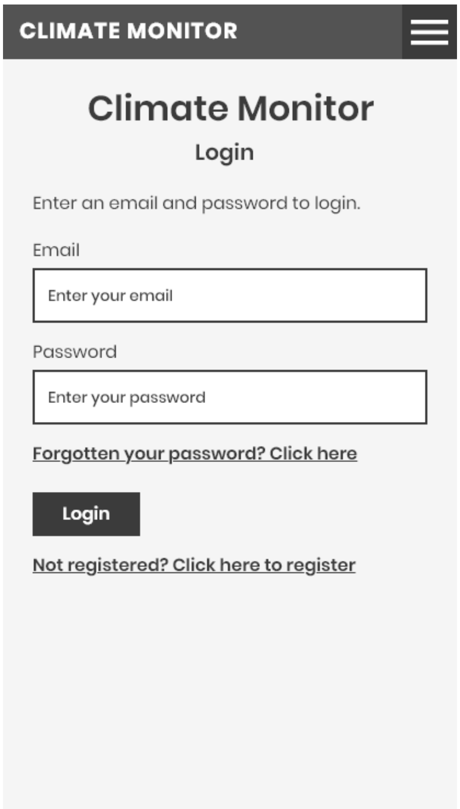
Confirm Password

Confirm the password

Register

Already registered? Click here to login

Figure 9. Wireframe Mobile Registration Page



CLIMATE MONITOR

Climate Monitor

Login

Enter an email and password to login.

Email

Enter your email

Password

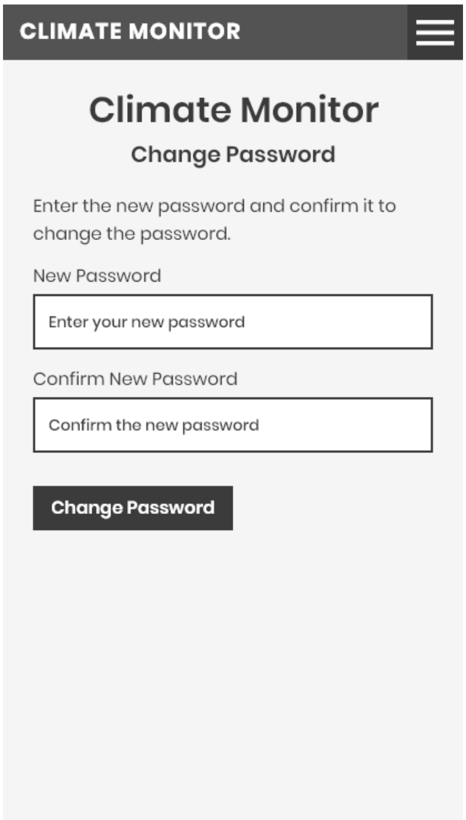
Enter your password

Forgotten your password? Click here

Login

Not registered? Click here to register

Figure 10. Wireframe Mobile Login Page

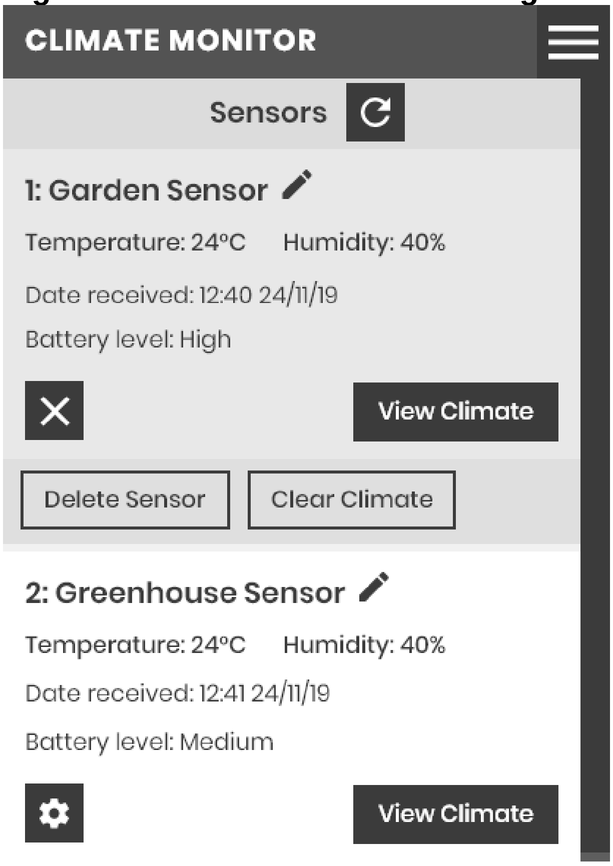


Enter the new password and confirm it to change the password.

New Password: Enter your new password

Confirm New Password: Confirm the new password

Figure 11. Wireframe Mobile Change Password Page



1: Garden Sensor

Temperature: 24°C Humidity: 40%

Date received: 12:40 24/11/19

Battery level: High

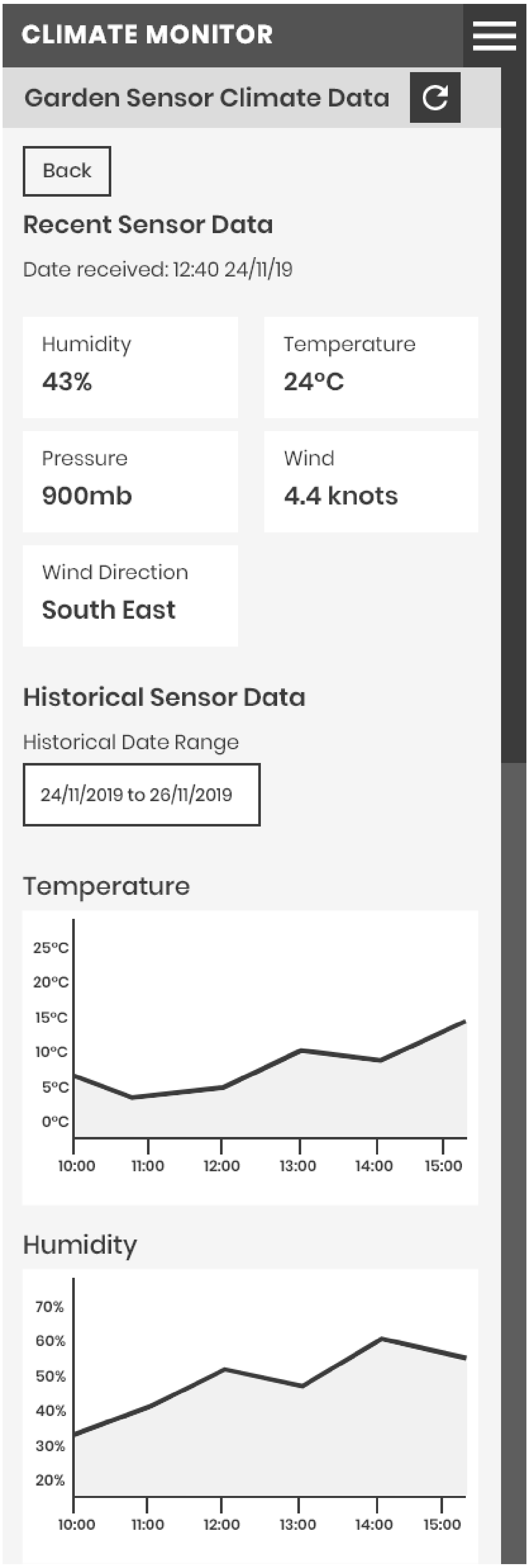
2: Greenhouse Sensor

Temperature: 24°C Humidity: 40%

Date received: 12:41 24/11/19

Battery level: Medium

Figure 12. Wireframe Mobile Dashboard Node List Page



CLIMATE MONITOR

Garden Sensor Climate Data

Back

Recent Sensor Data

Date received: 12:40 24/11/19

Humidity
43%

Temperature
24°C

Pressure
900mb

Wind
4.4 knots

Wind Direction
South East

Historical Sensor Data

Historical Date Range

24/11/2019 to 26/11/2019

Temperature

Humidity

Figure 13. Wireframe Mobile Dashboard Node Page

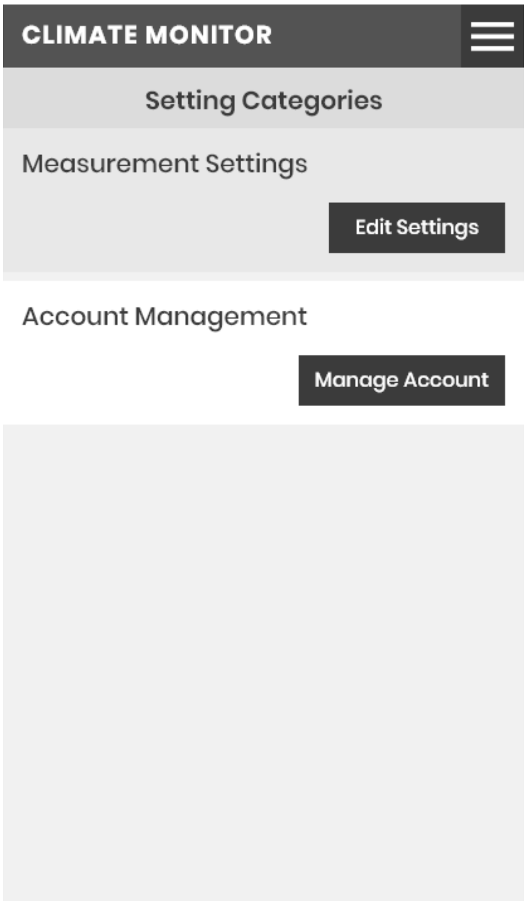
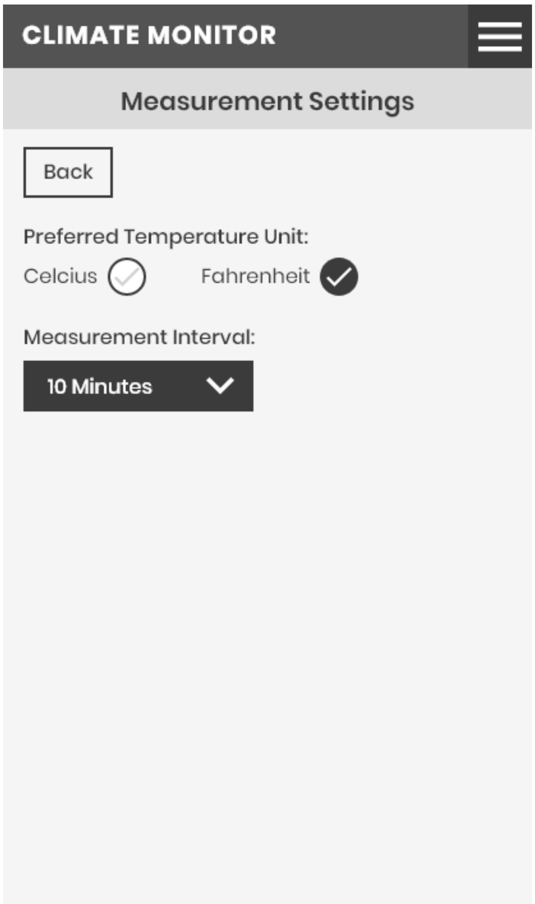


Figure 14. Wireframe Mobile Settings Category List Page



Preferred Temperature Unit:
Celcius ○ Fahrenheit ✓

Measurement Interval:
10 Minutes

Figure 15. Wireframe Mobile Settings Category Page

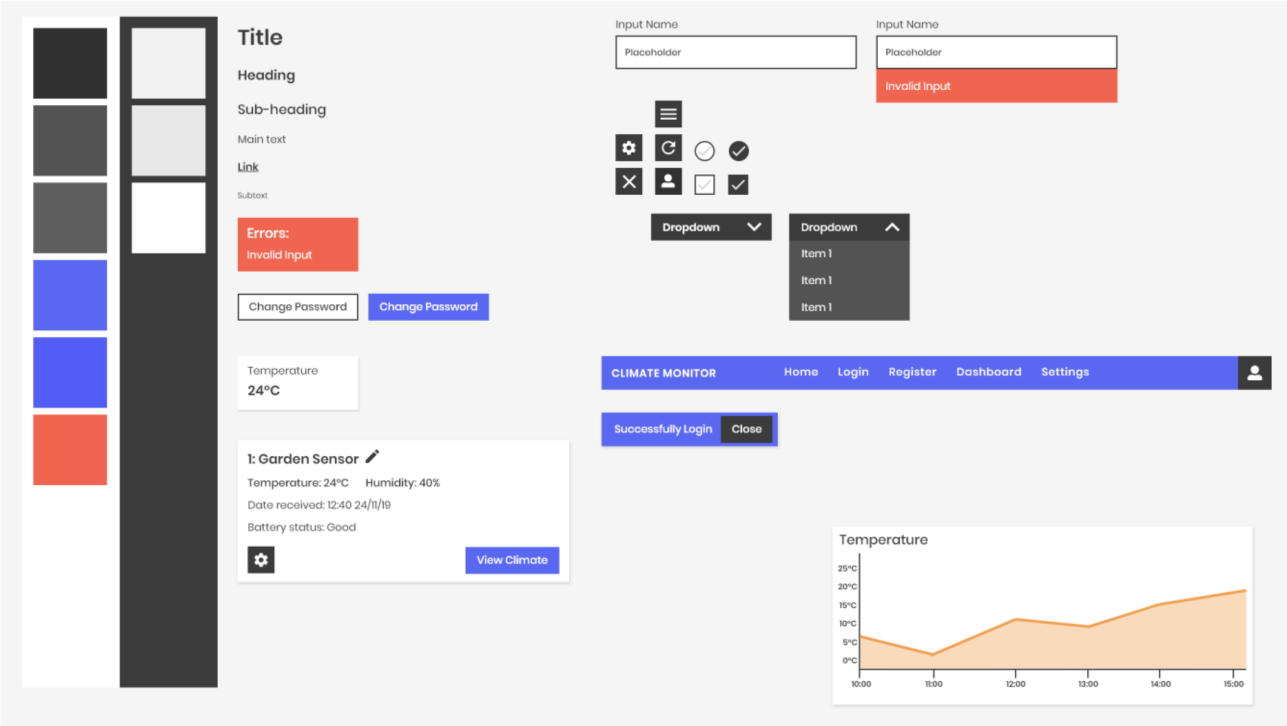
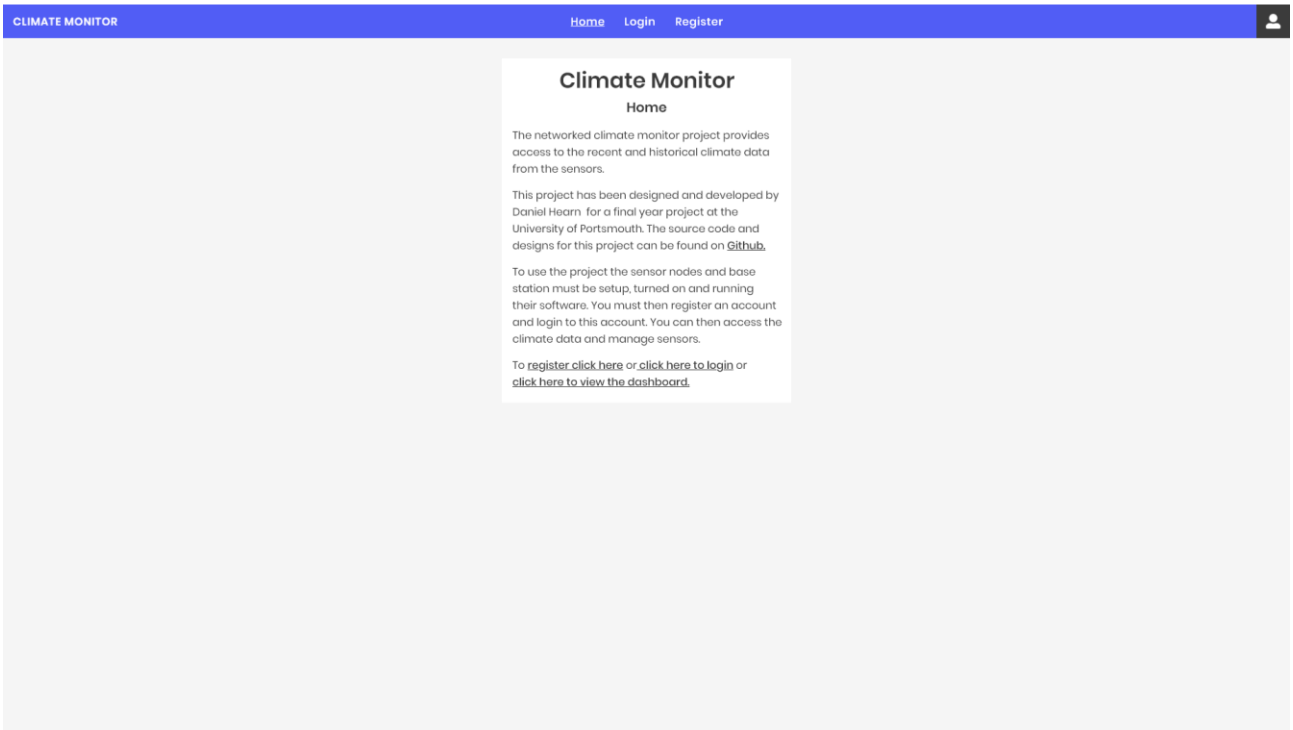


Figure 16. Final Stylesheet



Climate Monitor

Home

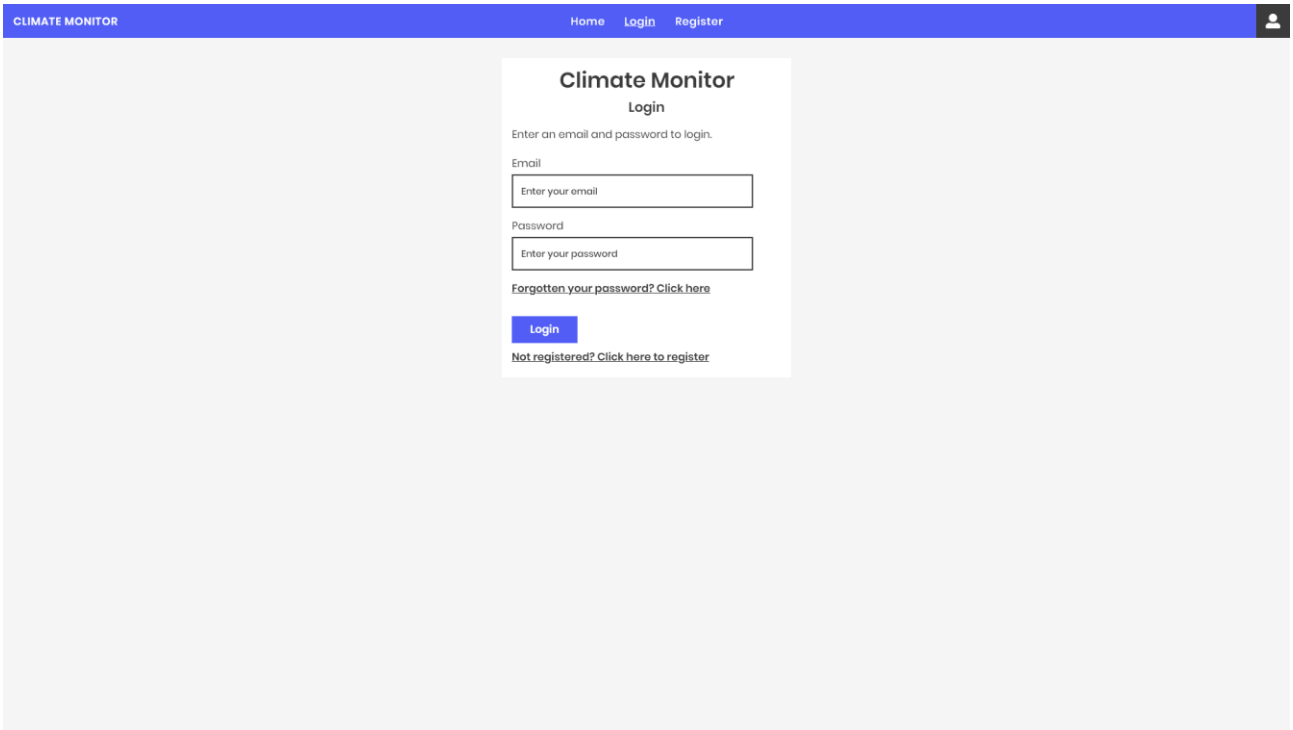
The networked climate monitor project provides access to the recent and historical climate data from the sensors.

