UNIVERSIDAD AUTONOMA DE MADRID

Escuela Politécnica Superior

22 | 23

Bachelor thesis

Climate control system for plants in the cloud

Sofía Xiaofan Fernández Marín

Escuela Politécnica Superior
Universidad Autónoma de Madrid
C\Francisco Tomás y Valiente nº 11

Campus Internacional excelencia UAM CSIC+

www.uam.es

UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Bachelor as Bilingual Computer Science

BACHELOR THESIS

Climate control system for plants in the cloud

Author: Sofía Xiaofan Fernández Marín
Advisor: Iván González Martínez

February 2023

Francisco Tomás y Valiente, n^o 1
Madrid, 28049
Spain

Sofía Xiaofan Fernández Marín
Climate control system for plants in the cloud

Sofía Xiaofan Fernández Marín
C\ Francisco Tomás y Valiente Nº 11

PRINTED IN SPAIN

Agradecimientos

Me gustaría expresar mi más sincero agradecimiento a mi tutor, quien ha sido una fuente de apoyo y orientación durante mi Trabajo de Fin de Grado. Sus aportaciones y consejos han sido valiosos y decisivos en el desarrollo de este mismo. También agradecer a los profesores que he tenido a lo largo de mis años en la carrera, ya que han desempeñado un papel fundamental en la enseñanza de conocimientos y habilidades que me han ayudado en este proyecto.

Quiero agradecer a mis padres por su apoyo emocional y económico durante toda la etapa de mis estudios y por ese amor incondiconal que me aportan dia a dia. A mi hermana y hemano, porque su inquebrantable apoyo y fe en mí han sido la motivación detrás de mis logros, académicos y personales.

Por último, quiero agradecer a mis amigos, que me han acompañado y han sido una parte muy importante de mi experiencia universitaria. Su apoyo y compañía han hecho que esta experiencia sea inolvidable, y les agradezco su constante motivación. Gracias por darle vida a mis días en clase y hacer que este proceso sea memorable.

Gracias a todos por vuestro apoyo y por ayudarme a llegar hasta aquí.

Resumen

Este proyecto presenta el diseño y la implementación de un sistema de control climático personal para plantas, que utiliza tecnologías de computación en la nube e Internet de las Cosas (IoT). El sistema se construye utilizando una placa Arduino, sensores de temperatura y luz, y una base de datos basada en la nube para el almacenamiento y análisis de datos. Los sensores recogen datos sobre las condiciones de crecimiento de las plantas y el sistema ajusta las condiciones, como la luz y la temperatura, en función de los datos. El sistema está diseñado para ser rentable y fácil de usar, utilizando componentes de hardware de bajo coste y software de código abierto.

Los resultados de este proyecto demuestran que el sistema es capaz de supervisar y controlar con precisión las condiciones de crecimiento de las plantas, lo que se traduce en una mejora de los rendimientos y un aumento de la eficiencia. Un análisis de los beneficios y desafíos potenciales del uso de la computación en la nube y la tecnología IoT para la jardinería personal mostró que el sistema ofrece un importante ahorro de costes y mejores rendimientos a través de la automatización. Las principales implicaciones de este trabajo destacan la importancia de utilizar las últimas tecnologías en computación en la nube e IoT para la gestión eficiente y eficaz de la jardinería personal.

Palabras clave

Internet of Things (IoT), Cloud Computing, Azure, Climate Control, Database, Arduino, Plants, Greenhouse, MySQL, Service bus, Resource Group, Humidity and Temperature sensor, Water Level Sensor, Photoresistor

Abstract

This project presents the design and implementation of a personal climate control system for plants using cloud computing and Internet of Things (IoT) technologies. The system is built using an Arduino board, temperature and light sensors, and a cloud-based database for data storage and analysis. The sensors collect data on the growing conditions of the plants and the system adjusts the conditions, such as light and temperature, based on the data. The system is designed to be cost-effective and easy to use, utilizing low-cost hardware components and open-source software.

The results of this project show that the system is able to accurately monitor and control the growing conditions of the plants, resulting in improved yields and increased efficiency. An analysis of the potential benefits and challenges of using cloud computing and IoT technology for personal gardening showed that the system offers significant cost savings and improved yields through automation. The major implications of this work highlight the importance of utilizing the latest technologies in cloud computing and IoT for the efficient and effective management of personal gardening.

Keywords

Internet of Things (IoT), Cloud Computing, Azure, Climate Control, Database, Arduino, Plants, Greenhouse, MySQL, Service bus, Resource Group, Humidity and Temperature sensor, Water Level Sensor, Photoresistor

Table of Contents

Lists

List of codes

List of figures

List of tables

1

Introduction

The integration of cloud computing and the Internet of Things (IoT) has the potential to revolutionize the way we manage and control our personal gardening operations. Cloud computing enables the delivery of computing services, such as storage, processing, networking, and software, over the internet, while IoT allows for the connection of physical objects and devices in order to collect and exchange data. Together, these technologies can be used to create a system that can monitor and control the environment of a personal greenhouse in real-time, with the goal of improving plant health and yield.

We build upon the work of previous researchers in the field, addressing issues such as security concerns and vendor lock-in. A comprehensive database of plants and their optimal sowing and growth conditions was created using datasets from data.world and gardenate.com, which was then edited and uploaded to Azure for easy access and management. The Arduino board is also connected to the database through a virtual machine in Azure, allowing for the automation of the climate control system in the greenhouse.