This project has been designed and developed by Daniel Hearn for a final year project at the University of Portsmouth. The source code and designs for this project can be found on Github.

To use the project the sensor nodes and base station must be setup, turned on and running their software. You must then register an account and login to this account. You can then access the climate data and manage sensors.

To register click here or click here to login or click here to view the dashboard.

Figure 17. Final Desktop Home Page



CLIMATE MONITOR Home Login Register

Climate Monitor

Login

Enter an email and password to login.

Email

Enter your email

Password

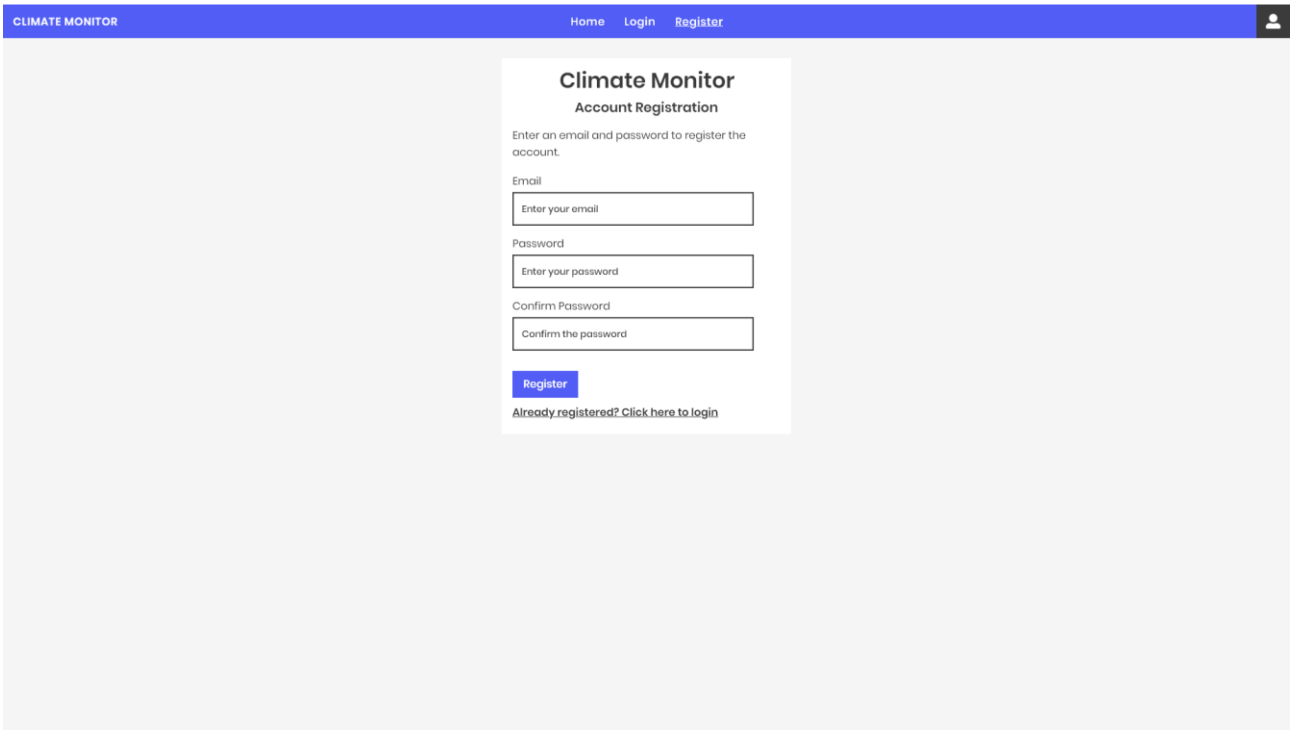
Enter your password

Forgotten your password? Click here

Login

Not registered? Click here to register

Figure 18. Final Desktop Login Page



CLIMATE MONITOR Home Login Register

Climate Monitor

Account Registration

Enter an email and password to register the account.

Email

Enter your email

Password

Enter your password

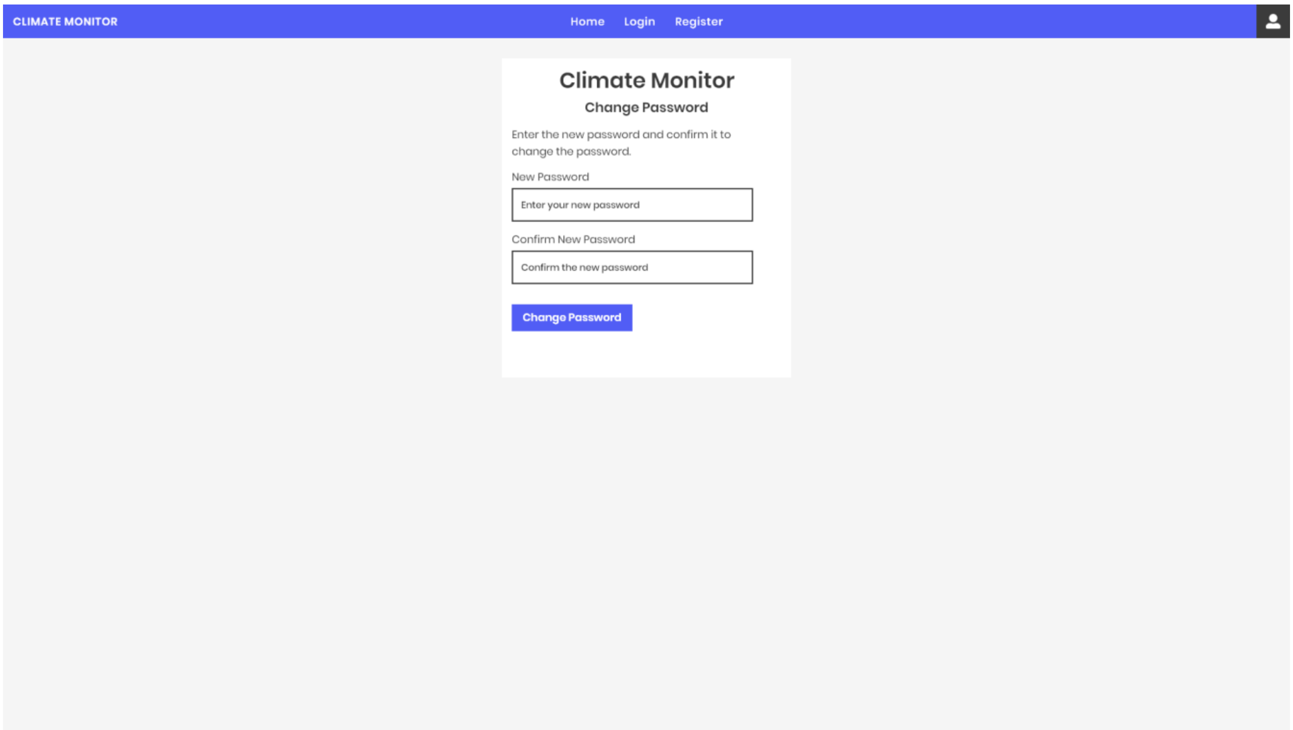
Confirm Password

Confirm the password

Register

Already registered? Click here to login

Figure 19. Final Desktop Registration Page



CLIMATE MONITOR

Home Login Register

Climate Monitor

Change Password

Enter the new password and confirm it to change the password.

New Password

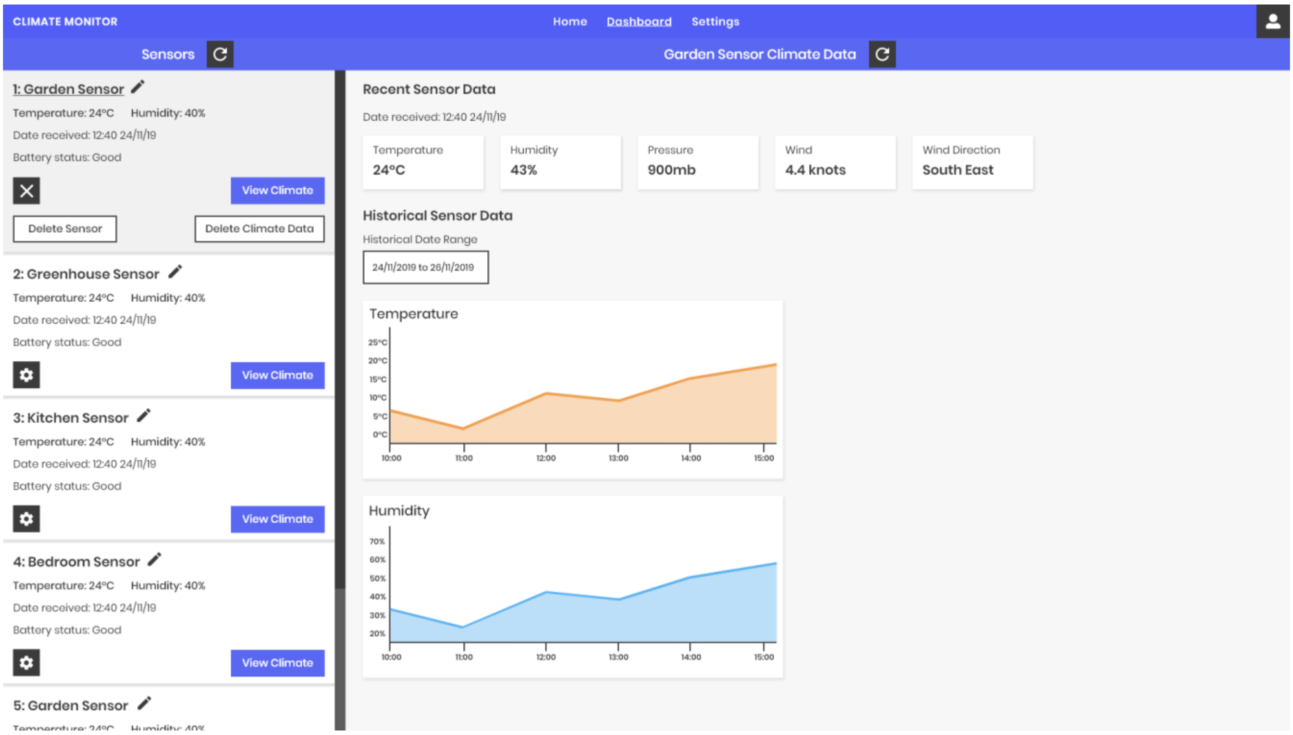
Enter your new password

Confirm New Password

Confirm the new password

Change Password

Figure 20. Final Desktop Change Password Page



CLIMATE MONITOR

Home Dashboard Settings

Sensors

Garden Sensor Climate Data

1: Garden Sensor

Temperature: 24°C Humidity: 40%

Date received: 12:40 24/11/19

Battery status: Good

View Climate

Delete Sensor

Delete Climate Data

2: Greenhouse Sensor

Temperature: 24°C Humidity: 40%

Date received: 12:40 24/11/19

Battery status: Good

View Climate

3: Kitchen Sensor

Temperature: 24°C Humidity: 40%

Date received: 12:40 24/11/19

Battery status: Good

View Climate

4: Bedroom Sensor

Temperature: 24°C Humidity: 40%

Date received: 12:40 24/11/19

Battery status: Good

View Climate

5: Garden Sensor

Temperature: 24°C Humidity: 40%

Recent Sensor Data

Date received: 12:40 24/11/19

Temperature	Humidity	Pressure	Wind	Wind Direction
24°C	43%	900mb	4.4 knots	South East

Historical Sensor Data

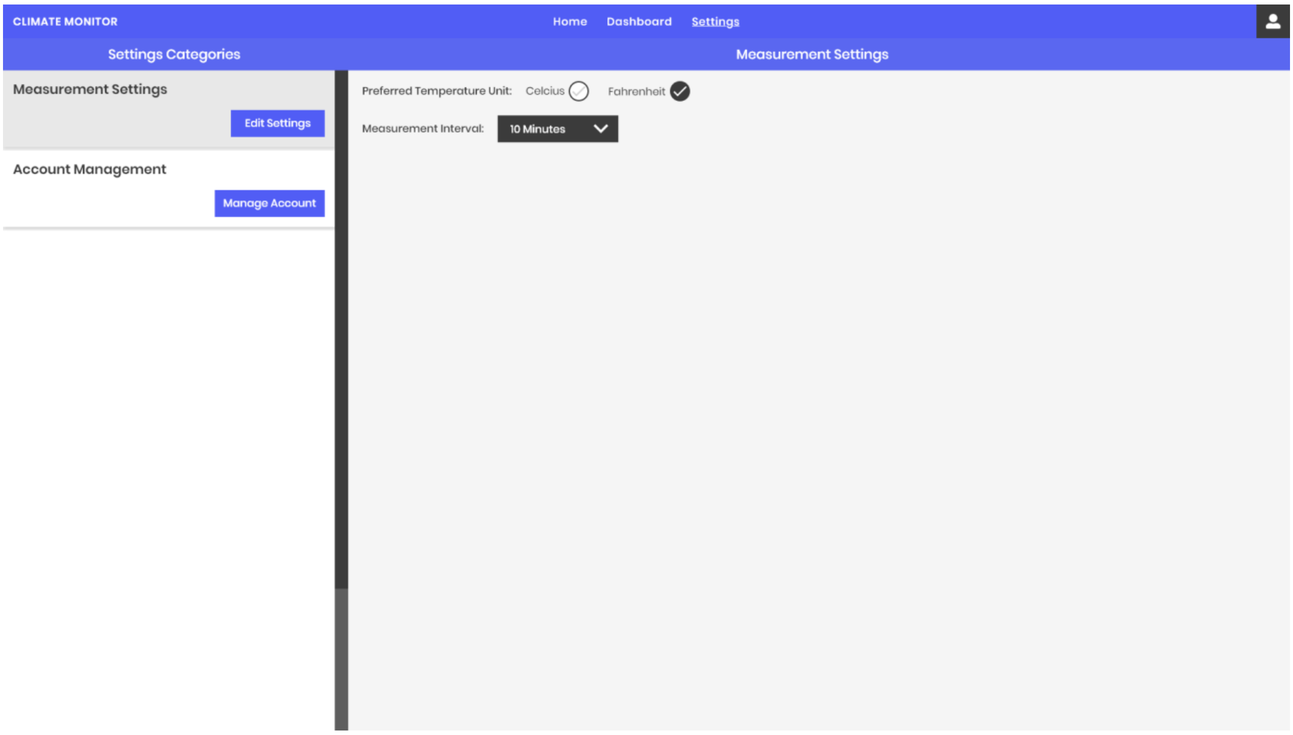
Historical Date Range

24/11/2019 to 26/11/2019

Temperature

Humidity

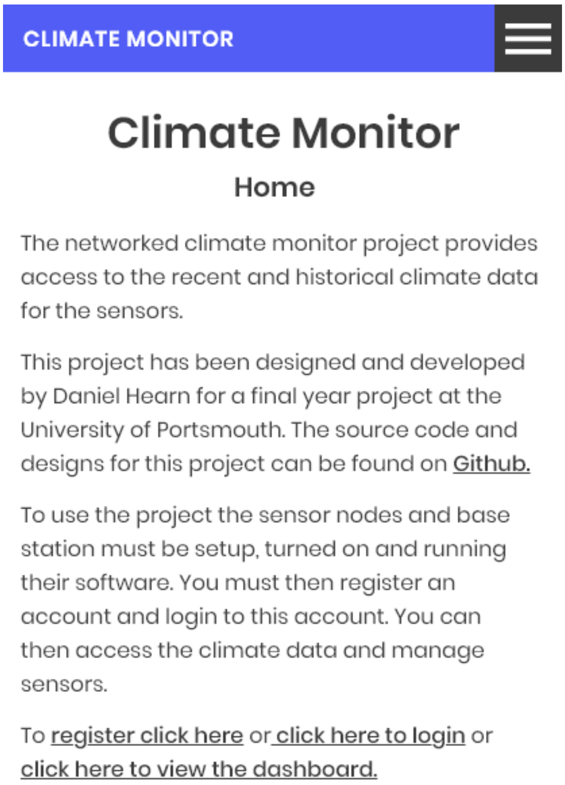
Figure 21. Final Desktop Dashboard Page



Preferred Temperature Unit: Celcius ○ Fahrenheit ✓

Measurement Interval: 10 Minutes

Figure 22. Final Desktop Settings Page



Climate Monitor

Home

The networked climate monitor project provides access to the recent and historical climate data for the sensors.

This project has been designed and developed by Daniel Hearn for a final year project at the University of Portsmouth. The source code and designs for this project can be found on Github.

To use the project the sensor nodes and base station must be setup, turned on and running their software. You must then register an account and login to this account. You can then access the climate data and manage sensors.

To register click here or click here to login or click here to view the dashboard.

Figure 23. Final Mobile Home Page

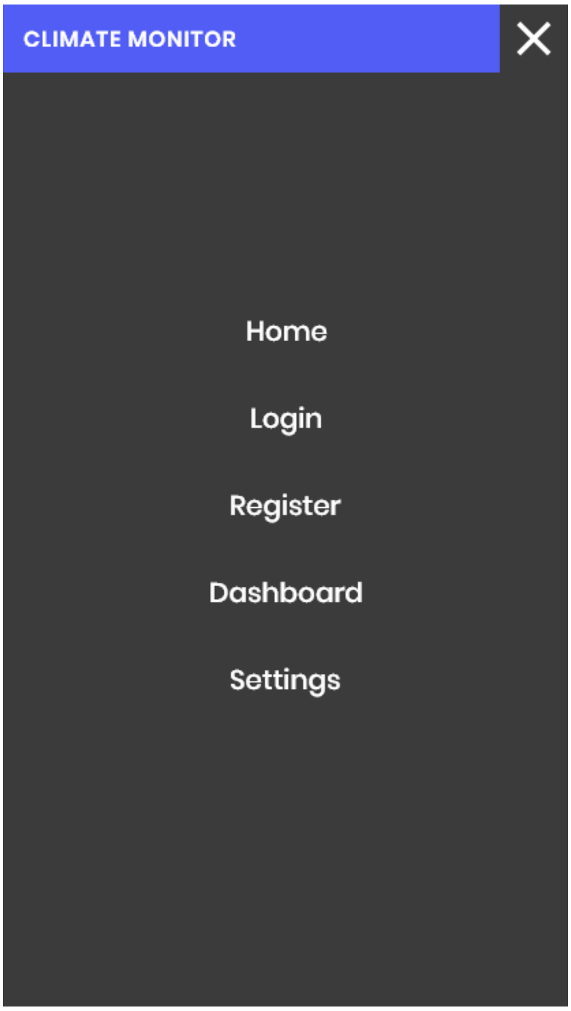
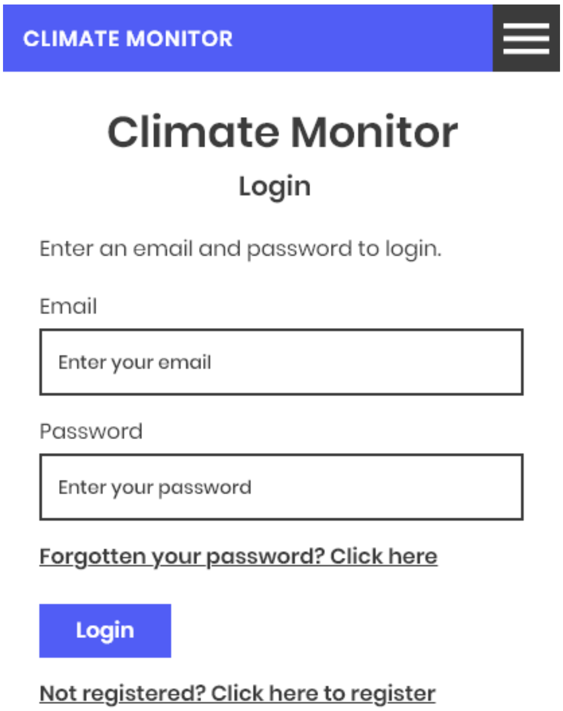


Figure 24. Final Mobile Menu



CLIMATE MONITOR

Climate Monitor

Login

Enter an email and password to login.