The study was undertaken to provide a practical and effective solution for individuals looking to improve their gardening skills and keep their plants healthy. This project aims to present a working prototype of the proposed personal climate control system for plants in the cloud, and to demonstrate its effectiveness in terms of automation, scalability, and cost savings.

The scope of the project includes the design and implementation of the system, as well as the evaluation of its performance. The study will not include a detailed analysis of the security concerns related to cloud computing and IoT, as it is out of the scope of the project. The final product of this project will be a working prototype of the proposed personal climate control system for plants in the cloud, along with a detailed report on the design, implementation and evaluation of the system.

1.1 Motivation

The idea of this project comes from the fact that I have always liked plants and gardening, despite my repeated struggles with keeping them alive. This project is an attempt to find a solution to this problem and to help others who share my passion for plants. By utilizing the latest technologies in

cloud computing and the Internet of Things (IoT), I aim to create a personal climate control system that can accurately monitor and control the growing conditions of plants in a personal greenhouse. With this system, I hope to improve the success rate of gardening and make it easier for people to grow and care for their plants.

1.2 Goals

One goal of this project is to design and implement a personal climate control system for plants in the cloud, utilizing IoT and cloud computing technologies. The system will consist of temperature sensors placed in a personal greenhouse, an Arduino board for controlling the sensors and sending data to the cloud, and a cloud-based database for storing and analyzing the data. The system will also include a feedback mechanism for controlling environmental conditions such as light and temperature based on the data collected from the sensors.

Another goal is to create a personal climate control system for plants in the cloud, using the latest technologies in cloud computing and the Internet of Things (IoT). By integrating these technologies, we aim to provide a system that can accurately monitor and control the growing conditions of plants in a personal greenhouse.

The system will be built using an Arduino board as the main controller, with temperature and light sensors placed in various locations throughout the growing area. The temperature sensor will collect data and send it to a cloud-based database, where it will be analyzed and used to make decisions about adjusting the growing conditions. For example, if the temperature is too high, the system will automatically open blinds or turn on fans to cool the plants down.

The system will also be designed to be easily expandable, allowing for the addition of new sensors and devices as needed. Additionally, the system will be able to provide detailed data and insights into the growing conditions, which can be used to optimize the growth of plants and improve yields.

One of the key challenges of this project is to create a system that is both cost-effective and easy to use. To achieve this, we will be using low-cost and widely available hardware components, such as the Arduino board, and open-source software. We will also be using a cloud-based database, which will allow us to easily store and analyze large amounts of data.

In addition to the technical aspects of the project, we will also be conducting a thorough analysis of the potential benefits and challenges of using cloud computing and IoT technology for personal gardening.

1.3 Document Structure

- **State-Of-The-Art:** In this section we will talk about the current state of the technologies used and the climate control system for plants.
- **Design of the system:** This second section describes the design of the control system that has been built by us. Includes a detailed explanation of the architecture of the connection with the cloud computing system. The section also describes the specific components of the system, such as the Arduino board, temperature and light sensors, and the feedback mechanism for controlling environmental conditions.
- **Dissertation:** This section provides a description of the creation and management of the database in cloud computing, as well as how the hardware and the software of the Arduino were built.
- **Results and discussion:** In this section we talk about the implementation and testing of our system. This includes a detailed analysis of the data collected from the temperature sensors, as well as a discussion of the system's performance and effectiveness. An example of a user interface is also shown.
- **Conclusions and Future work:** The final section provides a conclusion of the project, highlighting how the goals were achieved and the areas for improvement. This includes suggestions for future work, such as the addition of new sensors and devices.

Overall, this document provides a comprehensive overview of the design and implementation of a personal climate control system for plants, highlighting the potential benefits and challenges of using cloud computing and IoT technology for a personal greenhouse.

2

State-Of-The-Art

This chapter gives information about the context we will be working on and proper acknowledgement of previous work. We will be talking about cloud computing and the internet of things, the two most important technologies used in this project. After explaining this, a description of previous work related to this project will be given.

2.1 Cloud Computing

Cloud computing is a technology that allows for the delivery of computing services, such as storage, processing, networking, and software, over the internet. It enables users to access and use these resources on demand, without the need to purchase and maintain their own physical infrastructure. It has a long history, dating back to the 1950s with the advent of mainframe computers. However, it wasn't until the early 2000s that cloud computing began to gain significant attention, with the publication of core papers by Google in 2003 and the commercialization of Amazon's EC2 service in 2006. [1]

Cloud computing offers several benefits, including increased flexibility, scalability, and cost savings. It allows businesses to pay only for the resources they use, rather than having to invest in and maintain their own infrastructure. It also makes it easier for organizations to scale their operations up or down as needed, without the need to invest in additional hardware or software. However, there are also some potential downsides to cloud computing, including security concerns and vendor lock-in. It is important for organizations to carefully evaluate their needs and consider the potential risks before moving to the cloud. [1]

There are 3 types of cloud computing services, Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). [2] [3]

- **IaaS**, where the customers can access compute resources or storage. It allows you to [4].
- **PaaS**, a third-party provider makes available some platforms of its infrastructure for the public to use it. It is mostly used for developing applications using built-in SW components or mining data [5].
- **SaaS** or web services, a third-party provider host the application that the public will use. A good example of this is how we use our e-mail web page, where we get the mails from anywhere, and they are stored on the service provider network [6].

In figure 2.1 there is a diagram of how these services are arranged.

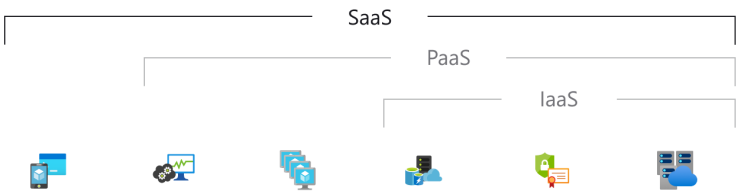


Figure 2.1: Diagram of azure computing services [7]

2.2 Internet of Things (IoT)

The Internet of Things (IoT) refers to the network of physical objects, devices, and appliances that are embedded with sensors, software, and network connectivity, allowing them to collect and exchange data. These connected devices can be found in a variety of settings, including industrial, social, healthcare, and infrastructure applications. [8]

Enabling technologies for the IoT include identification and tracking technologies such as RFID (radio-frequency identification) and WSN (wireless sensor networks), which allow devices to be uniquely identified and tracked. Integration of WSN and RFID technologies is important for creating a cohesive and efficient system. Communications technologies, including cellular networks, Wi-Fi, and Bluetooth, allow connected devices to communicate with each other and with central servers. Network technologies such as IP (Internet Protocol) and cloud computing are also key for connecting devices and facilitating data exchange. Service management technologies are necessary for organizing and managing the various connected devices and ensuring that they are operating efficiently. [9]

2.3 IoT in Cloud Computing

With the exponential growth of the IoT, there is a vast volume of data being produced by multiple outlets. It is not practical to store all of this data locally on IoT devices, as these devices often have strict constraints on energy and storage space. However, IoT networks can enable outsourced data collection and cloud storage, allowing for the storage of large amounts of data without being limited by the resource constraints of the IoT devices. However, IoT systems can be complex in design and have limited capacity for storage and retrieval. The integration of cloud computing with the IoT can offer numerous benefits to a wide range of IoT applications as: [10]

- **Data storage and analysis:** The vast amounts of data generated by IoT devices can be collected and stored in the cloud, allowing for centralized data management and analysis. This can help organizations to gain insights from their data and make data-driven decisions.

- **Data processing:** Cloud computing can also be used to process and analyze large amounts of data in real-time, providing the necessary computing power for tasks such as machine learning and predictive analytics.
- **Scalability:** The on-demand nature of cloud computing allows organizations to scale their IoT systems up or down as needed, without the need to invest in additional hardware or infrastructure.
- **Security:** Cloud-based IoT systems can also benefit from the security measures put in place by cloud service providers, such as encryption and secure data storage.
- **Cost savings:** By using cloud computing, organizations can save on the costs of purchasing and maintaining their own physical infrastructure.

Overall, the integration of cloud computing with the IoT can enable organizations to gain new insights from their data, improve their operations, and reduce costs. However, it is important to carefully consider the potential challenges and risks involved in such integration and to take steps to address these issues. [11] [12]

2.4 Models of a climate control system for plants in the cloud in the present

In this section, we will discuss the existing models of a climate control system for plants in the cloud that currently exist in the present. We will not describe these models too thoroughly, but we will be highlighting their features and capabilities. This information will provide a background and context for our proposed system and will help us to understand how our system compares to existing solutions.

2.4.1 Active plant wall for green indoor climate based on cloud and Internet of Things

This project consists of a remote monitoring control system of a plant wall, intending to have a non-contaminated indoor environment. This is achieved thanks to a system built around an Arduino Uno, using sensors such as temperature and humidity, light, CO_2, and more components. They used Azure as a cloud platform because it provides IoT Hub, very useful as it lets you connect almost any IoT device. They achieve this thanks to the microprocessor Intel Edison. [13]

2.4.2 Camponectado

Camponectado monitors and automatizes crops in real-time. The sensors are set on the ground and these send signals to the microcontroller (Arduino) that sends the information to the web. It can send the information by wire or wireless and the information will be stored on the cloud. This system can measure temperature, humidity, luminosity, UV radiation, and a lot of things more. [14]

2.4.3 Postscapes

This system remotely controls and monitors a greenhouse or other agriculture environment facility. It gathers data and uses it to control ventilation, cooling, heating, and irrigation in order to have a more productive growing environment. The system can also alert the user about extreme temperatures or pests. This system improves labour efficiency and reduces energy consumption. [15]

2.5 Database used

The dataset that we used was separated first into two different files. It was taken from the web page data.world, and in turn, taken from the web page [16, gardenate.com]. The DB we will focus on will be [17, What to plant and when to plant it]. This DB was given in two Excel files, named as two different zones in the USA [18], Zone 8a, which is mostly a humid subtropical climate zone, but also crosses over areas of Mediterranean climate, mid-latitude desert and semiarid steppe and Zone 10a, which is mostly a humid subtropical climate zone but also crosses over areas of Mediterranean and mid-latitude desert climates [19].

We wanted it to be only a file, to facilitate the search afterwards, we merged these two zones as there is not a problem at all because they both share similar climate characteristics. Both zones are characterized by Mediterranean, subtropical, mid-latitude desert, and marine coast climates, which would make them well-suited for similar types of plants and gardening activities. Furthermore, both zones also cross over with other climate regions, such as humid subtropical and semiarid steppe, which would further increase the similarity in plant hardiness and gardening practices between the two zones.