Email

Enter your email

Password

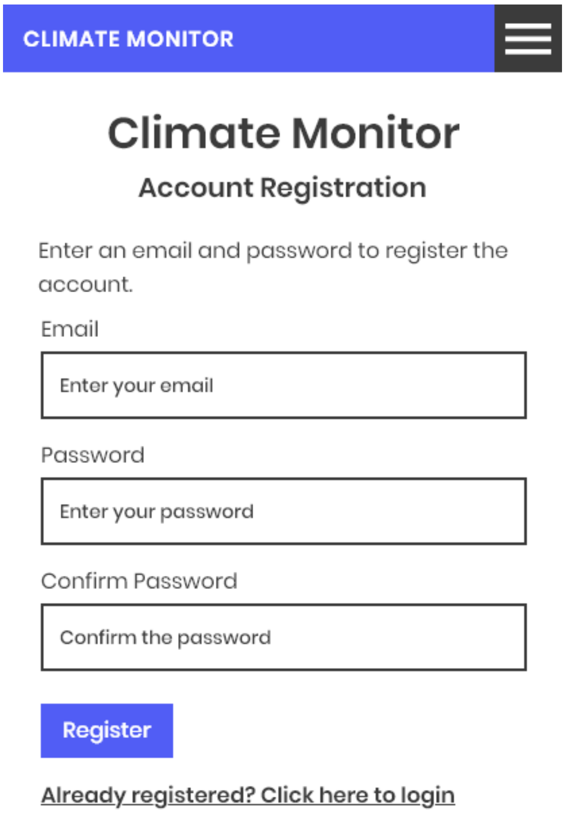
Enter your password

Forgotten your password? Click here

Login

Not registered? Click here to register

Figure 25. Final Mobile Login Page



Climate Monitor

Account Registration

Enter an email and password to register the account.

Email

Enter your email

Password

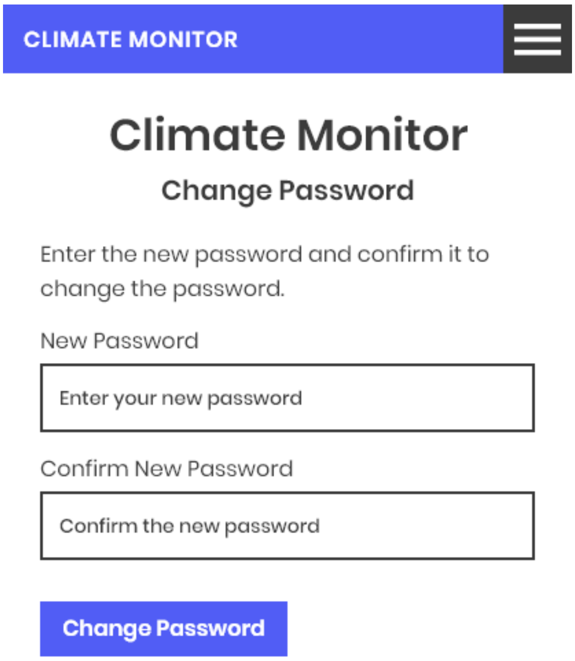
Enter your password

Confirm Password

Confirm the password

Already registered? Click here to login

Figure 26. Final Mobile Registration Page



Climate Monitor

Change Password

Enter the new password and confirm it to change the password.

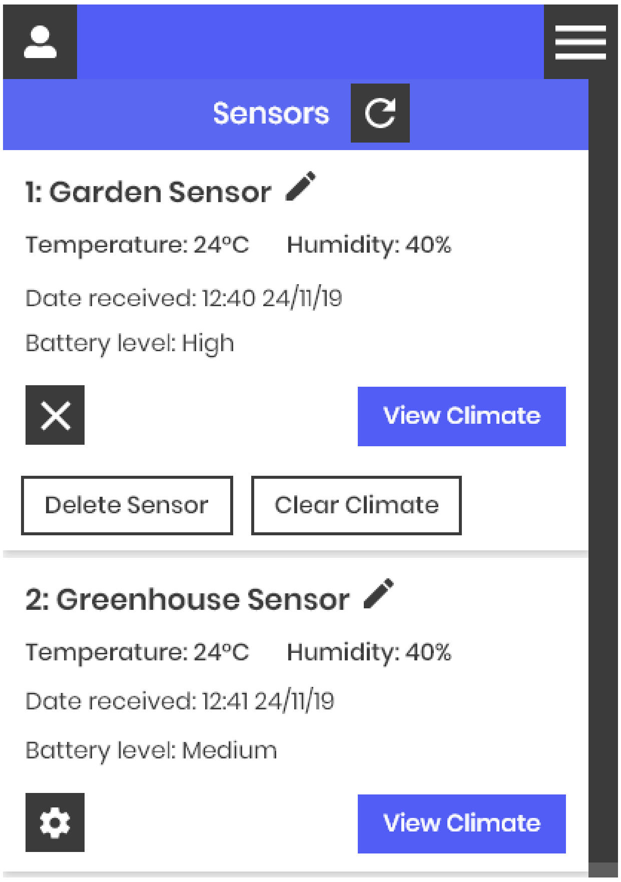
New Password

Enter your new password

Confirm New Password

Confirm the new password

Figure 27. Final Mobile Change Password Page



1: Garden Sensor

Temperature: 24°C Humidity: 40%

Date received: 12:40 24/11/19

Battery level: High

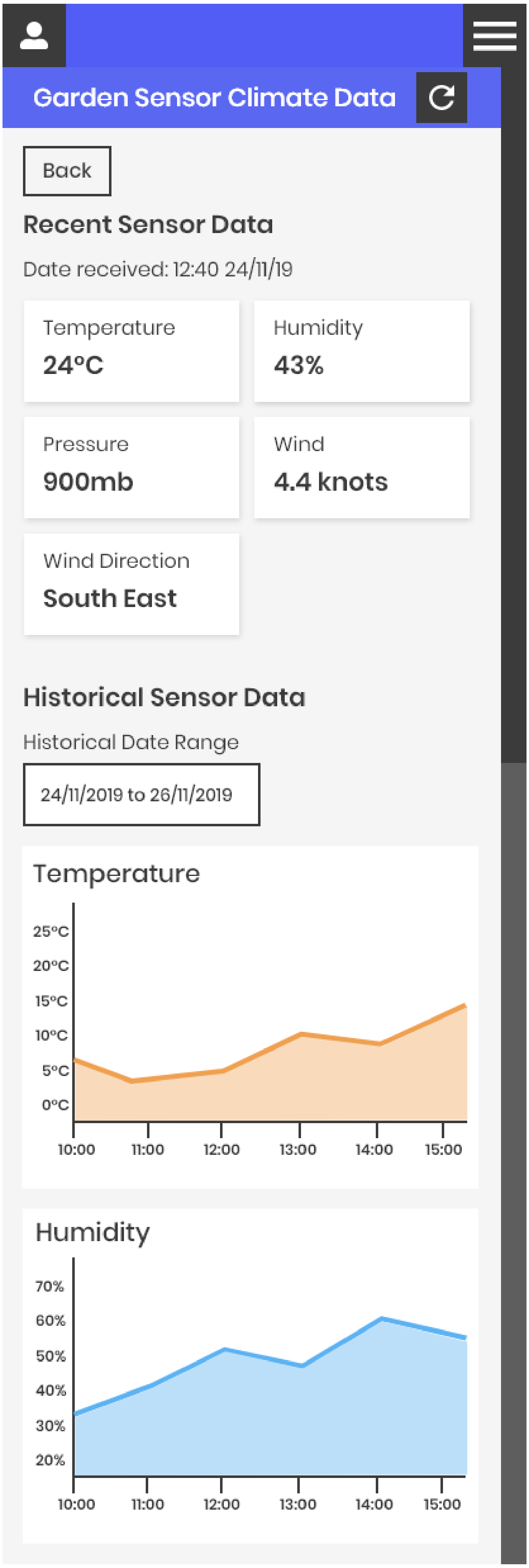
2: Greenhouse Sensor

Temperature: 24°C Humidity: 40%

Date received: 12:41 24/11/19

Battery level: Medium

Figure 28. Final Mobile Sensor Node List Page



Garden Sensor Climate Data

Back

Recent Sensor Data

Date received: 12:40 24/11/19

Temperature
24°C

Humidity
43%

Pressure
900mb

Wind
4.4 knots

Wind Direction
South East

Historical Sensor Data

Historical Date Range

24/11/2019 to 26/11/2019

Temperature

Humidity

Figure 29 .Final Mobile Sensor Node Page

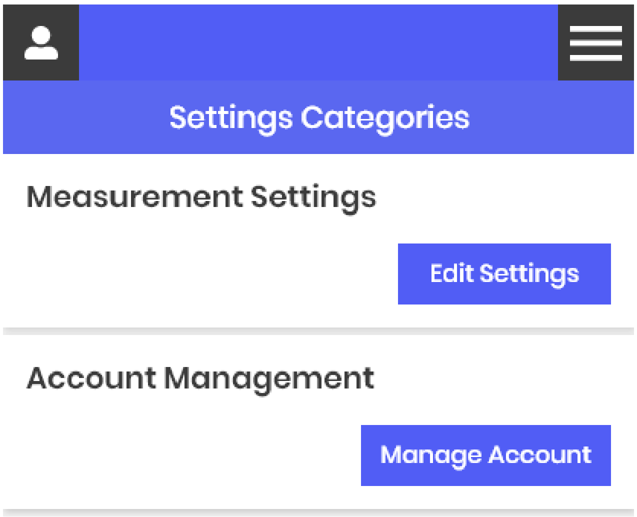
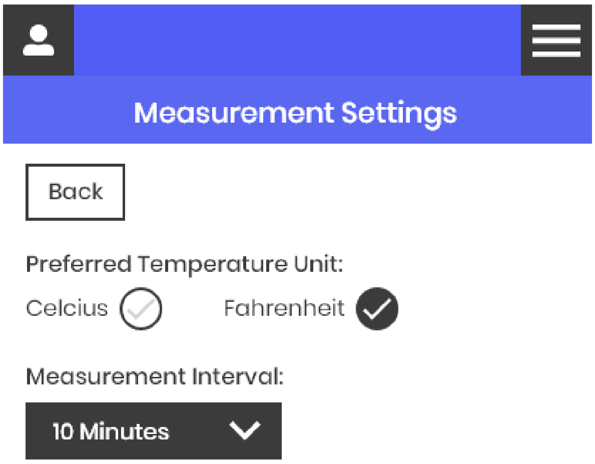


Figure 30. Final Mobile Category List Page



Measurement Settings

Preferred Temperature Unit:

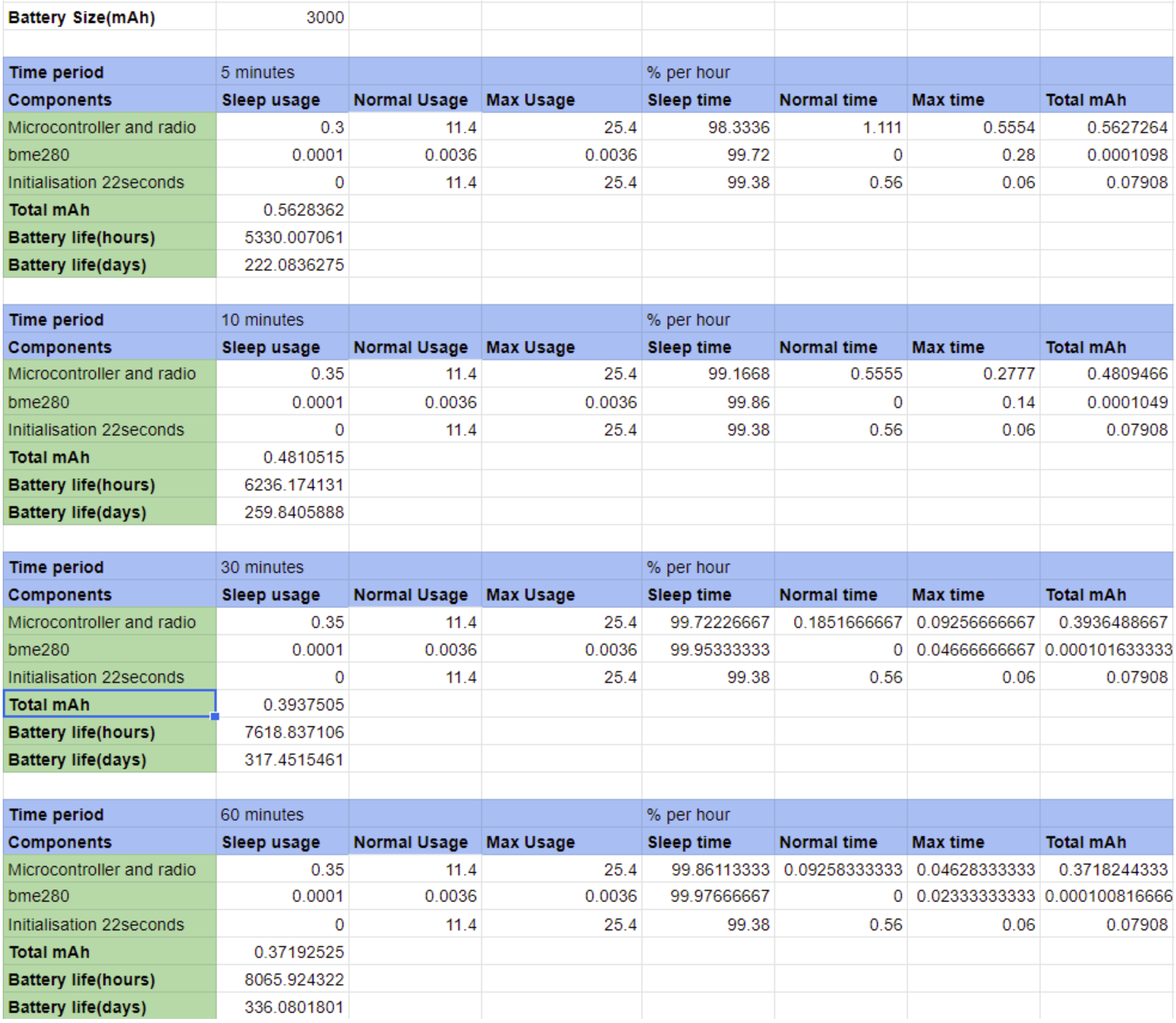
Celcius ○ Fahrenheit ✓

Measurement Interval:

10 Minutes

Figure 31. Final Mobile Settings Category Page

11.8 Appendix H: Battery Life Calculation Spreadsheet



Battery Size(mAh)	3000						

Time period	5 minutes			% per hour			
Components	Sleep usage	Normal Usage	Max Usage	Sleep time	Normal time	Max time	Total mAh
Microcontroller and radio	0.3	11.4	25.4	98.3336	1.111	0.5554	0.5627264
bme280	0.0001	0.0036	0.0036	99.72	0	0.28	0.0001098
Initialisation 22seconds	0	11.4	25.4	99.38	0.56	0.06	0.07908
Total mAh	0.5628362						
Battery life(hours)	5330.007061						
Battery life(days)	222.0836275						

Time period	10 minutes			% per hour			
Components	Sleep usage	Normal Usage	Max Usage	Sleep time	Normal time	Max time	Total mAh
Microcontroller and radio	0.35	11.4	25.4	99.1668	0.5555	0.2777	0.4809466
bme280	0.0001	0.0036	0.0036	99.86	0	0.14	0.0001049
Initialisation 22seconds	0	11.4	25.4	99.38	0.56	0.06	0.07908
Total mAh	0.4810515						
Battery life(hours)	6236.174131						
Battery life(days)	259.8405888						

Time period	30 minutes			% per hour			
Components	Sleep usage	Normal Usage	Max Usage	Sleep time	Normal time	Max time	Total mAh
Microcontroller and radio	0.35	11.4	25.4	99.72226667	0.1851666667	0.09256666667	0.3936488667
bme280	0.0001	0.0036	0.0036	99.95333333	0	0.04666666667	0.000101633333
Initialisation 22seconds	0	11.4	25.4	99.38	0.56	0.06	0.07908
Total mAh	0.3937505						
Battery life(hours)	7618.837106						
Battery life(days)	317.4515461						

Time period	60 minutes			% per hour			
Components	Sleep usage	Normal Usage	Max Usage	Sleep time	Normal time	Max time	Total mAh
Microcontroller and radio	0.35	11.4	25.4	99.86113333	0.09258333333	0.04628333333	0.3718244333
bme280	0.0001	0.0036	0.0036	99.97666667	0	0.02333333333	0.000100816666
Initialisation 22seconds	0	11.4	25.4	99.38	0.56	0.06	0.07908
Total mAh	0.37192525						
Battery life(hours)	8065.924322						
Battery life(days)	336.0801801						

11.9 Appendix I: Test Plan and Results

Name	Description	Success
Start up node that hasn’t been assigned a node ID	Whether a node will start up and attempt initialisation with the base station if it hasn’t been able to retrieve a stored node ID from memory	Yes
Start up node that has been assigned a node ID	Whether a node will start up and attempt initialisation with the base station if it has been able to retrieve a stored node ID from memory.	Yes
Send climate data	Whether the node sends valid climate data after sleeping for a time period received during initialisation from the base station	Yes
Initialisation of node	Whether the node sends the initialisation request and successfully receives the initialisation data from the base station after the node has been powered up.	Yes
Reinitialisation of node	Whether the node will receive the reinitialisation request after the base station reboots and the node has sent climate data to the base station.	Yes
Records temperature	Whether the node can record valid temperature data from the surrounding environment as specified in the functional requirements.	Yes
Records humidity	Whether the node can record valid humidity data from the surrounding environment as specified in the functional requirements.	Yes
Battery Power	Whether the node is battery powered.	Yes
Usb power	Whether the node is powered by usb.	Yes
Battery charging	Whether the node’s battery can be charged up instead of being replaced.	Yes
1m	Whether the node can send and receive packets with the base station at a distance of 1 metre	Yes
5m	Whether the node can send and receive packets with the base station at a distance of 5 metres	Yes
10m	Whether the node can send and receive packets with the base station at a distance of 10 metres	Yes
20m	Whether the node can send and receive packets with the base station at a distance of 20 metres	Yes
25m	Whether the node can send and receive packets with the base station at a distance of 25 metres	Yes

50m	Whether the node can send and receive packets with the base station at a distance of 50 metres	Yes
75m	Whether the node can send and receive packets with the base station at a distance of 75 metres	No
100m	Whether the node can send and receive packets with the base station at a distance of 100 metres	No
Wall penetration	Whether the node can send and receive packets with the base station through a single wall.	Yes
Max range	Max range of node communication	56.5m

Table 1. Sensor node test plan

Name	Description	Success
Create a sensor object	Whether the base station creates a sensor object to represent connected sensor nodes within the base station.	Yes
Convert string of ascii to unicode	Whether the base station can convert a list of ascii characters into a string of unicode characters.	Yes
Convert sensor data type	Whether the base station can retrieve the full name of a sensor data type from a sensor data type character.	Yes
Get measurement unit from sensor data type	Whether the base station can retrieve the measurement unit from a full name of a sensor data type.	Yes
Process radio packet	Whether the base station can process a radio packet from a sensor node.	Yes
Process radio packet control section	Whether the base station can process the string control section of a packet into a dictionary representing the keys and values from the control section of the packet.	Yes
Process climate data packet	Whether the base station can process a climate data packet from a sensor node and successfully send the climate data to the API.	Yes
Process initialisation packet	Whether the base station can process an initialisation packet and send the node ID and time period interval back to the node, and if it creates a node for that node ID within the database.	Yes

Generate milliseconds for a time period	Whether the base station can calculate the number of milliseconds to the next send interval for a node based on it’s time period.	Yes
Get next available sensor ID	Whether the base station can retrieve the next available sensor ID based on the used sensor node IDs within the database.	Yes
Filter active sensors	Whether the base station can filter active sensors that have communicated recently with the base station.	Yes
Remove inactive sensors	Whether the base station removes inactive sensors that have not recently communicated with the base station.	Yes
Send climate data to the API	Whether the base station sends climate data to API after the base station receives climate data from a node.	Yes
Reinitialise node	Whether the base station re-initialises a node after receiving climate data from a node that is not being currently tracked by the base station.	Yes
Connect to mobile network	Whether the base station connects to the mobile WiFi network when the user specified WiFi network is not available.	Yes
Connect to user network	Whether the base station connects to the user specified WiFi network.	Yes
Connect to user network when mobile network is available	Whether the base station connects to the user specified WiFi network when the mobile WiFi network is also available.	Yes