It would also fit Spain's climate. Spain has a diverse climate, with regions that can be Mediterranean, semi-arid, or even oceanic, depending on the location. While Spain and the USA have different climates overall, the Mediterranean, subtropical, mid-latitude desert, and marine coast climates found in USA_zone_8a and USA_zone_10a would likely have similarities to some regions in Spain. However, it's not a match for all of Spain, and would still be specific regions within Spain that share similar climates with these zones. [20]

3

Design of the system

In this chapter, we will focus on the materials, procedure, and theory behind the design and implementation of the climate control system for plants in the cloud. We will go through the different options for building the system and the factors that were considered when making decisions about the materials and components used. Specifically, we will discuss the different boards available for the project and the pros and cons of using each one. We will also describe the specific board that was chosen for the project and the reasoning behind the decision. Additionally, this section will also provide an overview of the procedure and theory behind the system, including the installation and use of sensors and the process of sending data to the cloud-based database. Overall, this section aims to provide a detailed understanding of the materials and methods used in the design and implementation of the system.

We will also be providing detailed system diagrams that will help to visualize the overall structure and components of the climate control system for plants in the cloud. These diagrams will include information on the placement and connections of the sensors section 3.3, the Arduino board, and the cloud-based database (section 3.3), as well as any additional components that are necessary for the proper functioning of the system. This will provide a clear understanding of how all the different parts of the system work together to achieve the goal of automating the management and control of a greenhouse or field for plants.

3.1 High-Level diagram of the system

At figure 3.1 we can see a high-level diagram of the system. From left to right, we have the breadboard with all its components and the Arduino on a greenhouse. The sensors will measure the metrics of the field and those will be sent to the personal computer via Arduino. After the personal computer has received the measurements, it will connect to Azure, check some columns and the cloud will return an action that the Arduino has to do. It will be sent through the personal computer.

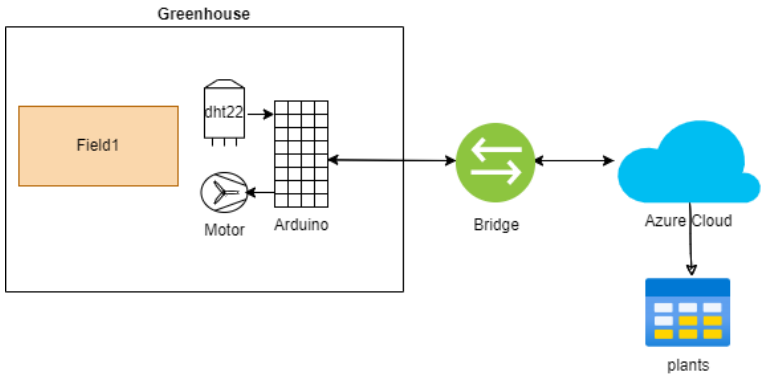


Figure 3.1: Diagram of the High-level system

3.2 Description of the components

Here we will describe all the possibilities we had for building the system.

3.2.1 Boards

The Board is a crucial part of the system. It has to control the whole hardware structure part, controlling temperature sensors, each one of them in a different part of the greenhouse, and then, sending the red variables to the cloud.

For this part, we had to take into account the prices of the boards in the market, it had to be a non-expensive board. We found out that two of the most popular manufacturers were Arduino and Raspberry, among others [21].

Arduino

Arduino is an open-source microcontroller, which means that any company is allowed to create compatible boards [22]. A microcontroller is a set of CPU (central processing unit), I/O ports (input/output), memory and peripherals (like timers or counters). An embedded microcomputer that performs specific tasks [23].

Arduino was created in 2005, by Massimo Banzi et al., with the goal of making this type of technology available to everyone. And as microcontrollers, the projects that these boards are used for, are usually performing repetitive instructions [24]. It has gained popularity as it can upload a new code with only the use of a USB cable, instead of the need of having to have another hardware piece. But not only because of this, but also because it is very easy to program and you have a lot of packages and code as it is open-source, so the source code is available for free, so anyone can use them for making their own board. [25] This is why there is a lot of different hardware and software created by different people all around the world.

The most popular one is the Arduino Uno, the one we chose, described better in section 3.2.1.

Raspberry Pi

Unlike Arduino, Raspberry Pi is a microprocessor. The main difference between a microprocessor and a microcontroller is that the former has a more powerful CPU and can do more complex operations [26]. Raspberry Pi can have many peripherals and can run as a single computer, that can even run a Linux distribution. It was first created by Eben Christopher Upton, in the UK in 2012, and as it has a more powerful CPU, it can be used for more than just repetitive instructions, like playing games or operating robots. The downside is that Raspberry Pi does not have open-source hardware, but it has open-source software, so you can also find various projects to make. [24]

Final decision

The main goal of the project is to have various temperature sensors, each one of them in a different part of the greenhouse, so we finally decided to go with the former board. The work needed to be done is not so big, as it only reads the temperature from some DHT22 sensors. It is possible to code the instructions for an Arduino. If we needed to implement more features than reading the temperature, we would need a Raspberry Pi or add more shields to the Arduino. It is also a good idea to work with an Arduino board as it counts with more I/O ports, so if we want to expand the greenhouse, we will only need to put in another sensor.

For the Arduino board, we chose one from the ”Elegoo” manufacturer. It is a board that we already had in our house, as it was bought a time ago, and it was compatible with Arduino. As we have to say, Arduino is open-source so that anyone can create hardware. Elegoo is one of them. It was born with the idea of allowing everyone to have electronic hardware to create. [27]

The board we will use in this project is called ”Elegoo Uno R3”, where you have to install the Arduino drivers and software and it works as a ”Arduino Uno R3” board. The specifications are almost the same, it uses the ATMEGA328 microcontroller (8-bit), and has 6 analogue and 14 digital I/O pins with another 6 pins for powering a breadboard or the circuit you build. The connection to the power source can be via a USB cable or a wall power supply and comes with a led light for checking this. [28] [29]

3.2.2 Cloud Computing

Our first idea was to have our database in the cloud, public or private. The public one can be used for anyone, they can be free or paying on-demand depending on the usage [30]. The private one is usually used by certain corporations and is more customized to the needs of that certain company with the plus of having more privacy and security [31]. The one we could use was the public one, so we looked through some options in this category.

The two most popular cloud computing services are Azure and Amazon Web Services (AWS). Our use of the cloud was going to be basic, so we did not consider the additional products offered by AWS. Although AWS was cheaper, we chose Azure because we had a free year with our UAM email [32] [33]. Also, according to a study done by the university of Napoli, the performance (the use of the computing resources efficiently) is similar in both of them, and we can see low values in the latency variability (low interval between request and response), also in Azure and AWS. [34]

With all this in mind, we based our decision on the things that were going to be more used, the things that were taken into account for deciding, where: the knowledge that the database that we were going to use was not large and the most important thing, we wanted it to be connected with the Arduino. And because of this, the chosen cloud database (a database with cloud computing benefits) should not be a problem, as we only need the basic features: Access the database anywhere and anytime, no need to have special hardware for hosting the database. There is also better scalability, security and accessibility. There are other benefits available for companies, not discussed here, as it is not relevant to our use [35] [36]. The connection with Arduino depends on the Internet of Things (IoT) platform, defined as:

”An open and comprehensive network of intelligent objects that can auto-organize, share information, data and resources, reacting and acting in face of situations and changes in the environment”. [37, Internet of things (IoT): A literature review, S. Madakam, et al.]

The IoT platform choice depends on the users. AWS is the best one in when market share and has more hubs but Azure has more tools and stands out in data visualization tools: PowerBI. According to the study [38, Comparison of the IoT platform vendors, Microsoft Azure, Amazon web services, and Google cloud, from users’ perspectives, Ucuz, Derya and others], AWS is the best one, but as said before, we are not using market share (not companies), it is not relevant, so at the end, because of the PowerBI tool we can use and as we have the free trial because of the UAM account, we will be using Azure.

3.3 Low-Level diagram of the system

System diagram

Components

1.– **Azure portal:** Azure portal, interface for managing Azure resources.

2.– **myResourceGroup:** Azure resource group, a logical container for resources in Azure. It consistently allows the management of resources and all the resources inside this group can be managed together and deleted.

3.– **mysqlserver-tfg2022:** An Azure SQLserver relational database management system that will store a SQL database.

4.– **plants:** An Azure SQLdatabase stored in mysqlserver-tfg2022 and that will be storing the tables ”plants_db”

and ”my_plants”.

5.– **plants_db and my_plants:** Azure NoSQL tables stored in plants and linked by the column ”Name” as Primary Key (PK) and Foreign Key (FK).
More in section 4.1.

6.– **myVM:** Azure virtual machine, a software emulation of a physical computer. It will allow us to send data from the Arduino device.

7.– **arduinodb_cloud.py:** File on the cloud, it allows connecting to the personal computer using a bus (servicebus_client).
More in section 4.3.

8.– **servicebus_client:** Azure Service Bus Namespace, a container for the queues we will use.

9.– **myqueue and myqueue2:** Service Bus Queues, messaging services that enable sending and receiving messages between different components.
More in section 4.3

10.– **Personal Computer:** User’s personal computer.

11.– **arduinodb_pc.py:** File on the personal computer, it allows connecting to the cloud using a bus (servicebus_client).
More in section 4.3

12.– **database.py** File on the personal computer that connects with azure SQLserver. More in section 4.2.

13.– **gui.py** Creates the graphical user interface (GUI) that the user will use, utilizing ”database.py”. More in section 5.1.1.

14.– **cpp-py.ino:** File on the Arduino device that allows it to connect with the computer. More in section 4.2.