Table 2. Base station test plan

Name	Description	Success
Success	Whether the API can achieve a successful login request with valid post data	Yes
Missing json data	Whether the API produces an error response when no post data is sent	Yes
Missing email	Whether the API produces an error response when the email field is missing from the post data	Yes
Missing password	Whether the API produces an error response when the password field is missing from the post data	Yes
Email isn’t a string	Whether the API produces ates an error response when the email isn't a string	Yes
Password isn’t a string	Whether the API produces an error response when the password isn't a string	Yes
Incorrect email and	Whether the API produces an error response when the email or password don't match the stored use credentials	Yes

password		

Table 3. Login endpoint test plan

Name	Description	Success
Success	Whether the API can achieve a successful access token logout request with valid post data.	Yes
Incorrect access token	Whether the API produces an error response when the access token is not valid.	Yes
Missing Auth	Whether the API produces an error response when the access token is missing.	Yes

Table 4. Access token logout endpoint test plan

Name	Description	Success
Success	Whether the API can achieve a successful refresh token logout request with valid post data.	Yes
Incorrect refresh token	Whether the API produces an error response when the access token is not valid.	Yes
Missing Auth	Whether the API produces an error response when the access token is missing.	Yes

Table 5. Refresh token logout endpoint test plan

Name	Description	Success
Success	Whether the API can achieve a successful access token refresh request with valid post data.	Yes
Incorrect refresh token	Whether the API produces an error response when the refresh token is not valid.	Yes

Table 6. Refresh token endpoint test plan

Name	Description	Success
Success	Whether the API can achieve a successful sensor creation request with valid post data.	Yes
Missing json data	Whether the API produces an error response when no post data is sent	Yes
Missing name	Whether the API produces an error response when the name field is missing from the post data.	Yes
Missing sensor_id	Whether the API produces an error response when the sensor_id field is missing from the post data.	Yes
Missing user_id	Whether the API produces an error response when the user_id field is missing from the post data.	Yes
Name isn’t string	Whether the API produces an error response when the name isn't a string.	Yes

Name is too long	Whether the API produces an error response when the name is longer than 40 characters.	Yes
sensor_id isn't integer	Whether the API produces an error response when the sensor_id isn't an integer.	Yes
user_id isn’t integer	Whether the API produces an error response when the user_id isn't a string.	Yes
Duplicate ID	Whether the API produces an error response when a sensor node within the database already has the same ID as the ID in the post data.	Yes
Invalid API key	Whether the API produces an error response when the api_key querystring value is invalid.	Yes
Missing API key	Whether the API produces an error response when the api_key querystring is missing from the request.	Yes

Table 7. Create sensor endpoint test plan

Name	Description	Success
Success no sensors	Whether the API can achieve a successful sensor retrieval request when no sensors are stored within the database.	Yes
Success multiple sensors with no climate data	Whether the API can achieve a successful sensor retrieval request when sensors are stored within the database but they haven’t got any stored climate data.	Yes
Success multiple sensors with climate data	Whether the API can achieve a successful sensor retrieval request when sensors are stored within the database and they have stored climate data.	Yes
Missing authentication token	Whether the API produces an error response when the access token is not valid.	Yes
Invalid authentication token	Whether the API produces an error response when the access token is missing.	Yes

Table 8. Get sensors endpoint test plan

Name	Description	Success
Success no sensors	Whether the API can achieve a successful sensors deletion request when no sensors are stored within the database.	Yes
Success sensors	Whether the API can achieve a successful sensors deletion request when sensors are stored within the database.	Yes
Missing authentication token	Whether the API produces an error response when the access token is not valid.	Yes
Invalid authentication token	Whether the API produces an error response when the access token is missing.	Yes

Table 9. Delete sensors endpoint test plan

Name	Description	Success

Success	Whether the API can achieve a successful sensor update request with valid post data.	Yes
Missing json data	Whether the API produces an error response when no post data is sent	Yes
Name isn't string	Whether the API produces an error response when the name field is missing from the post data.	Yes
Name is too long	Whether the API produces an error response when the name is longer than 40 characters.	Yes
sensor_id isn't integer	Whether the API produces an error response when the sensor_id isn't an integer.	Yes
Sensor doesn't exist	Whether the API produces an error response when a sensor with that ID doesn't exist within the database.	Yes
Missing authentication token	Whether the API produces an error response when the access token is not valid.	Yes
Invalid authentication token	Whether the API produces an error response when the access token is missing.	Yes

Table 10. Update sensor endpoint test plan

Name	Description	Success
Success	Whether the API can achieve a successful sensor retrieval request.	Yes
Sensor doesn't exist	Whether the API produces an error response when a sensor with that ID doesn't exist within the database.	Yes
Missing authentication token	Whether the API produces an error response when the access token is not valid.	Yes
Invalid authentication token	Whether the API produces an error response when the access token is missing.	Yes

Table 11. Get sensor endpoint test plan

Name	Description	Success
Success	Whether the API can achieve a successful sensor deletion request.	Yes
Sensor doesn't exist	Whether the API produces an error response when a sensor with that ID doesn't exist within the database.	Yes
Missing authentication token	Whether the API produces an error response when the access token is not valid.	Yes
Invalid authentication token	Whether the API produces an error response when the access token is missing.	Yes

Table 12. Delete sensor endpoint test plan

Name	Description	Success
Success	Whether the API can achieve a successful climate data creation request with valid post data.	Yes

Missing json data	Whether the API produces an error response when no post data is sent.	Yes
Sensor doesn’t exist	Whether the API produces an error response when a sensor with that ID doesn’t exist within the database.	Yes
Missing climate_data	Whether the API produces an error response when the climate_data field is missing from the post data.	Yes
Missing unit	Whether the API produces an error response when the unit field is missing from the post data.	Yes
Missing value	Whether the API produces an error response when the value field is missing from the post data.	Yes
Missing type	Whether the API produces an error response when the type field is missing from the post data.	Yes
Missing battery_voltage	Whether the API produces an error response when the battery_voltage field is missing from the post data.	Yes
climate_data isn’t a list	Whether the API produces an error response when the climate_data isn't a list.	Yes
unit isn’t a string	Whether the API produces an error response when the unit isn't a list.	Yes
value isn’t a string	Whether the API produces an error response when the type isn't a list.	Yes
type isn’t a string	Whether the API produces an error response when the value isn't a list.	Yes
battery_voltage isn’t a string	Whether the API produces an error response when the battery_voltage isn't a list.	Yes
Incorrect date	Whether the API produces an error response when the date field is not a valid date.	Yes
Invalid API key	Whether the API produces an error response when the api_key querystring value is invalid.	Yes
Missing API key	Whether the API produces an error response when the api_key querystring is missing from the request.	Yes

Table 13. Create climate data endpoint test plan

Name	Description	Success
Success default	Whether the API can achieve a successful climate data retrieval request when no quantity or date range is provided.	Yes
Success quantity 20	Whether the API can achieve a successful climate data retrieval request when a quantity of 20 is provided.	Yes
Success range 1 day	Whether the API can achieve a successful climate data retrieval request when a date range of 1 day is provided.	Yes
Success range 2 day	Whether the API can achieve a successful climate data retrieval request when a date range of 2 days is provided.	Yes
Success range 7 days	Whether the API can achieve a successful climate data retrieval request when a date range of 7 days is provided.	Yes
Success range 1	Whether the API can achieve a successful climate data retrieval request	Yes

month	when a date range of 1 month is provided.	
Quantity isn’t an integer	Whether the API produces an error response when the quantity isn’t an integer.	Yes
Quantity is invalid	Whether the API produces an error response when the quantity is outside of the valid 1 to 50 quantity range.	Yes
Sensor doesn’t exist	Whether the API produces an error response when a sensor with that ID doesn’t exist within the database.	Yes
Date range is invalid	Whether the API produces an error response when the date range isn’t a valid range where the range_start is after the range_end or either of the range values aren’t valid dates.	Yes
Missing authentication token	Whether the API produces an error response when the access token is not valid.	Yes
Invalid authentication token	Whether the API produces an error response when the access token is missing.	Yes

Table 14. Get climate data endpoint test plan

Name	Description	Success
Success	Whether the API can achieve a successful climate data deletion request.	Yes
Sensor doesn’t exist	Whether the API produces an error response when a sensor with that ID doesn’t exist within the database.	Yes
Missing authentication token	Whether the API produces an error response when the access token is not valid.	Yes
Invalid authentication token	Whether the API produces an error response when the access token is missing.	Yes

Table 15. Delete climate data endpoint test plan

Name	Description	Success
Success	Whether the API can achieve a successful account creation request.	Yes
Missing json data	Whether the API produces an error response when no post data is sent.	Yes
Missing email	Whether the API produces an error response when the email field is missing from the post data.	Yes
Missing password	Whether the API produces an error response when the password field is missing from the post data.	Yes
Email isn’t a string	Whether the API produces an error response when the email isn't a valid email address.	Yes
Password isn’t a string	Whether the API produces an error response when the password isn’t a string.	Yes
Password isn’t	Whether the API produces an error response when the password isn’t	Yes

between 8 to 40 characters long	between 8 to 40 characters long.	

Table 16. Create account endpoint test plan

Name	Description	Success
Success	Whether the API can achieve a successful account update request.	Yes
Email isn’t a string	Whether the API produces an error response when the email isn't a valid email address.	Yes
Password isn’t a string	Whether the API produces an error response when the password isn't a string.	Yes
Password isn’t between 8 to 40 characters long	Whether the API produces an error response when the password isn’t between 8 to 40 characters long.	Yes
Settings isn’t a string	Whether the API produces an error response when the settings isn't a string.	Yes
Account not created	Whether the API produces an error response when the account has not been created.	Yes
Missing Authentication Token	Whether the API produces an error response when the access token is not valid.	Yes
Invalid Authentication Token	Whether the API produces an error response when the access token is missing.	Yes

Table 17. Update account endpoint test plan

Name	Description	Success
Success	Whether the API can achieve a successful account retrieval request.	Yes
Account not created	Whether the API produces an error response when the account has not been created.	Yes
Missing Authentication Token	Whether the API produces an error response when the access token is not valid.	Yes
Invalid Authentication Token	Whether the API produces an error response when the access token is missing.	Yes

Table 18. Get account endpoint test plan

Name	Description	Success
Success	Whether the API can achieve a successful password change request.	Yes
Missing JSON data	Whether the API produces an error response when no post data is sent.	Yes
Account not created	Whether the API produces an error response when the account has not been created.	Yes

Missing password	Whether the API produces an error response when the password field is missing from the post data.	Yes
Missing reset_token	Whether the API produces an error response when the reset_token field is missing from the post data.	Yes
Wrong reset_token length	Whether the API produces an error response when the reset_token field is not 20 characters long.	Yes
Password isn’t a string	Whether the API produces an error response when the password isn't a string.	Yes
Password isn’t between 8 to 40 characters long	Whether the API produces an error response when the password isn’t between 8 to 40 characters long.	Yes
Invalid reset token	Whether the API produces an error response when the reset_token does not match the reset token stored within the database.	Yes

Table 19. Change password endpoint test plan

Name	Description	Success
Success no sensors	Whether the API can achieve a successful next available sensor ID request when no sensors have been stored in the database.	Yes
Success with sensors	Whether the API can achieve a successful next available sensor ID request when multiple sensors have been stored in the database.	Yes
Invalid API key	Whether the API produces an error response when the api_key querystring value is invalid.	Yes
Missing API key	Whether the API produces an error response when the api_key querystring is missing from the request.	Yes

Table 20. Get next available sensor ID endpoint test plan

Name	Description	Success
Success	Whether the API will periodically remove climate data that is older than 6 months from the database.	Yes

Table 21. Old climate data removal test plan

Name	Description	Success
End point doesn’t exist	Test that an error status is always given for any endpoint url that hasn’t been defined in the API.	Yes

Table 22. Nonexistent endpoint test plan

Name	Description	Success
Serve HTML	Whether the web-server will serve the HTML page of the website when a request to the root of the web-server’s IP address is made.	Yes
Serve JavaScript	Whether the web-server will serve a JavaScript file used by the website when the JavaScript file’s name and folder	Yes

Table 21. Web-server unit test plan

Name	Description	Success
Create	Whether an instance of UserModel is created when valid data is input.	Yes
save	Whether an instance of UserModel is saved into the database using the save class method.	Yes
delete	Whether an instance of UserModel is deleted from the database using the delete class method.	Yes
find_by_email	Whether an instance of UserModel is retrieved from the database by inputting an email that matches the email address of a user within the database.	Yes
return_first	Whether the first instance of UserModel is retrieved from the database by using the return_first class method.	Yes
verify_hash	Whether an unhashed password can be correctly compared against the hashed password of a user.	Yes

Table 23. Database UserModel test plan

Name	Description	Success
Create	Whether an instance of RevokedTokenModel is created when valid data is input.	Yes
save	Whether an instance of RevokedTokenModel is saved into the database using the save class method.	Yes
delete	Whether an instance of RevokedTokenModel is deleted from the database using the delete class method.	Yes
is_jti_blacklisted	Whether an input JWT token already exists within the database.	Yes

Table 24. Database RevokedTokenModel test plan

Name	Description	Success
Create	Whether an instance of SensorModel is created when valid data is input.	Yes
save	Whether an instance of SensorModel is saved into the database using the save class method.	Yes
delete	Whether an instance of SensorModel is deleted from the database using the delete class method.	Yes
delete_all	Whether all instances of SensorModel are deleted from the database using the delete_all class method.	Yes

Table 25. Database SensorModel test plan

Name	Description	Success
Create	Whether an instance of ClimateModel is created when valid data is input.	Yes
interval	Whether the ID of an instance of ClimateModel is reduced down to the input integer.	Yes
save	Whether an instance of ClimateModel is saved into the database using the	Yes

	save class method.	
delete	Whether an instance of ClimateModel is deleted from the database using the delete class method.	Yes

Table 26. Database ClimateModel test plan

Name	Description	Success
Create	Whether an instance of SensorDataModel is created when valid data is input.	Yes
save	Whether an instance of SensorDataModel is saved into the database using the save class method.	Yes
delete	Whether an instance of SensorDataModel is deleted from the database using the delete class method.	Yes

Table 27. Database SensorDataModel test plan

Name	Description	Success
Header Slot	Whether the component renders content input into the header slot.	Yes
Content Slot	Whether the component renders content input into the content slot.	Yes

Table 28. Website MainPanel component test plan

Name	Description	Success
Header Slot	Whether the component renders content input into the header slot.	Yes
Content Slot	Whether the component renders content input into the content slot.	Yes

Table 29. Website SidePanel component test plan

Name	Description	Success
Title prop	Whether the component renders the correct title text input via the title prop.	Yes
Errors prop	Whether the component renders the error text input via the errors prop.	Yes

Table 30. Website ErrorList component test plan

Name	Description	Success
hierarchyLevel	Whether the component renders with the correct styling based on the hierarchyLevel input via the hierarchyLevel prop..	Yes
text	Whether the component renders the text input via the text prop.	Yes
isIcon	Whether the component renders an icon if the isIcon prop is input.	Yes

Table 31. Website vButton component test plan

Name	Description	Success
chartData	Whether the component renders the correct chart data input via the chartData prop.	Yes
options	Whether the component renders the correct options input via the options	Yes

	prop.	

Table 32. Website Chart component test plan

Name	Description	Success
loggedIn visibility	Whether the component renders the content that requires the user to be logged in once the loggedIn prop is input.	Yes
Mobile only links	Whether the component renders the content that requires the user to be on a mobile device once the mobile prop is input.	Yes
Mobile menu toggle	Whether the component correctly toggles the mobile menu once the mobile menu button is clicked.	Yes

Table 33. Website Nav component test plan

Name	Description	Success
recentClimateData	Whether the component renders the climate data input via the recentClimateData prop.	Yes
temperatureUnit	Whether the component renders the temperature data in the correct measurement unit that was input via the temperatureUnit prop.	Yes

Table 34. Website RecentClimateData component test plan

Name	Description	Success
No email error	Whether the component renders the no email error when no email is input into the form.	Yes
No password error	Whether the component renders the no password error when no password is input into the form.	Yes
Invalid password error	Whether the component renders the invalid password error when the login API request fails once the form is submitted.	Yes
Success	Whether the component renders the success status when the login is successful.	Yes

Table 35. Website LoginForm component test plan

Name	Description	Success
No email error	Whether the component renders the no email error when no email is input into the form.	Yes
No password error	Whether the component renders the no password error when no password is input into the form.	Yes
Mismatch password error	Whether the component renders the no mismatch password error when the password doesn’t match the confirm password value.	Yes
Success	Whether the component renders the success status when the registration is successful.	Yes

Table 35. Website RegisterForm component test plan

Name	Description	Success
No reset token error	Whether the component renders the no email error when no email is input into the form.	Yes
No password error	Whether the component renders the no password error when no password is input into the form.	Yes
Mismatch password error	Whether the component renders the no mismatch password error when the password doesn't match the confirm password value.	Yes
Success input token	Whether the component renders the success status when the password token is manually input and the password reset is successful.	Yes
Success loaded token	Whether the component renders the success status when the password token is automatically input and the password reset is successful.	Yes

Table 36. Website PasswordResetForm component test plan

Name	Description	Success
loggedIn visibility	Whether the component renders the content that requires the user to be logged in once the loggedIn prop is input.	Yes
toggleMenu	Whether the component correctly toggles the mobile menu once the mobile menu button is clicked.	Yes

Table 37. Website MobileMenu component test plan

Name	Description	Success
getStoredAccessToken	Whether the function correctly retrieves the JWT access token from the local storage of the user's browser.	Yes
getStoredRefreshToken	Whether the function correctly retrieves the JWT refresh token from the local storage of the user's browser.	Yes
setStoredAccessToken	Whether the function correctly stores the JWT access token into the local storage of the user's browser.	Yes
setStoredRefreshToken	Whether the function correctly stores the JWT refresh token into the local storage of the user's browser.	Yes

Table 38. Website Storage component test plan

Name	Description	Success
setMobile	Whether the function sets the mobile value in the vuex state.	Yes
setMobileMenu	Whether the function sets the mobile menu value in the vuex state.	Yes
setUser	Whether the function sets the user value in the vuex state.	Yes

Table 39. Website Store component test plan

Name	Description	Success
capitalise	Whether the function capitalises the first letter of the string.	Yes

convertTemperature	Whether the function converts the temperature value to the correct value depending on the input measurement unit.	Yes
formatClimateData	Whether the function formats the input climate data value into a correctly formatted string.	Yes
processErrors	Whether the function returns correctly formatted errors when raw errors are input.	Yes
getBatteryStatusFromVoltage	Whether the function returns the correct battery status text when the battery voltage is input.	Yes

Table 40. Website Helpers component test plan

Name	Description	Success
Snapshot	Whether the component's rendered HTML matches the stored snapshot.	Yes

Table 41. Website Home view test plan

Name	Description	Success
Snapshot	Whether the component's rendered HTML matches the stored snapshot.	Yes
No sensors	Whether the component renders the correct no sensor text when no sensors are stored.	Yes
Multiple sensors	Whether the component renders the correct no sensor text when no sensors are stored.	Yes
Change active sensor	Whether the component correctly changes the active sensor when sensor active buttons are clicked.	Yes
Refresh sensors	Whether the component refreshes the sensors once the refresh button is clicked.	Yes
Delete sensor	Whether the component sends a sensor deletion request to the API once a delete sensor button is clicked.	Yes
Delete climate data	Whether the component sends a sensor climate data deletion request to the API once a delete sensor climate data button is clicked.	Yes
Rename sensor	Whether the component sends a sensor rename request to the API once a sensor name text box value is changed.	Yes
Change historical data period	Whether the component	Yes

Table 42. Website Dashboard view test plan

Name	Description	Success
Snapshot	Whether the component's rendered HTML matches the stored snapshot.	Yes

Table 43. Website Login view test plan

Name	Description	Success

Snapshot	Whether the component’s rendered HTML matches the stored snapshot.	Yes
Form shown	Whether the component renders the registration form is the user is not registered.	Yes
Instructions shown	Whether the component renders the registration success information if the user is registered.	Yes

Table 44. Website Register view test plan

Name	Description	Success
Snapshot	Whether the component’s rendered HTML matches the stored snapshot.	Yes