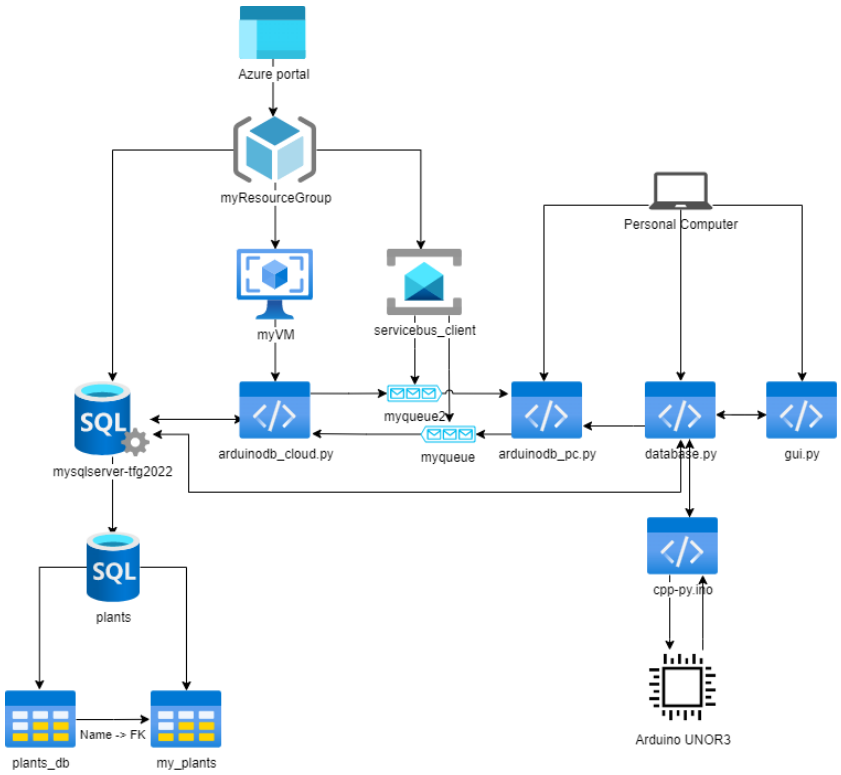


Figure 3.2: Diagram of the system

Arduino connections diagram

Components

1.– **Arduino Uno:** Microcontroller board based on the ATmega328P. It has 14 digital input/output pins, 6 analogue inputs, a 16 MHz quartz crystal, a USB connection, a power jack, an ICSP header, and a reset button. It is designed to be used with the Arduino software development environment. [29]

2.– **DHT22 Temperature and Humidity Module:** Digital sensor that measures temperature and humidity. It uses built-in sensors to measure temperature and humidity. It measures temperature between -40℃ to 80℃ with an accuracy of ±0.5℃ and humidity between 0-100% with an accuracy of ±2-5%. It requires a pull-up resistor of around 10K Ohm, already build-in. [39]

3.– **3-6V Motor and Fan Blade:** Motor that can be used to drive a fan blade. The voltage it needs is between 3 and 6V and can be controlled using a microcontroller or other electronic device. [40]

4.– **Water Level Detection Sensor Module:** Detects the level of water in a container. It uses a probe immersed in the water and a circuit that can detect changes in the probe's resistance as the water level changes. [41]

5.– **Passive Buzzer:** Generates sound when it is supplied with an alternating current of the appropriate frequency. It is passive because it does not have an active electronic oscillator circuit inside, thus it needs an external alternating current to produce sound. [42]

6.– **Photoresistor (Photocell):** Type of resistor whose resistance changes based on the amount of light that it is exposed to. [43]

7.– **Red LED:** Light-emitting diode (LED) that emits red light when a current is passed through it. [44]

8.– **Blue LED:** Light-emitting diode (LED) that emits blue light when a current is passed through it. It [44]

9.– **Diode Rectifier:** Diode that is used to convert alternating current (AC) to direct current (DC). It allows current to flow in one direction and is commonly used in power supplies to convert AC input to DC output. [45]

10.– **NPN Transistor PN2222:** Bipolar junction transistor (BJT) that is commonly used as an amplifier or switch. It is an NPN-type transistor whose maximum collector-emitter voltage is 40V, and its maximum collector current is 800mA. [46]

11.– **Resistors:** Component that resists the flow of electric current, in this case, values of 100, 220 and 10K Ohm [47].

With the components mentioned before, our idea was to create different units that would control different parts of the system:

- **Light Detection Unit:** With the Photoresistor, simulate we had a blind we can put down and up. This blind would be represented as LED lights, the blue one, if ON, would represent opening the blinds and the red LED, if ON, closing the blinds. In the future, this would be replaced by a motor. Photo of the breadboard in figure 4.4(b).
- **Water Level Unit:** Using the Water Level Detection Sensor, if the water level of our tank is half empty, it will send a signal to turn on an alarm (buzzer). Photo of the breadboard in figure 4.4(c).
- **Temperature and Humidity Unit:** In the project, we will be focusing the *automatizing part on this module one*. This will be our main part. With the DHT22, we will measure the temperature, as explained in section 1.2. Photo of the breadboard in figure 4.4(a).

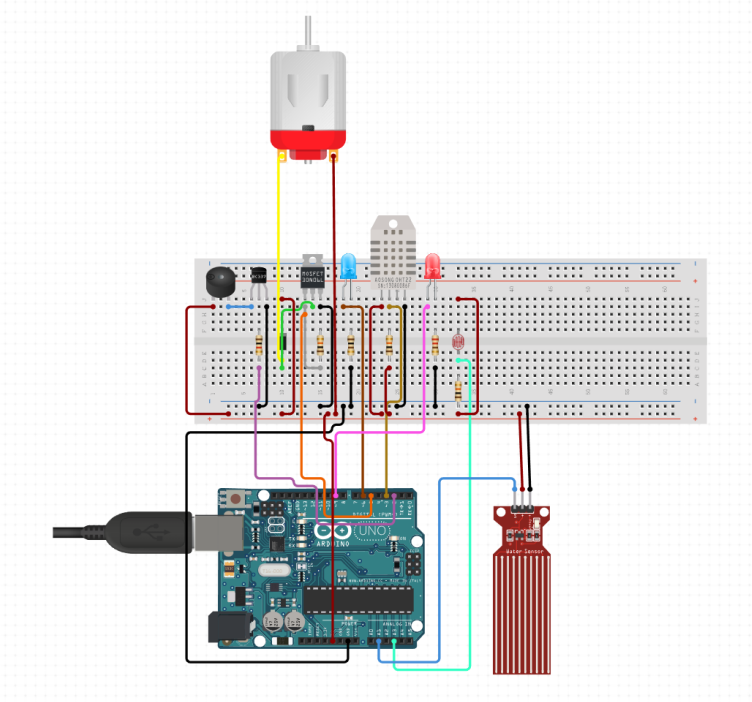


Figure 3.3: Diagram of the Arduino connections with the breadboard and components created in [48, circuito.io].