Table 45. Website Forgot Password view test plan

Name	Description	Success
Snapshot	Whether the component’s rendered HTML matches the stored snapshot.	Yes
Change category	Whether the component changes the active category once a category selection button is clicked.	Yes
Change temperature unit	Whether the component changes the active temperature unit once a unit radio input is selected.	Yes
Reset token	Whether the component renders the user’s password reset token.	Yes
Change wifi settings	Whether the component changes the wifi settings once the save settings button is clicked.	Yes
Change measurement interval	Whether the component changes the active measurement interval once a dropdown option is selected.	Yes

Table 46. Website Settings view test plan

Name	Description	Success
Snapshot	Whether the component’s rendered HTML matches the stored snapshot.	Yes

Table 47. Website NotFound view test plan

Name	Description	Success
Access of website on base station IP address	Whether the website is accessible from a static IP address.	Yes
Desktop access	Whether the website is accessible and usable on desktop and laptop devices.	Yes

Mobile access	Whether the website is accessible and usable on mobile and tablet devices.	Yes
Load login page	Whether the website loads the login page once the login url is visited.	Yes
Load register page	Whether the website loads the register page once the register url is visited.	Yes
Load home page	Whether the website loads the home page once the home url is visited.	Yes
Load reset password page	Whether the website loads the reset password page once the reset password url is visited.	Yes
Load dashboard page	Whether the website loads the dashboard page once the dashboard url is visited.	Yes
Load settings page	Whether the website loads the settings page once the settings url is visited.	Yes
Mobile menu	Whether the mobile menu is visible and correctly toggles visibility on mobile devices.	Yes
Logout	Whether the user is logged out once the logout button is clicked	Yes

Table 48. End-to-end website access test plan

Name	Description	Success
Login	Whether the website allows users to login and authenticate their access.	Yes
Login error	Whether the website shows input errors when invalid data is input into the login form on the login page.	Yes

Table 49. End-to-end website login page test plan

Name	Description	Success
Forgot password	Whether the website allows users to reset or change the password of the account.	Yes
Forgot password error	Whether the website shows input errors when invalid data is input into the forgot password form on the forgot password page.	Yes

Table 50. End-to-end website forgot password page test plan

Name	Description	Success
Register	Whether the website allows users to register an account.	Yes
Register error	Whether the website shows input errors when invalid data is input into the register form on the register page.	Yes

Table 51. End-to-end website register page test plan

Name	Description	Success
Name and ID	Whether the website's node list shows each node's name and ID.	Yes
Climate data deletion	Whether the website's node list allows for the deletion of a node's climate data.	Yes
Sensor deletion	Whether the website's node list allows for the deletion of a node.	Yes
Renaming	Whether the website's node list allows for the renaming of each node's name based upon user input.	Yes
Battery status	Whether the website's node list shows each node's battery status.	Yes

Table 52. End-to-end website node list test plan

Name	Description	Success
Battery	Whether the website shows a line chart of the selected node's battery voltage over a user selected time period.	Yes
Climate data chart	Whether the website shows a line chart for each type of climate data collected by the selected node's over a user selected time period.	Yes

Table 53. End-to-end website historical data chart test plan

Name	Description	Success
Recent climate data list	Whether the website shows the most recent climate data collected by the selected node.	Yes

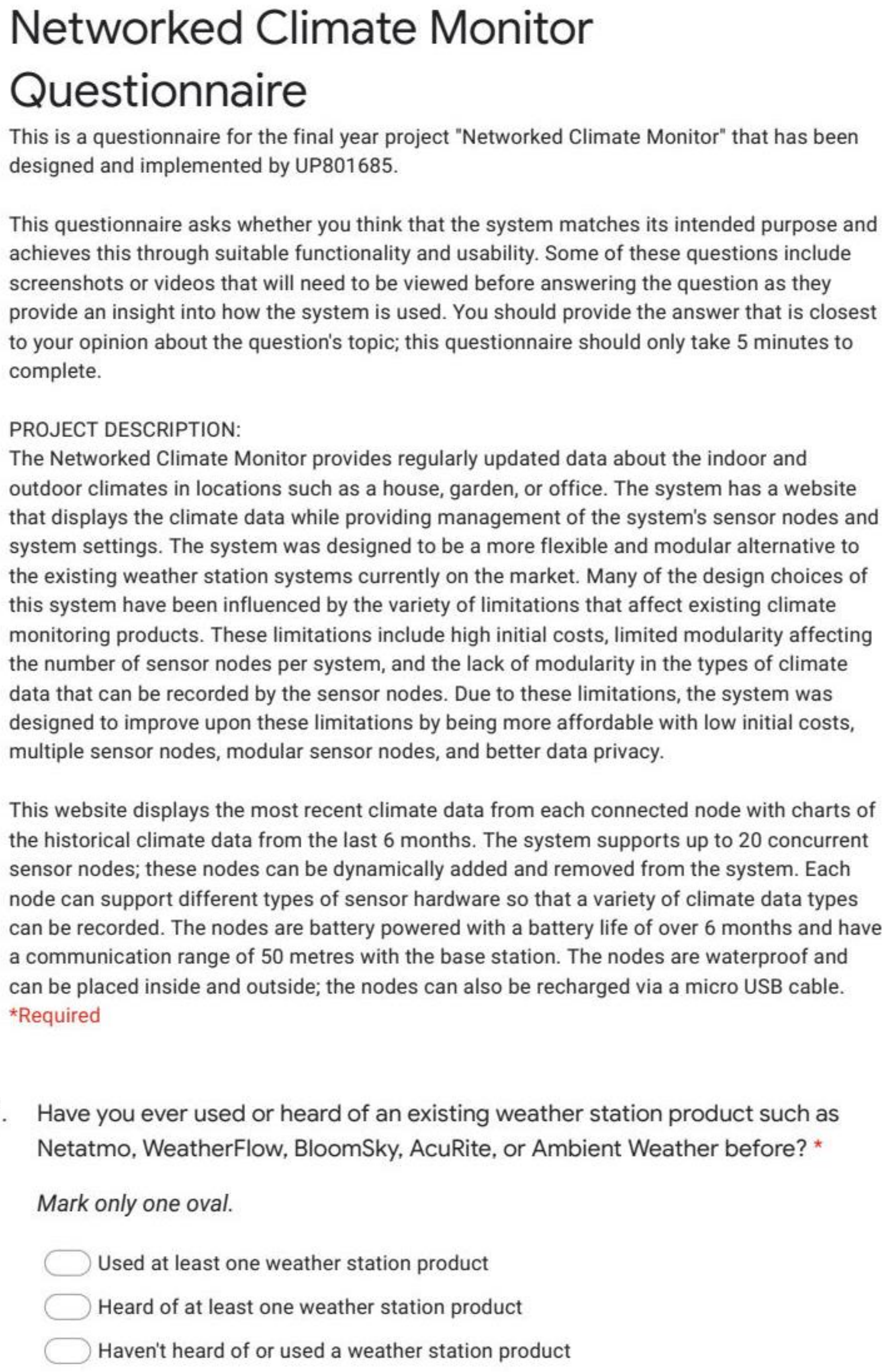
Table 54. End-to-end website recent climate data list test plan

Name	Description	Success
Celsius	Whether the website can show the sensor temperature data in Celsius values.	Yes

Farenheit	Whether the website can show the sensor temperature data in fahrenheit values.	Yes
Change unit	Whether the website changes the active temperature unit once a unit radio input is selected.	Yes
Change settings category	Whether the website changes the active settings category once a category selection button is clicked.	Yes
Change base station wifi settings	Whether the website changes the wifi settings once the save settings button is clicked.	Yes
Change measurement interval	Whether the website changes the active measurement interval once a dropdown option is selected.	Yes

Table 55. End-to-end website settings test plan

11.10 Appendix J: Questionnaire Design



Networked Climate Monitor Questionnaire

This is a questionnaire for the final year project "Networked Climate Monitor" that has been designed and implemented by UP801685.

This questionnaire asks whether you think that the system matches its intended purpose and achieves this through suitable functionality and usability. Some of these questions include screenshots or videos that will need to be viewed before answering the question as they provide an insight into how the system is used. You should provide the answer that is closest to your opinion about the question's topic; this questionnaire should only take 5 minutes to complete.

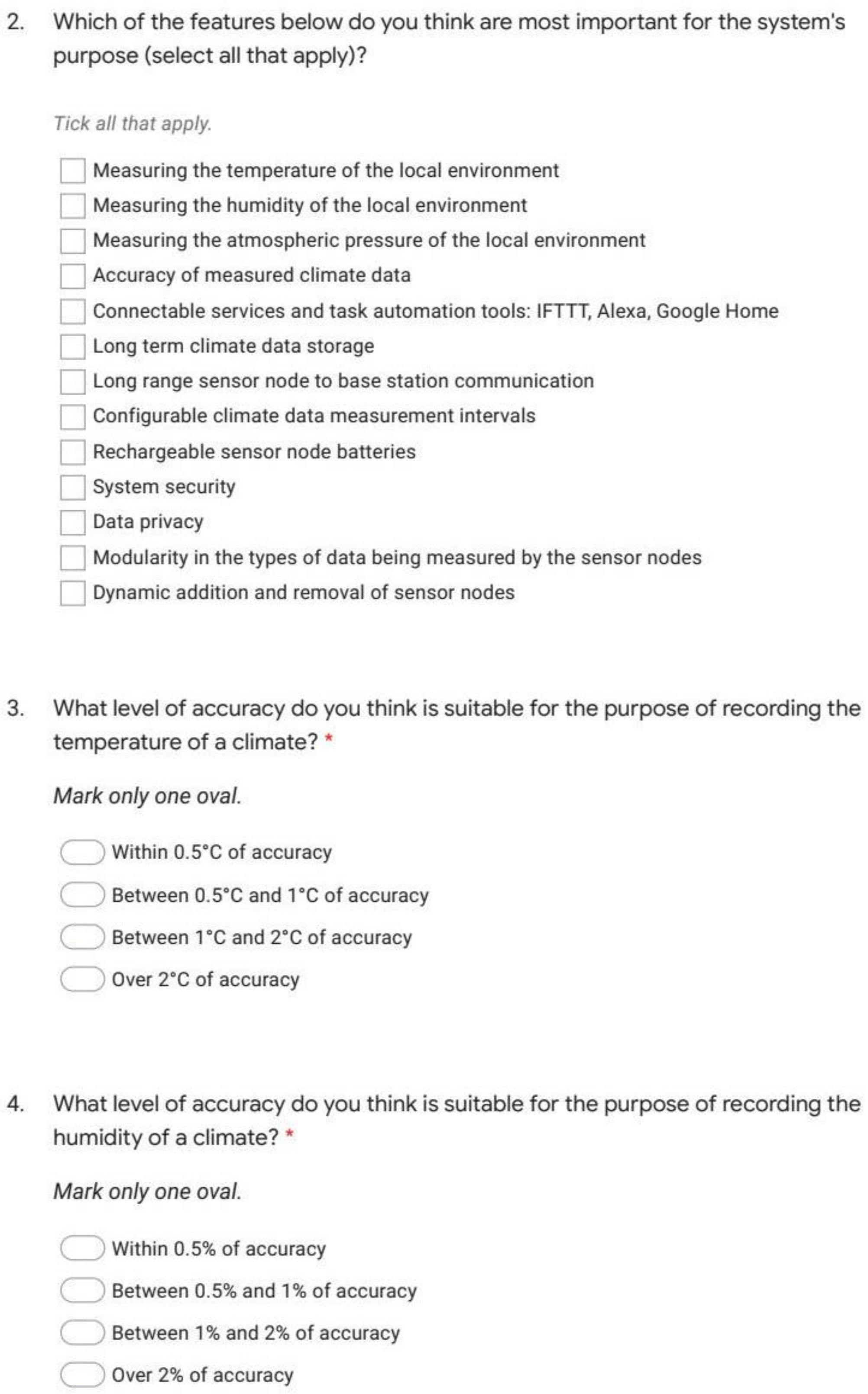
PROJECT DESCRIPTION:
The Networked Climate Monitor provides regularly updated data about the indoor and outdoor climates in locations such as a house, garden, or office. The system has a website that displays the climate data while providing management of the system's sensor nodes and system settings. The system was designed to be a more flexible and modular alternative to the existing weather station systems currently on the market. Many of the design choices of this system have been influenced by the variety of limitations that affect existing climate monitoring products. These limitations include high initial costs, limited modularity affecting the number of sensor nodes per system, and the lack of modularity in the types of climate data that can be recorded by the sensor nodes. Due to these limitations, the system was designed to improve upon these limitations by being more affordable with low initial costs, multiple sensor nodes, modular sensor nodes, and better data privacy.

This website displays the most recent climate data from each connected node with charts of the historical climate data from the last 6 months. The system supports up to 20 concurrent sensor nodes; these nodes can be dynamically added and removed from the system. Each node can support different types of sensor hardware so that a variety of climate data types can be recorded. The nodes are battery powered with a battery life of over 6 months and have a communication range of 50 metres with the base station. The nodes are waterproof and can be placed inside and outside; the nodes can also be recharged via a micro USB cable.
*Required

1. Have you ever used or heard of an existing weather station product such as Netatmo, WeatherFlow, BloomSky, AcuRite, or Ambient Weather before? *

   Mark only one oval.

   ○ Used at least one weather station product
   ○ Heard of at least one weather station product
   ○ Haven't heard of or used a weather station product



2. Which of the features below do you think are most important for the system's purpose (select all that apply)?

Tick all that apply.

□ Measuring the temperature of the local environment
□ Measuring the humidity of the local environment
□ Measuring the atmospheric pressure of the local environment
□ Accuracy of measured climate data
□ Connectable services and task automation tools: IFTTT, Alexa, Google Home
□ Long term climate data storage
□ Long range sensor node to base station communication
□ Configurable climate data measurement intervals
□ Rechargeable sensor node batteries
□ System security
□ Data privacy
□ Modularity in the types of data being measured by the sensor nodes
□ Dynamic addition and removal of sensor nodes

3. What level of accuracy do you think is suitable for the purpose of recording the temperature of a climate? *

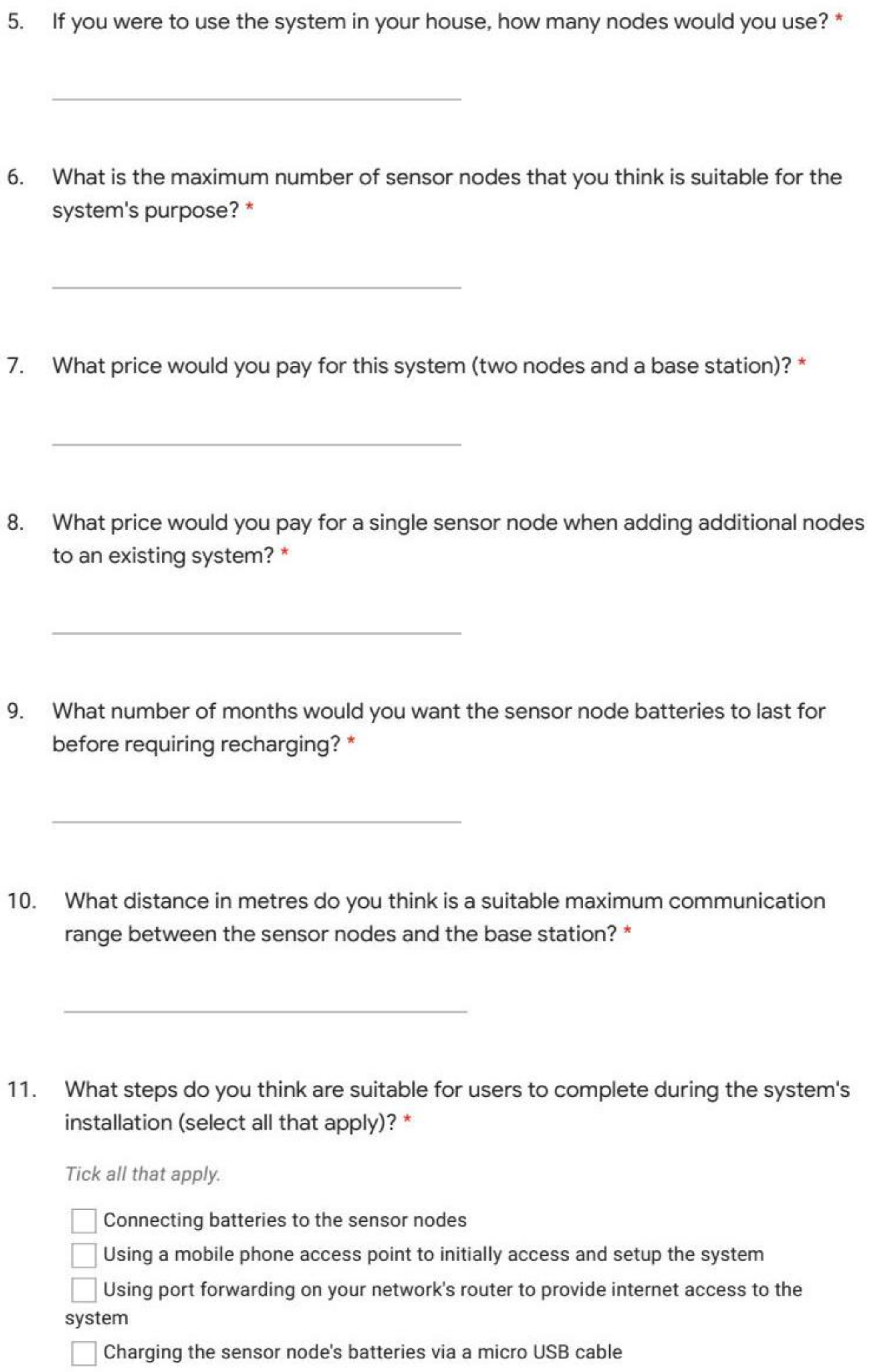
Mark only one oval.

○ Within 0.5°C of accuracy
○ Between 0.5°C and 1°C of accuracy
○ Between 1°C and 2°C of accuracy
○ Over 2°C of accuracy

4. What level of accuracy do you think is suitable for the purpose of recording the humidity of a climate? *

Mark only one oval.

○ Within 0.5% of accuracy
○ Between 0.5% and 1% of accuracy
○ Between 1% and 2% of accuracy
○ Over 2% of accuracy



5. If you were to use the system in your house, how many nodes would you use? *

____

6. What is the maximum number of sensor nodes that you think is suitable for the system's purpose? *

____

7. What price would you pay for this system (two nodes and a base station)? *

____

8. What price would you pay for a single sensor node when adding additional nodes to an existing system? *

____

9. What number of months would you want the sensor node batteries to last for before requiring recharging? *

____

10. What distance in metres do you think is a suitable maximum communication range between the sensor nodes and the base station? *

____

11. What steps do you think are suitable for users to complete during the system's installation (select all that apply)? *

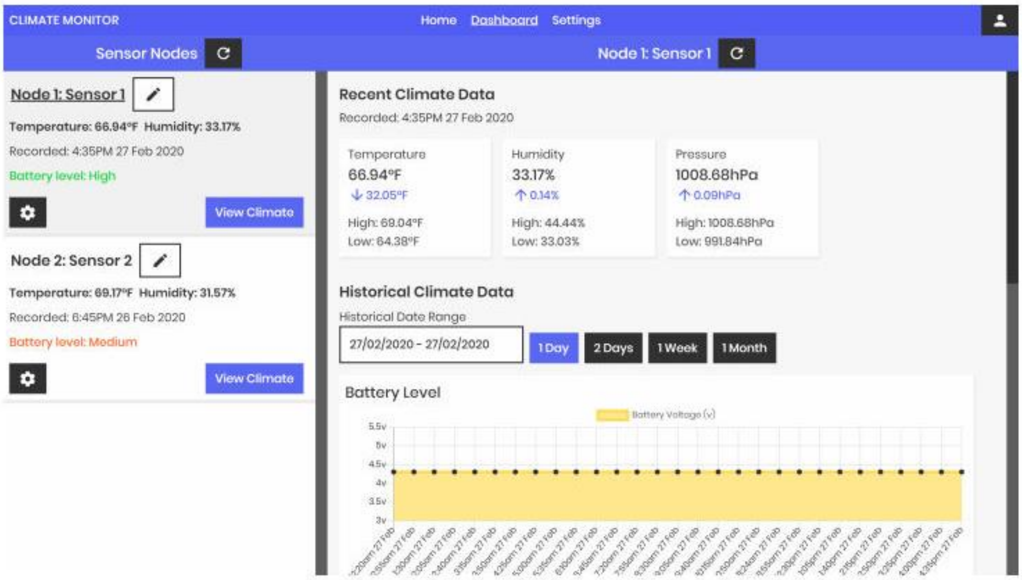
Tick all that apply.