4

Dissertation

In this section, we describe the creation and management of the database for our proposed IoT and cloud computing solution for climate control. We used a dataset from data.world, which was separated into two Excel files for the sowing month. We then used a VBA script to convert the Excel files into CSV format and combined the two-zone files using Python code. We also merged this dataset with another CSV from the same web page, which had more information about the plants. We cleaned the file by deleting unnecessary information and reorganizing the columns. The final database included information such as sowing month, ease of growth, and minimum and maximum temperature range for each plant. We then uploaded the database to Azure, using Azure Data Studio and creating a new resource group and server. We also built an Arduino board to measure light and temperature and connected it to the database using a virtual machine in Azure and Service Bus. This allows for real-time monitoring and automation of the climate control system in the greenhouse.

4.1 Creation of the database

The used dataset was separated into two different files. It was taken from the web page data.world, and in turn, taken from gardenate.com.

The first DB used was [17, What to plant and when to plant it]. This DB was given in two Excel files, files USA_zone_8a and USA_zone_10a. These zones are zones with a Mediterranean, subtropical, midlatitude desert and marine coast climate [16], so it would fit Spain's climate. The data were sorted according to the month in which it can be sow. This arrangement could e seen as each month was a different sheet of the Excel file and each sheet had around 5 to 50 plants, depending on the season.

As there was no interest in having an excel file, we created a VBA script listing 4.1 for creating a CSV for all the sheets in each file.

After having the datasets we needed, we created some Python code listing 4.2 for merging the different sheets of the excel file (files 8a and 10a). We had the result of two files, with two different zones 8a and 10a, and we merge them with another script.

Code 4.1: VB for creating CSV

```
Public Sub SaveWorksheetsAsCsv()
Dim xWs As Worksheet, xDir As String, folder As FileDialog
Set folder = Application.FileDialog(msoFileDialogFolderPicker)
If folder.Show <> -1 Then Exit Sub
  xDir = folder.SelectedItems(1)
  For Each xWs In Application.ActiveWorkbook.Worksheets
  xWs.SaveAs xDir & "\" & xWs.Name, xlCSV
Next
End Sub
```

Code 4.2: Script for merging the different sheets of the Excel file in one file

```
import pandas as pd

# Create the dataframe we will use later
merged = pd.DataFrame()
# As there are 12 files in total, we create a loop
for file in range(1, 13):

    # Read the data from each file
    month_data = pd.read_csv(f"{file}.csv")

    # Get the name and delete the null rows
    name = list(month_data["Name"])
    name = [x for x in name if not(pd.isnull(x)) == True]

    # Get the month you have to sow each plant. Depends on the file it is in
    sowMonth = []
    sowMonth.extend([file]*len(name))

    # Create a dicitonary to append it to the file
    data = {"Name": name,
            "sowMonth": sowMonth}
    merged = merged.append(pd.DataFrame(data))

# Create file
merged.to_csv("merged_Xa.csv")
```

Following the design of the DB, having a CSV with the combination of the sowing month, we merge it with the CSV [49, Vegetables, herbs, and edible flowers], as it had more information about the plants than the first database. The first resource had only the columns ”Name” and the instructions to sow it, and the second one had the following columns: name and alternative names (**Name**, **alternateName**), how you should sow the specific plant (**sowInstructions**, **spaceInstructions**, **compatiblePlants**, **avoidInstructions**) and how to harvest it (**harvestInstructions**). There is also a column on how to preserve and some hints after you harvest it (**culinaryHints**, **culinaryPreservation**) with an URL you can follow for more information (**url**).

For cleaning the file, the information repeated or not needed was deleted and columns were reorganized. Now the final file looks like the following: It has the same columns as the database [49] plus the month it is recommended to sow (**sowMonth*****), if it is easy to grow (**easyGrow**), and the minimum and maximum temperatures the plant can handle (**minTinF**, **maxTinF**).

We did this as we wanted to have the current month and plants that anyone could sow because they are easy to grow. We also were interested in having the comfort temperature of each one, as we had our thermometer and we wanted to get which ones are pleasant or if we need to lower or rising the temperature.

After doing this, we uploaded the database into Azure, using Azure Data Studio and importing work items. Before that we had to create a new resource group called ”myResourceGroup”, using the student subscription, and a server called ”mysqlserver-tfg2022”, as it is needed to upload the DB there.