□ Connecting batteries to the sensor nodes
□ Using a mobile phone access point to initially access and setup the system
□ Using port forwarding on your network's router to provide internet access to the system
□ Charging the sensor node's batteries via a micro USB cable

User Interface

The following section asks questions on whether the user interface meets usability criteria.

Desktop Screenshot of Website Dashboard (Look at this before answering the next question)



Node 1: Sensor 1

Temperature: 66.94°F Humidity: 33.17%

Recorded: 4:35PM 27 Feb 2020

Battery level: High

Node 2: Sensor 2

Temperature: 69.17°F Humidity: 31.57%

Recorded: 6:45PM 26 Feb 2020

Battery level: Medium

Recent Climate Data

Recorded: 4:35PM 27 Feb 2020

Temperature	Humidity	Pressure
66.94°F	33.17%	1008.68hPa
↓ 32.05°F	↑ 0.14%	↑ 0.09hPa
High: 69.04°F	High: 44.44%	High: 1008.68hPa
Low: 64.38°F	Low: 33.03%	Low: 991.84hPa

Historical Climate Data

Historical Date Range: 27/02/2020 - 27/02/2020

Battery Level

12. Do you think that the text in the screenshot above is easy to read? *

Mark only one oval.

○ No

○ Yes

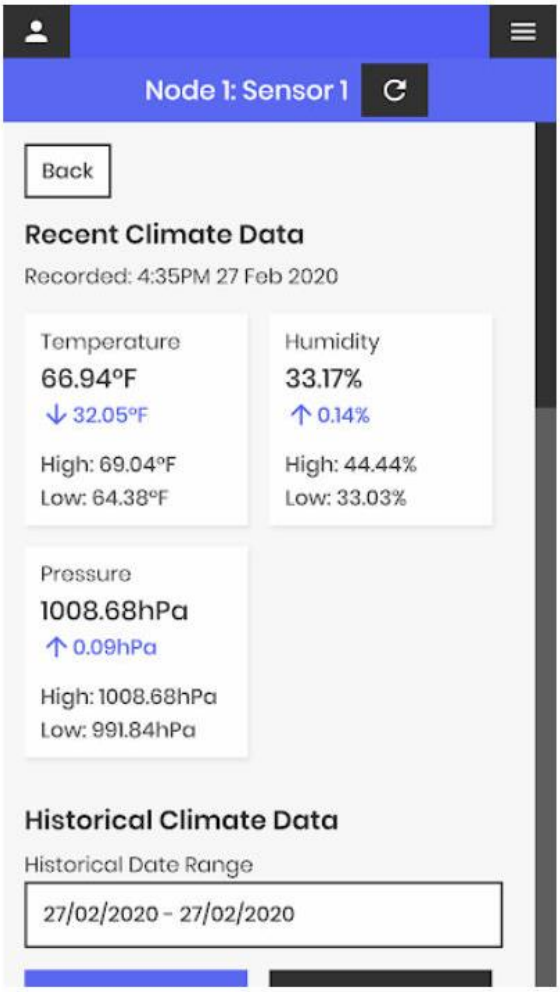
13. Do you think that the purpose and content of the desktop version of the dashboard page are easy to understand? *

Mark only one oval.

○ No

○ Yes

Mobile Screenshot of Website Dashboard (Look at this before answering the next question)



Node 1: Sensor 1

Back

Recent Climate Data

Recorded: 4:35PM 27 Feb 2020

Temperature
66.94°F
↓ 32.05°F
High: 69.04°F
Low: 64.38°F

Humidity
33.17%
↑ 0.14%
High: 44.44%
Low: 33.03%

Pressure
1008.68hPa
↑ 0.09hPa
High: 1008.68hPa
Low: 991.84hPa

Historical Climate Data

Historical Date Range

27/02/2020 - 27/02/2020

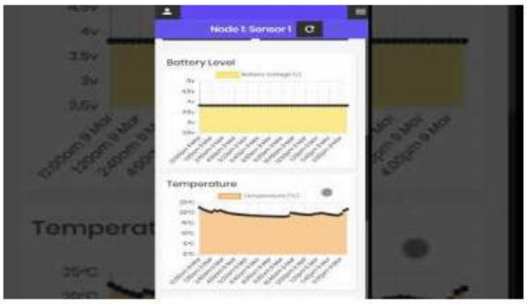
14. Do you think that the purpose and content of the mobile version of the dashboard page are easy to understand? *

Mark only one oval.

○ No

○ Yes

Video of Mobile Dashboard Interactions (Watch at this before answering the next question)



http://youtube.com/watch?v=HlVal-01giQ

15. Do you think that the mobile version of the dashboard page has suitable interactions for its purpose of managing sensor nodes and the viewing of recent and historical climate data? *

Mark only one oval.

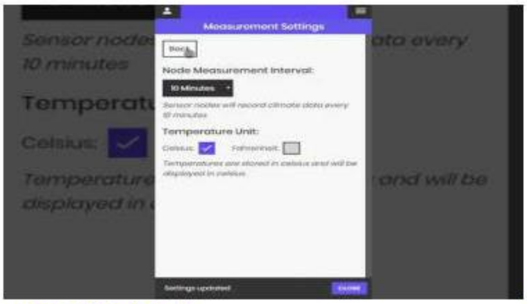
○ Not suitable

○ Partially suitable

○ Mostly suitable

○ Very suitable

Video of Mobile Settings Interactions (Watch at this before answering the next question)


Measurement Settings

Back

Node Measurement Interval:

10 Minutes

Sensor nodes will record climate data every 10 minutes

Temperature Unit:

Celsius: ☑ Fahrenheit: □

Temperatures are stored in celsius and will be displayed in celsius

Settings updated

http://youtube.com/watch?v=RX5UDizbGtU

16. Do you think that the mobile version of the settings page has suitable interactions for its purpose of managing measurement and wifi settings, and resetting the account's password? *

Mark only one oval.

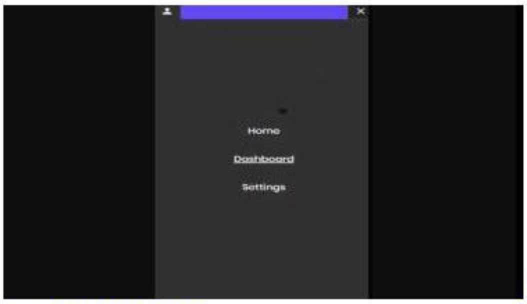
○ Not suitable

○ Partially suitable

○ Mostly suitable

○ Very suitable

Video of Mobile Menu Interactions (Watch at this before answering the next question)



http://youtube.com/watch?v=mBE2neQEnI8

17. Do you think that the mobile menu is a suitable method for providing navigation between the website's pages while on a mobile device? *

Mark only one oval.

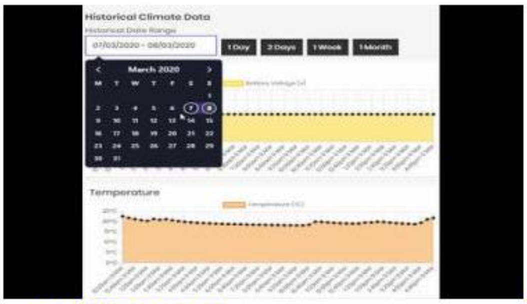
○ Not suitable

○ Partially suitable

○ Mostly suitable

○ Very suitable

Video of historical climate data date period selection (Watch at this before answering the next question)



http://youtube.com/watch?v=ZZW3SQ7AW_M

18. Do you think that the method shown in the video above is suitable for selecting the date periods for the historical climate data graphs? *

Mark only one oval.

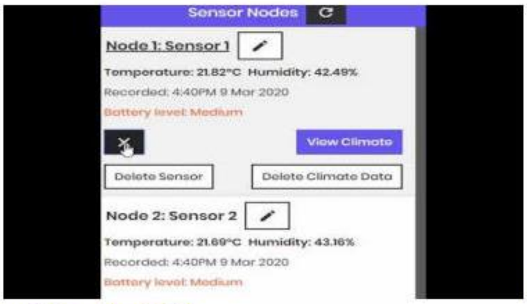
○ Not suitable

○ Partially suitable

○ Mostly suitable

○ Very suitable

Video of Deleting a Sensor Node (Watch at this before answering the next question)



Node 1: Sensor 1

Temperature: 21.82°C Humidity: 42.49%

Recorded: 4:40PM 9 Mar 2020

Battery level: Medium

Node 2: Sensor 2

Temperature: 21.69°C Humidity: 43.16%

Recorded: 4:40PM 9 Mar 2020

Battery level: Medium

http://youtube.com/watch?v=871ez6xg2CY

19. Do you think that the method shown above for deleting a sensor node is suitable? *

Mark only one oval.

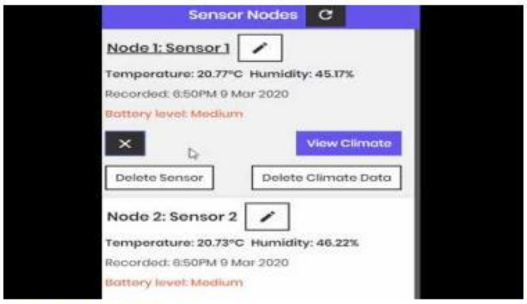
○ Not suitable

○ Partially suitable

○ Mostly suitable

○ Very suitable

Video of Deleting Climate Data from a Sensor Node (Watch at this before answering the next question)



Temperature: 20.77°C Humidity: 45.17%

Recorded: 6:50PM 9 Mar 2020

Battery level: Medium

Temperature: 20.73°C Humidity: 46.22%

Recorded: 6:50PM 9 Mar 2020

Battery level: Medium

http://youtube.com/watch?v=-PULNWHHN4o

20. Do you think that the method shown above for deleting the climate data from a sensor node is suitable? *

Mark only one oval.

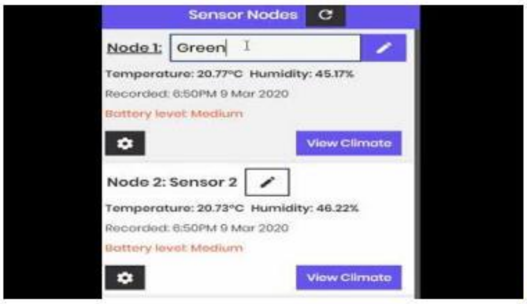
○ Not suitable

○ Partially suitable

○ Mostly suitable

○ Very suitable

Video of Renaming a Sensor Node (Watch at this before answering the next question)



Sensor Nodes

Node 1: Green

Temperature: 20.77°C Humidity: 45.17%

Recorded: 6:50PM 9 Mar 2020

Battery level: Medium

View Climate

Node 2: Sensor 2

Temperature: 20.73°C Humidity: 46.22%

Recorded: 6:50PM 9 Mar 2020

Battery level: Medium

View Climate

http://youtube.com/watch?v=QSrFkv8GGHo

21. Do you think that the method of renaming sensor nodes shown above is suitable? *

Mark only one oval.

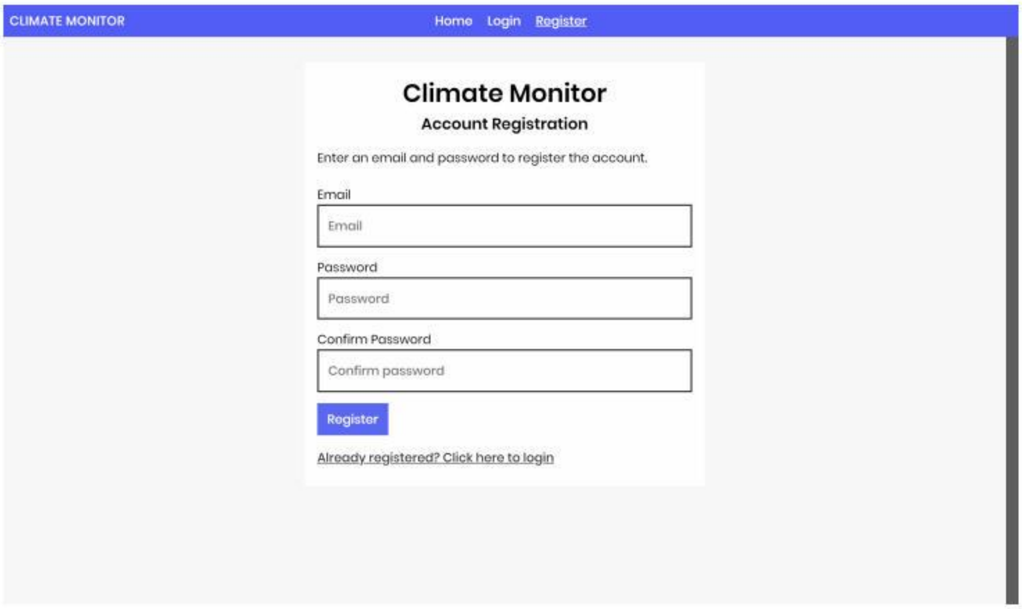
○ Not suitable

○ Partially suitable

○ Mostly suitable

○ Very suitable

Screenshot of Registration Page (Look at this before answering the next question)



CLIMATE MONITOR Home Login Register

Climate Monitor

Account Registration

Enter an email and password to register the account.

Email

Email

Password

Password

Confirm Password

Confirm password

Register

Already registered? Click here to login

22. Do you think that the register page shown above, is suitable for the purpose of registering the account? *

Mark only one oval.

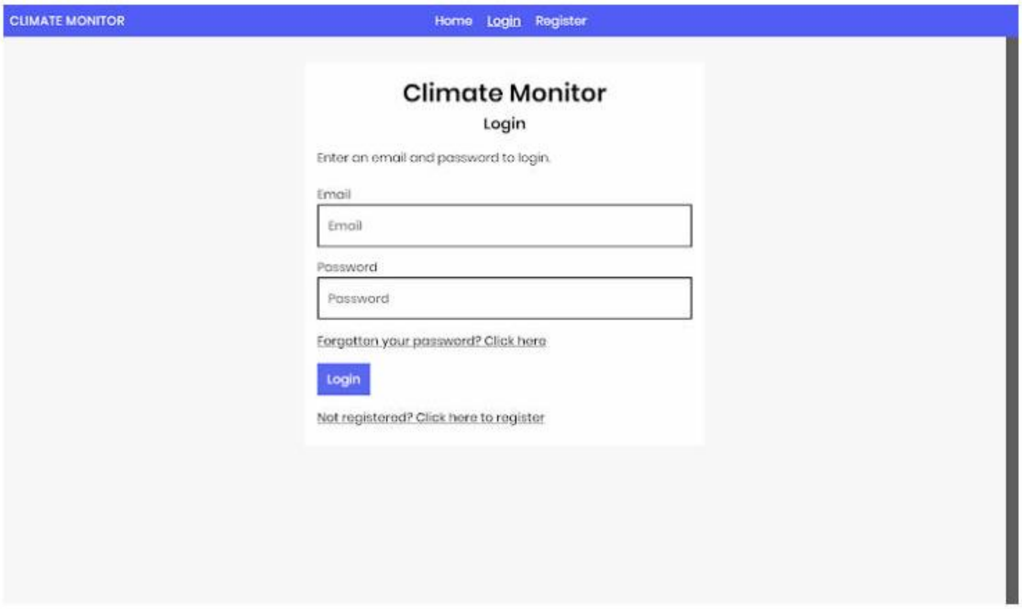
○ Not suitable

○ Partially suitable

○ Mostly suitable

○ Very suitable

Screenshot of Login Page (Look at this before answering the next question)



Climate Monitor

Login

Enter an email and password to login.

Email: Email

Password: Password

Forgotten your password? Click here

Not registered? Click here to register

23. Do you think that the login page shown above, is suitable for the purpose of logging into the existing account? *

Mark only one oval.

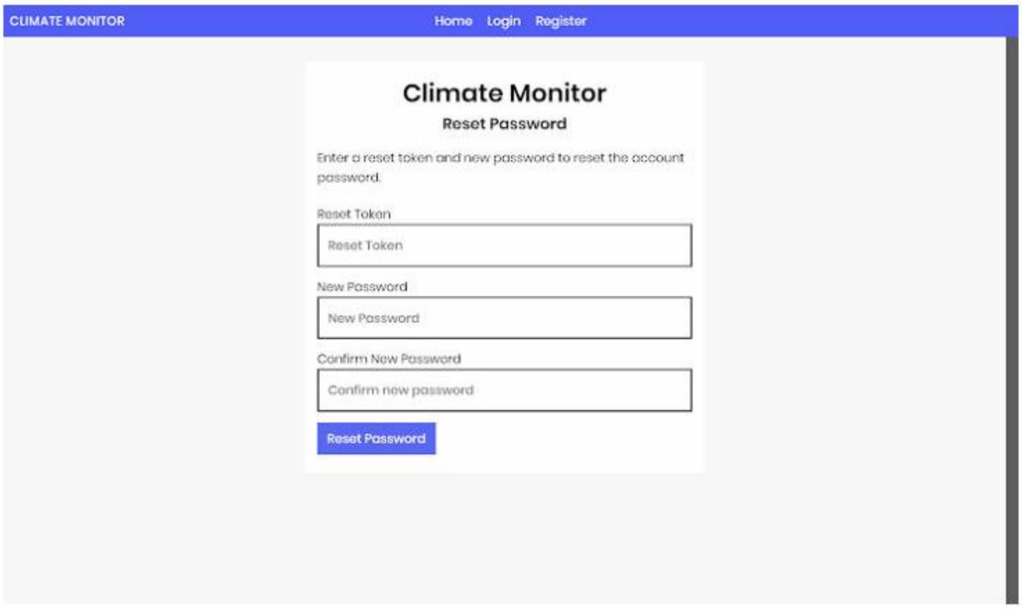
○ Not suitable

○ Partially suitable

○ Mostly suitable

○ Very suitable

Screenshot of Password Reset Page (Look at this before answering the next question)



Climate Monitor

Reset Password

Enter a reset token and new password to reset the account password.

Reset Token

Reset Token

New Password

New Password

Confirm New Password

Confirm new password

24. Do you think that the password reset page shown above, is suitable for the purpose of resetting the account password? *

Mark only one oval.

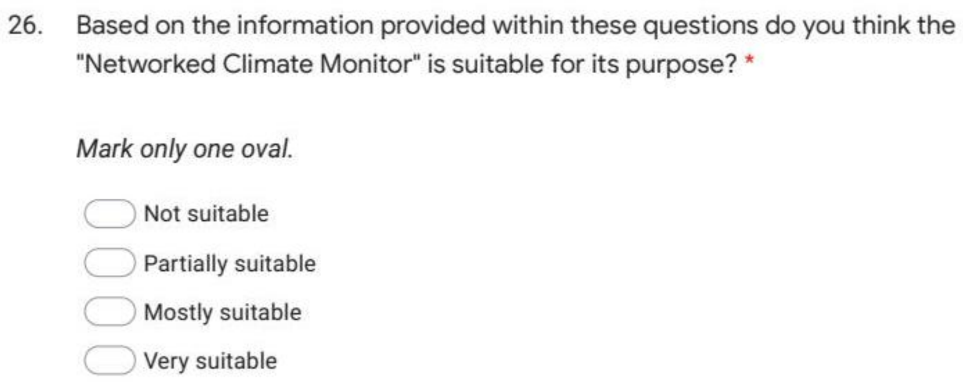
○ Not suitable

○ Partially suitable

○ Mostly suitable

○ Very suitable

25. Do you have any comments or suggestions on how to improve the user interface?



26. Based on the information provided within these questions do you think the "Networked Climate Monitor" is suitable for its purpose? *

Mark only one oval.

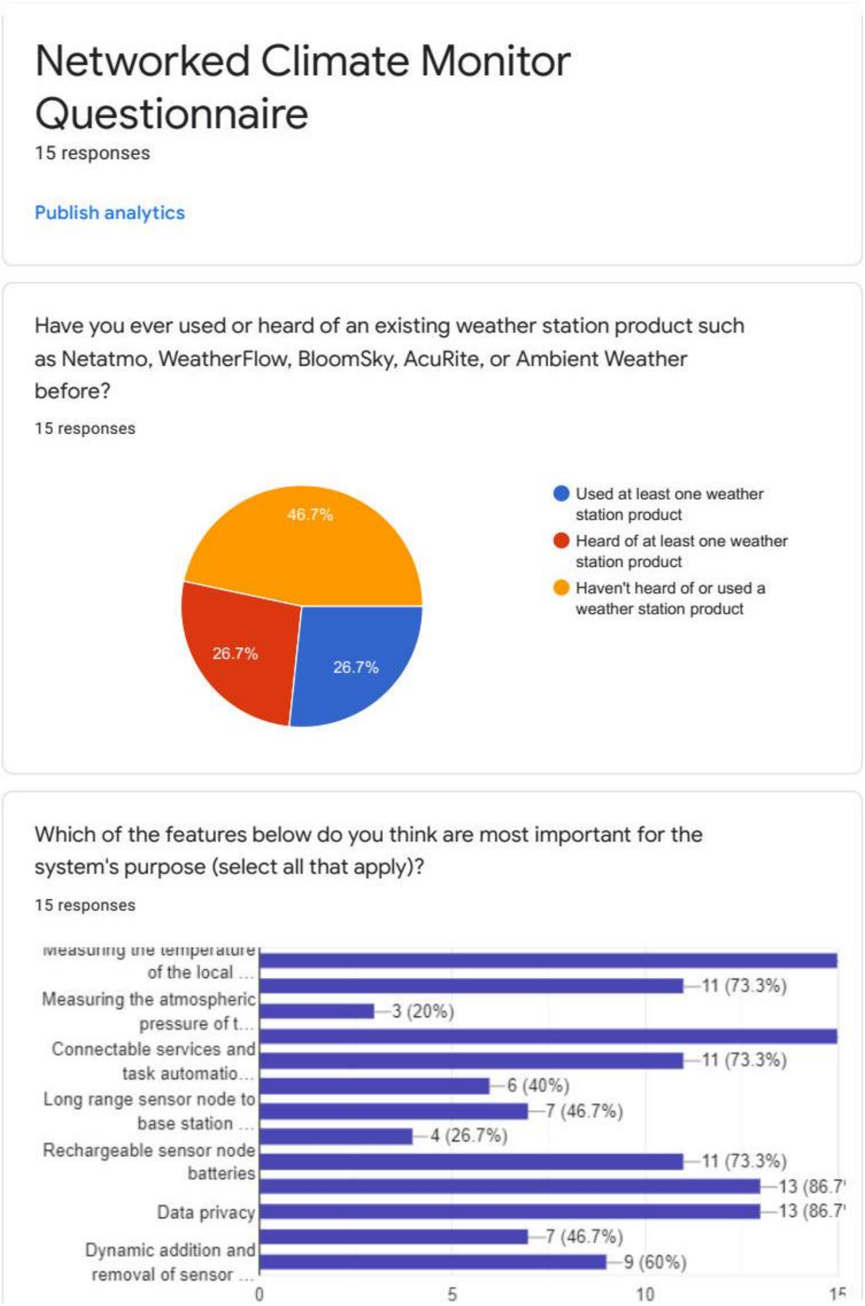
○ Not suitable

○ Partially suitable

○ Mostly suitable

○ Very suitable

11.11 Appendix K: Questionnaire Responses



Networked Climate Monitor Questionnaire

15 responses

Publish analytics

Have you ever used or heard of an existing weather station product such as Netatmo, WeatherFlow, BloomSky, AcuRite, or Ambient Weather before?

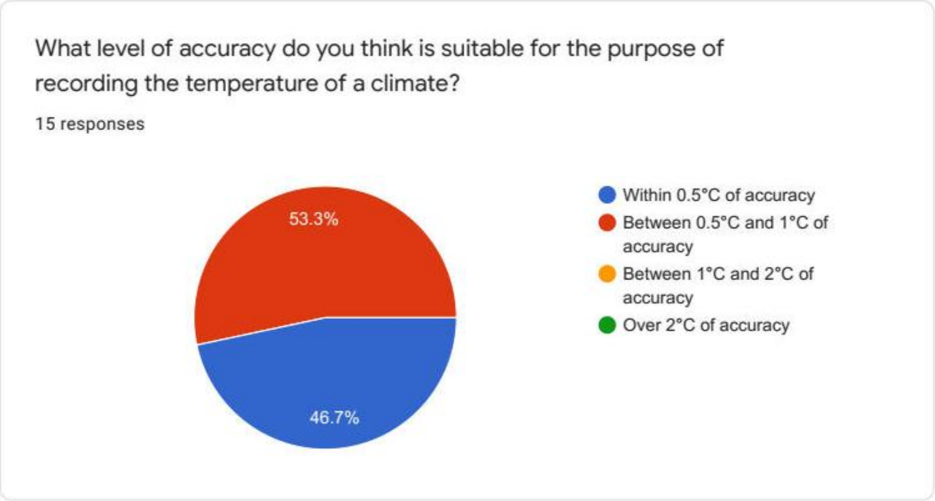
15 responses

46.7%
26.7%
26.7%
● Used at least one weather station product
● Heard of at least one weather station product
● Haven't heard of or used a weather station product

Which of the features below do you think are most important for the system's purpose (select all that apply)?

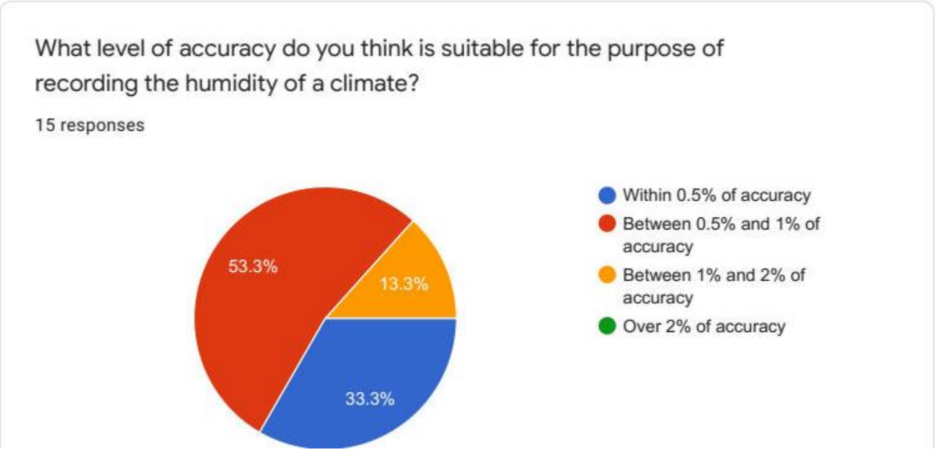
15 responses

Measuring the temperature of the local ...
Measuring the atmospheric pressure of t...
Connectable services and task automatio...
Long range sensor node to base station ...
Rechargeable sensor node batteries
Data privacy
Dynamic addition and removal of sensor ...
11 (73.3%)
3 (20%)
11 (73.3%)
6 (40%)
7 (46.7%)
4 (26.7%)
11 (73.3%)
13 (86.7
13 (86.7
7 (46.7%)
9 (60%)
0
5
10
15



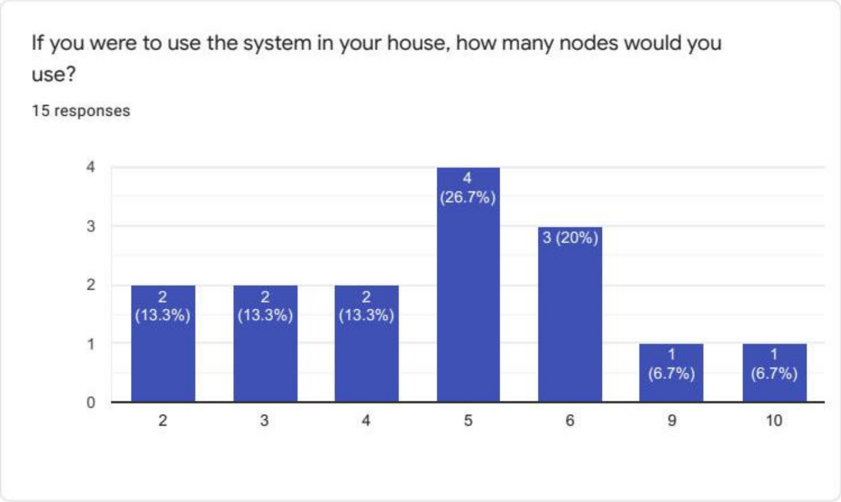
What level of accuracy do you think is suitable for the purpose of recording the temperature of a climate?

15 responses



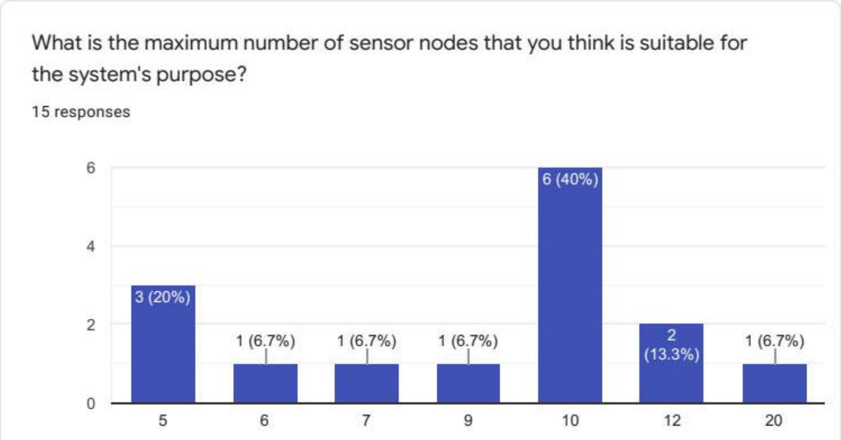
What level of accuracy do you think is suitable for the purpose of recording the humidity of a climate?

15 responses



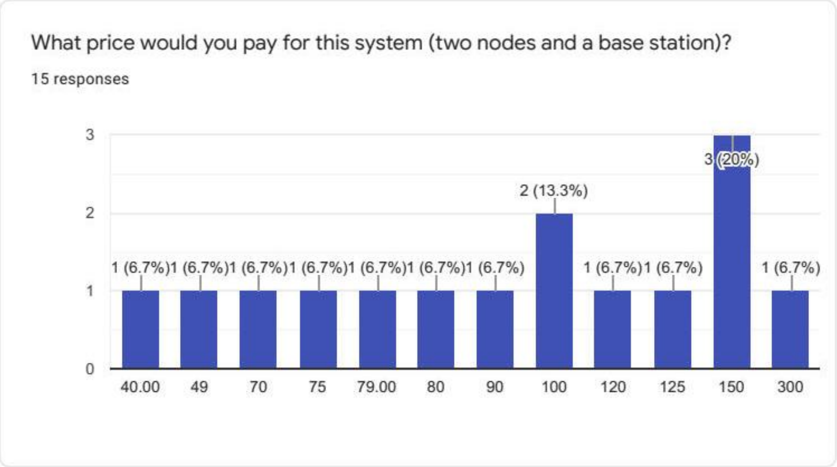
If you were to use the system in your house, how many nodes would you use?

15 responses



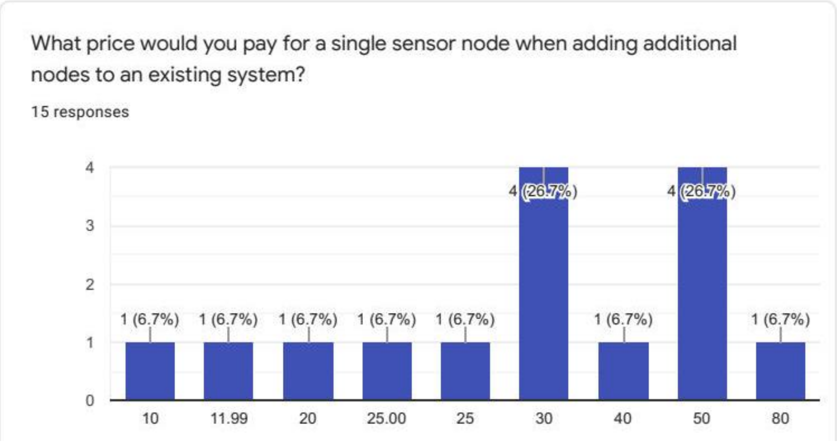
What is the maximum number of sensor nodes that you think is suitable for the system's purpose?

15 responses



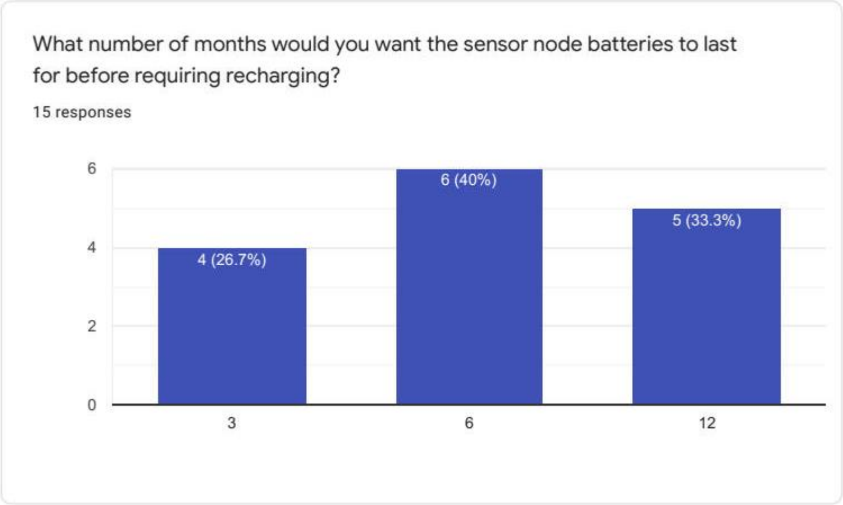
What price would you pay for this system (two nodes and a base station)?

15 responses



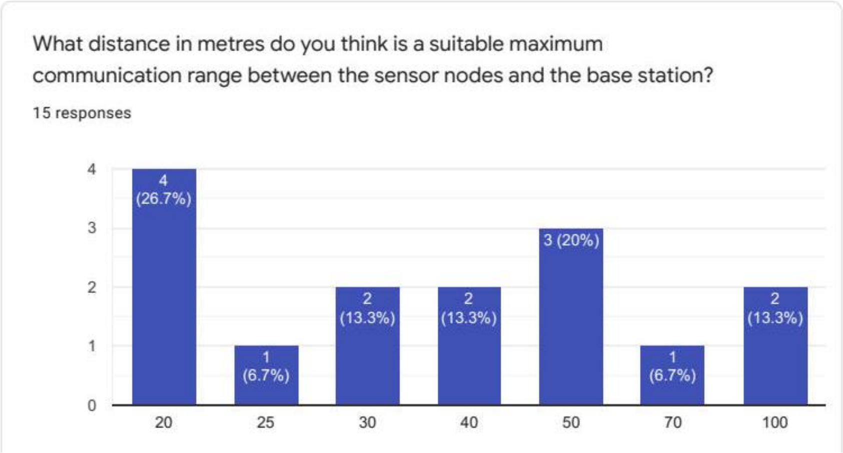
What price would you pay for a single sensor node when adding additional nodes to an existing system?

15 responses



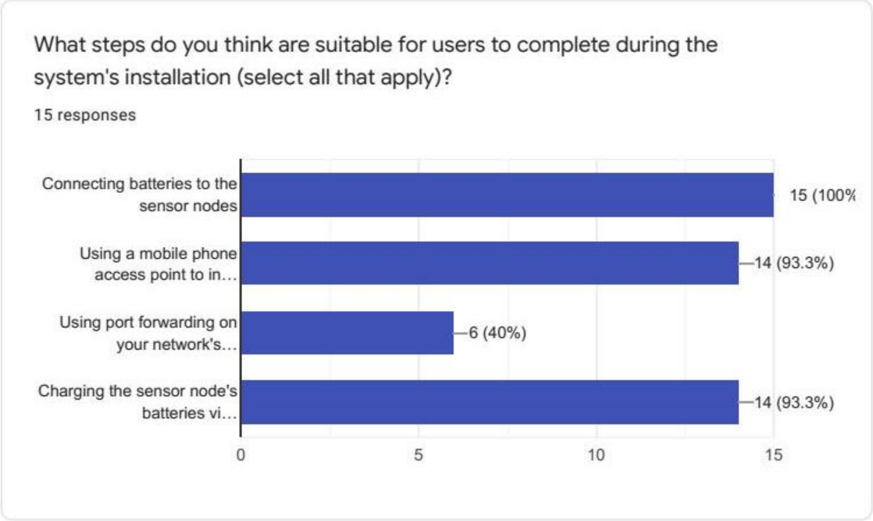
What number of months would you want the sensor node batteries to last for before requiring recharging?

15 responses



What distance in metres do you think is a suitable maximum communication range between the sensor nodes and the base station?

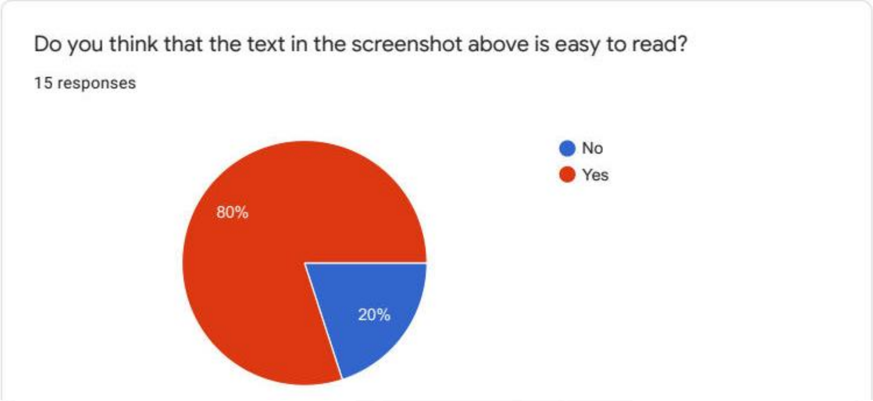
15 responses



What steps do you think are suitable for users to complete during the system's installation (select all that apply)?

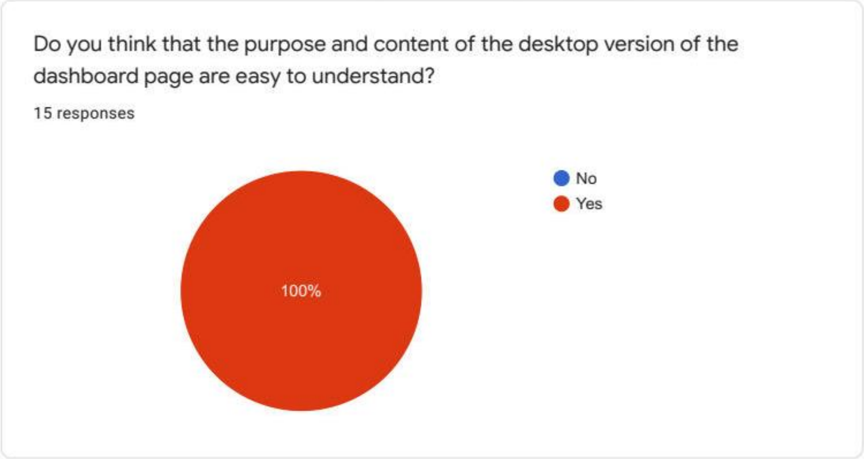
15 responses

User Interface



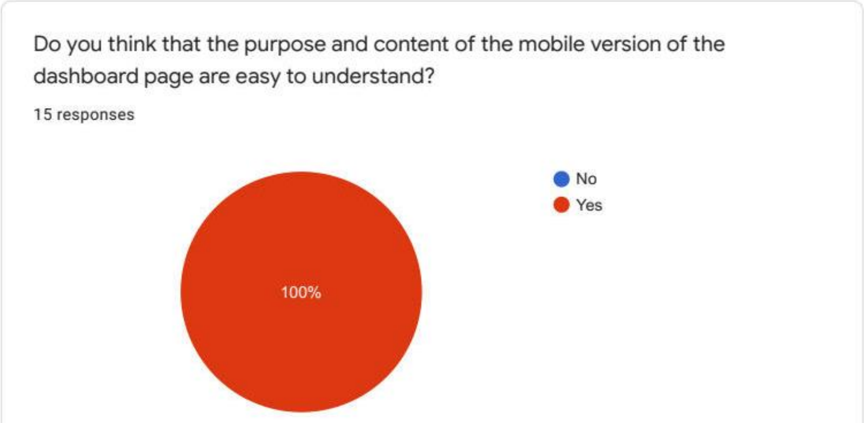
Do you think that the text in the screenshot above is easy to read?

15 responses



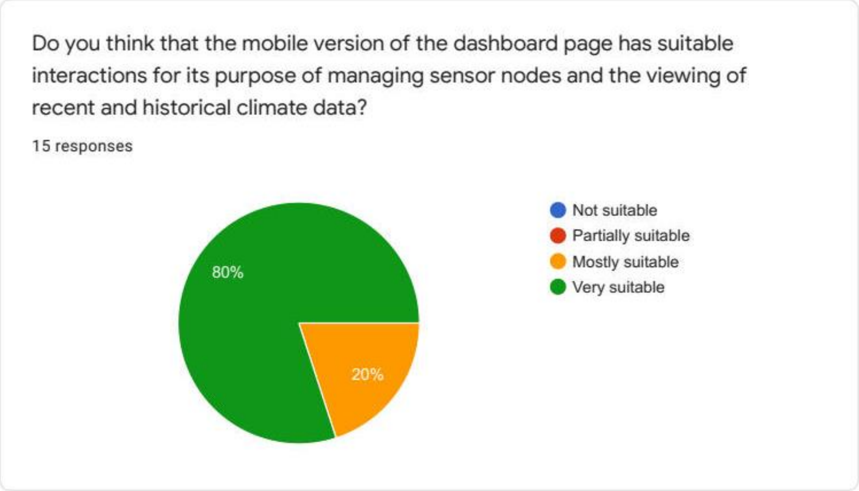
Do you think that the purpose and content of the desktop version of the dashboard page are easy to understand?

15 responses



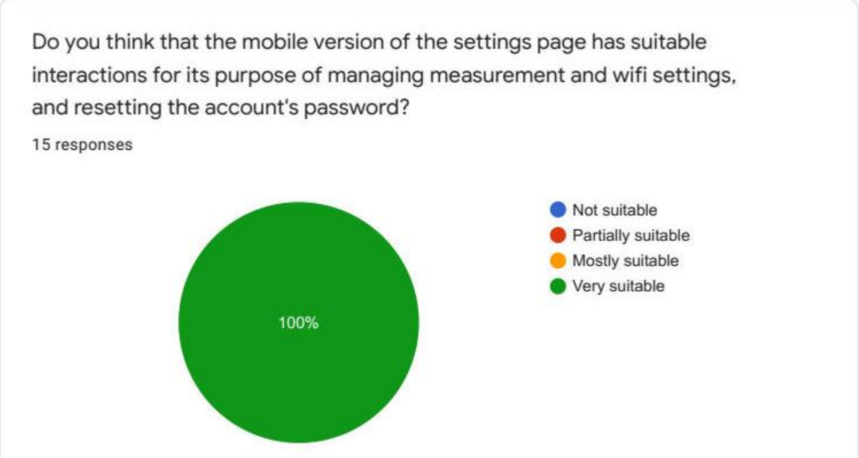
Do you think that the purpose and content of the mobile version of the dashboard page are easy to understand?

15 responses



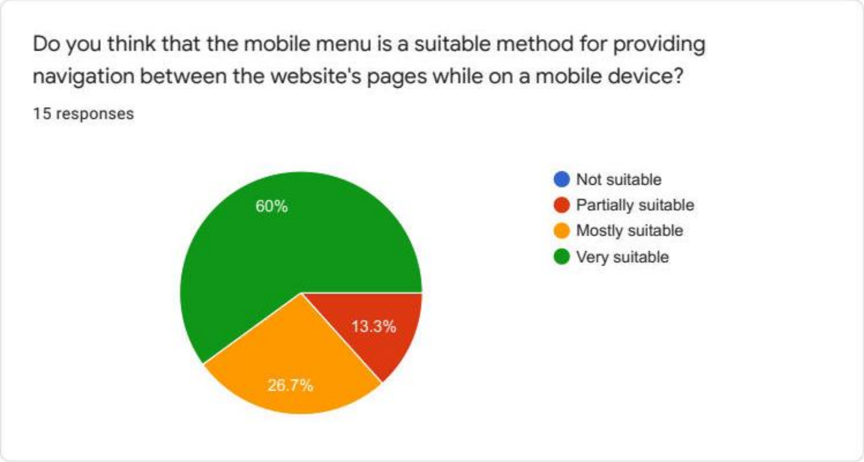
Do you think that the mobile version of the dashboard page has suitable interactions for its purpose of managing sensor nodes and the viewing of recent and historical climate data?

15 responses



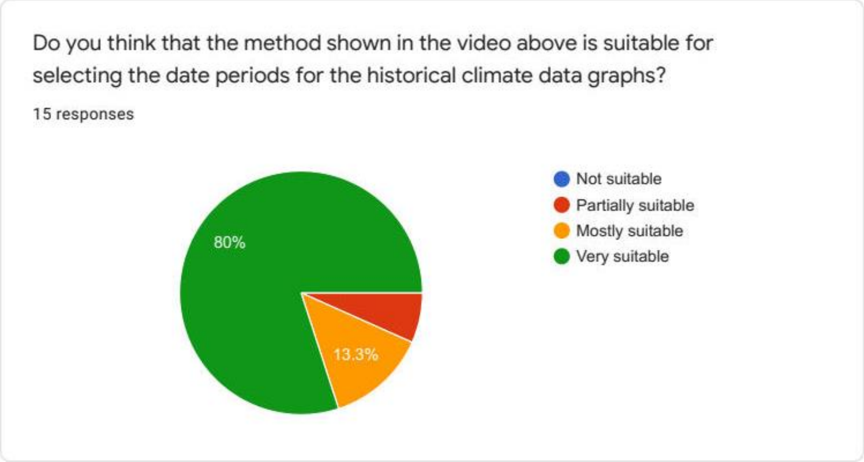
Do you think that the mobile version of the settings page has suitable interactions for its purpose of managing measurement and wifi settings, and resetting the account's password?

15 responses



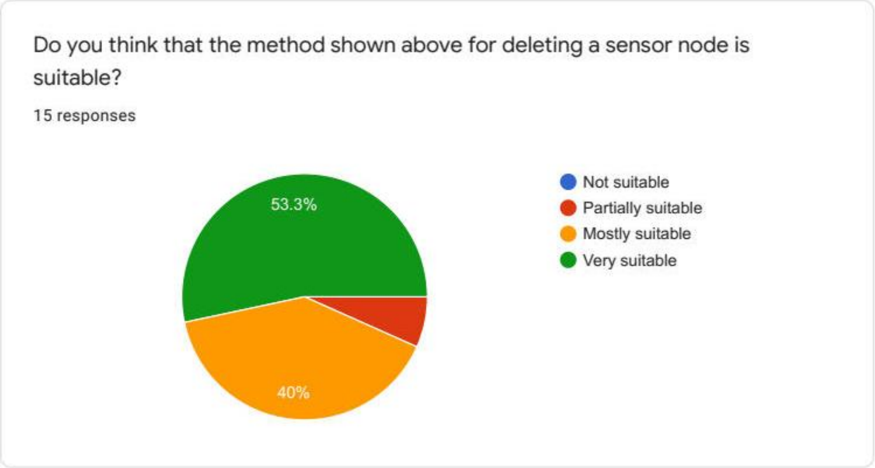
Do you think that the mobile menu is a suitable method for providing navigation between the website's pages while on a mobile device?

15 responses



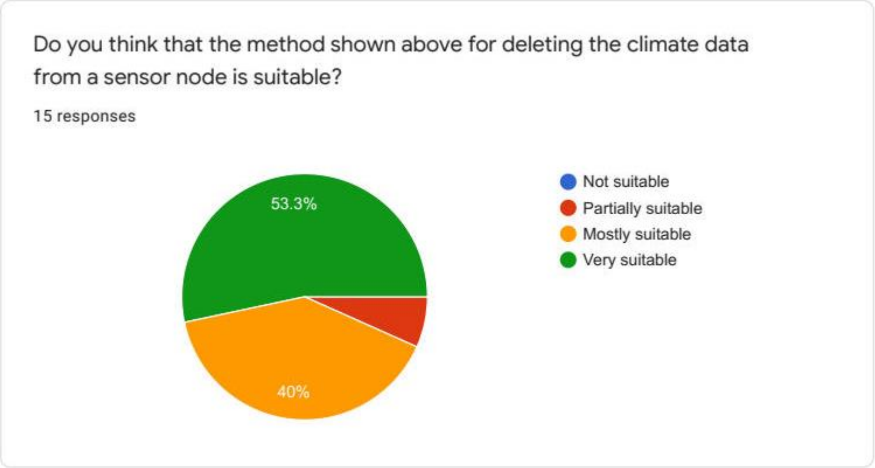
Do you think that the method shown in the video above is suitable for selecting the date periods for the historical climate data graphs?

15 responses



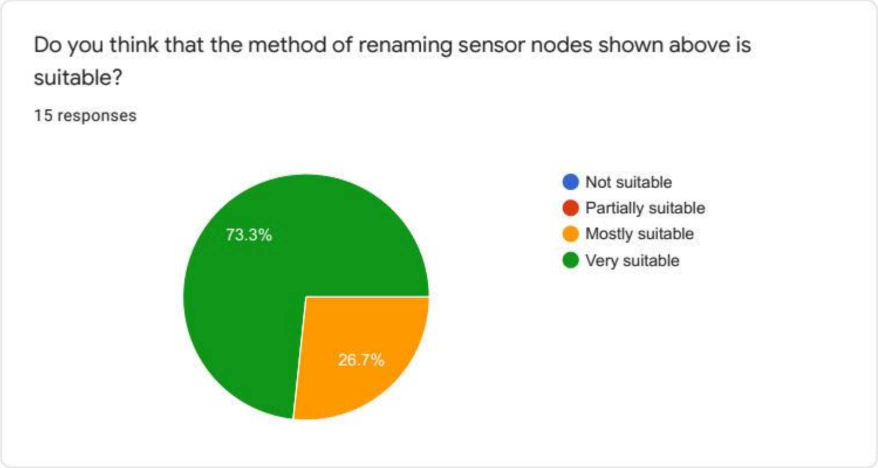
Do you think that the method shown above for deleting a sensor node is suitable?

15 responses



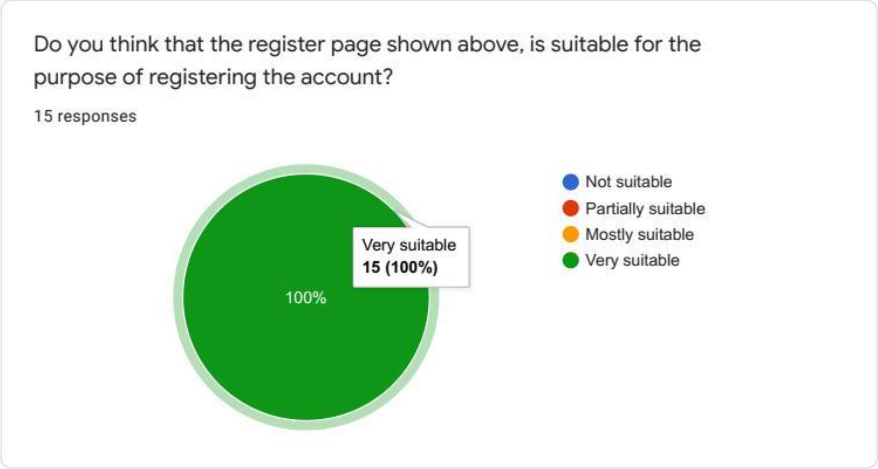
Do you think that the method shown above for deleting the climate data from a sensor node is suitable?

15 responses



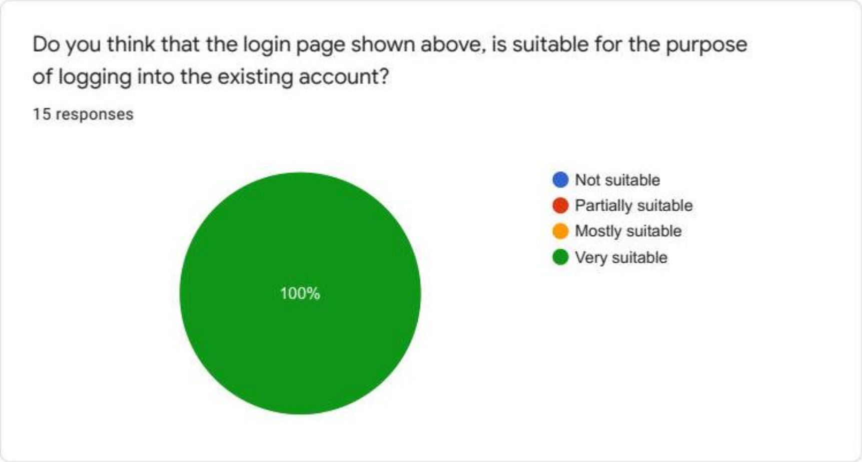
Do you think that the method of renaming sensor nodes shown above is suitable?

15 responses



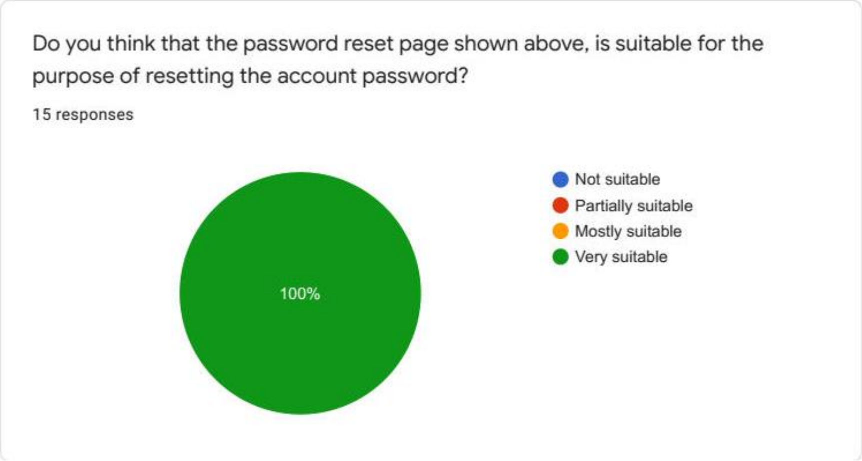
Do you think that the register page shown above, is suitable for the purpose of registering the account?

15 responses



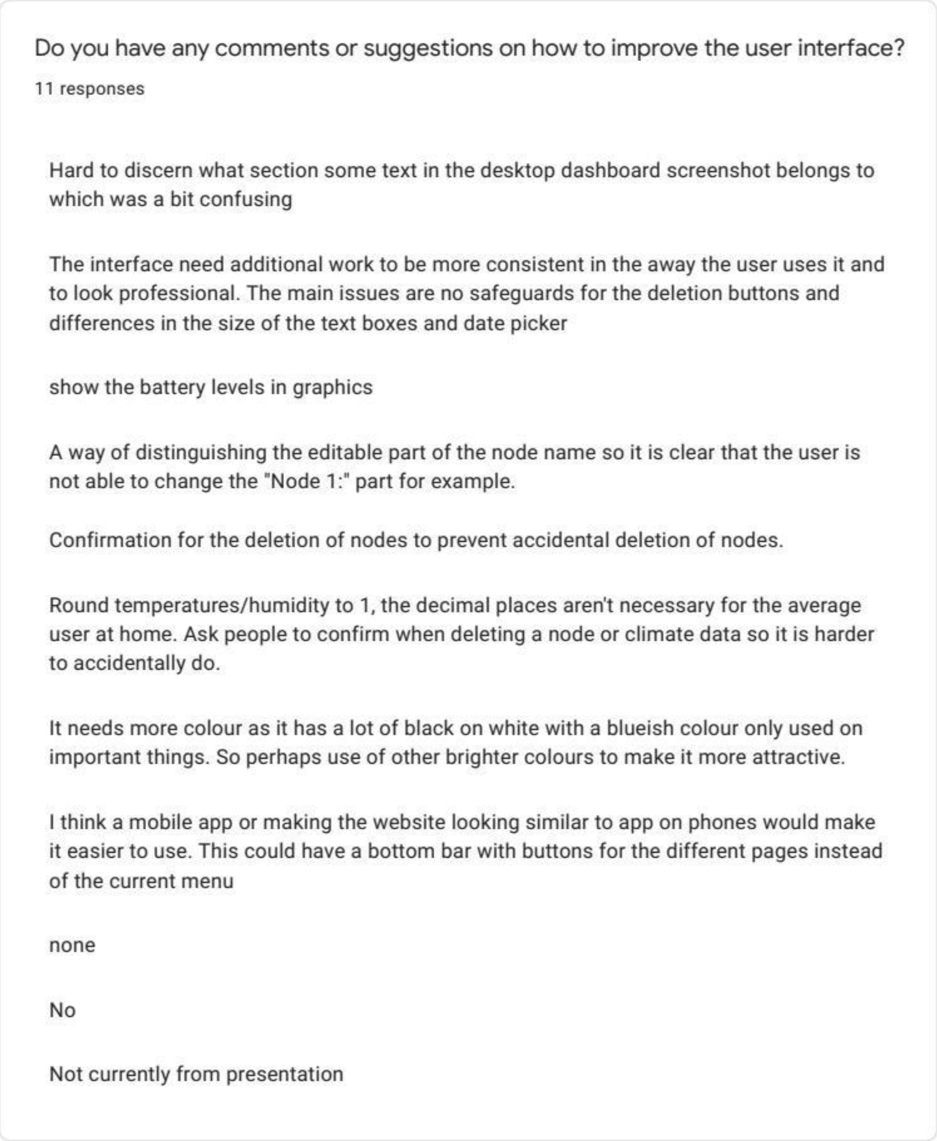
Do you think that the login page shown above, is suitable for the purpose of logging into the existing account?

15 responses



Do you think that the password reset page shown above, is suitable for the purpose of resetting the account password?

15 responses



Do you have any comments or suggestions on how to improve the user interface?

11 responses

Hard to discern what section some text in the desktop dashboard screenshot belongs to which was a bit confusing

The interface need additional work to be more consistent in the away the user uses it and to look professional. The main issues are no safeguards for the deletion buttons and differences in the size of the text boxes and date picker

show the battery levels in graphics

A way of distinguishing the editable part of the node name so it is clear that the user is not able to change the "Node 1:" part for example.

Confirmation for the deletion of nodes to prevent accidental deletion of nodes.

Round temperatures/humidity to 1, the decimal places aren't necessary for the average user at home. Ask people to confirm when deleting a node or climate data so it is harder to accidentally do.

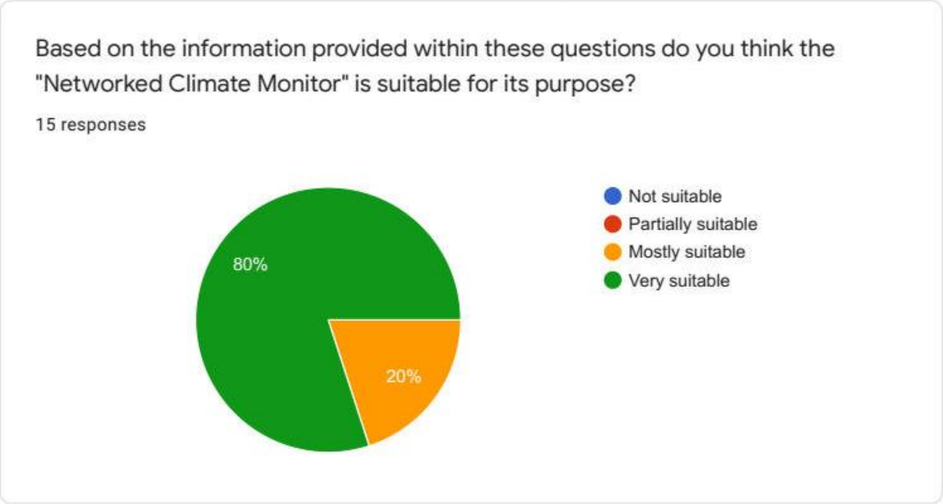
It needs more colour as it has a lot of black on white with a blueish colour only used on important things. So perhaps use of other brighter colours to make it more attractive.

I think a mobile app or making the website looking similar to app on phones would make it easier to use. This could have a bottom bar with buttons for the different pages instead of the current menu

none

No

Not currently from presentation



Based on the information provided within these questions do you think the "Networked Climate Monitor" is suitable for its purpose?

15 responses