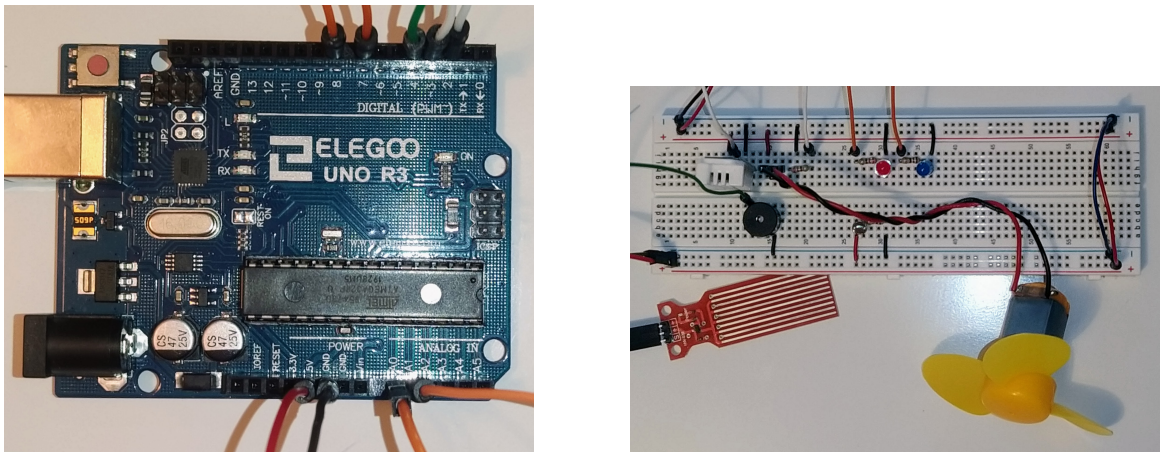
We called the main Database ”plants” in the figure 4.1, is just shown the columns: Name, alternateName, sowInstructions, spaceInstructionsInches, compatiblePlants, culinaryHints, sowMonthJan, easyGrow, minTinF, maxTinF for easier reading of the table. We also had to create another database with the plants we were supposed to have, as we owned only a cauliflower.

	Name	alter…	sowInstructions	space…	compatiblePlants	culinaryHints	sowMonthJan	easyGrow	minTinF	maxTinF
1	Amaranth	Love-lies…	Sow in garden. Sow seed at a depth a…	20	Onionsor cornor peppersor eg…	Both leaves and seeds can be…	1	0	64	86
2	Angelica	*NULL*	Sow in garden. Sow seed at a depth a…	18	Any herbs that like dampor s…	The stems can be candied and…	0	1	50	77
3	Artichokes (Glob…	*NULL*	Sow in garden. Sow seed at a depth a…	71	Needs a lot of space. Best i…	Pick buds before scales deve…	0	1	59	64
4	Asparagus	*NULL*	Plant as crowns.	12	Parsleyor Basilor Nasturtium…	Steaming is traditionalor th…	1	1	61	86
5	Asparagus Pea	Winged be…	Sow in garden. Sow seed at a depth a…	9	Best grown in separate bed	Cook quickly by steaming and…	0	1	59	68
6	Basil	*NULL*	Grow in seed traysor and plant out i…	9	Tomato	Basil is commonly used fresh…	0	0	64	95
7	Beetroot	Beets	Sow in garden. Sow seed at a depth a…	10	Onionsor Silverbeet (Swiss C…	Apart from boiling whole for…	0	1	45	77
8	Borage	Burrageor…	Sow in garden. Sow seed at a depth a…	8	Strawberryor tomatoesor zucc…	Has a slight cucumber taste …	1	1	50	77
9	Broad beans	Fava bean	Sow in garden. Sow seed at a depth a…	8	Dillor Potatoes	The fresh beans are eaten st…	0	1	43	75
10	Broccoli	*NULL*	Grow in seed traysor and plant out i…	17	Dwarf (bush) beansor beetsor…	The stem (peeled)or leavesor…	0	1	45	86
11	Brussels sprouts	*NULL*	Grow in seed traysor and plant out i…	21	Dwarf (bush) beansor beetsor…	Remove any discoloured outer…	0	0	45	86
12	Burdock	Gobo (Jap…	Sow in garden. Sow seed at a depth a…	24	Best grown in separate bed.	Harvest in the first year wh…	*NULL*	1	50	68
13	Cabbage	*NULL*	Grow in seed traysor and plant out i…	25	Dwarf (bush) beansor beetsor…	Young spring cabbage can be …	0	1	41	64
14	Cape Gooseberry	Golden Be…	Sow in garden. Sow seed at a depth a…	20	Will happily grow in a flowe…	The berry is the size of a c…	0	1	50	77
15	Capsicum	Bell pepp…	Grow in seed traysor and plant out i…	14	Egg plant (Aubergine)or Nast…	Can be sliced and seeded and…	1	0	64	95
16	Cardoon	*NULL*	Grow in seed traysor and plant out i…	39.5	Best grown in separate bed.	Cut off the base and leaveso…	0	0	55	77
17	Carrot	*NULL*	Sow in garden. Sow seed at a depth a…	7	Onionsor Leeksor Lettuceor S…	Steamed or raw carrots are t…	1	1	46	86
18	Cauliflower	*NULL*	Grow in seed traysor and plant out i…	31.5	Dwarf (bush) beansor beetsor…	Cauliflower can be steamed. …	0	0	50	86
19	Celeriac	*NULL*	Grow in seed traysor and plant out i…	24.5	Beansor brassicasor carrotso…	Cook wholeor scrubbed and pe…	1	0	46	70
20	Celery	*NULL*	Grow in seed traysor and plant out i…	9	Not applicable as celery nee…	Chop and use raw in salad or…	1	0	54	70
21	Chicory	Witloofor…	Sow in garden. Sow seed at a depth a…	11	Florence fennelor tomatoes.	Good in salads. Grill lightl…	0	0	50	68
22	Chilli peppers	Hot peppe…	Grow in seed traysor and plant out i…	18	Best grown in a separate bed…	Washor dryor and free whole.…	0	0	64	95
23	Chinese cabbage	Wong boko…	Sow direct in the garden. Sow seed a…	12	Dwarf (bush) beansor beetsor…	Use in stir-fry . Has a mild…	0	1	50	68
24	Chives	Garden ch…	Sow in garden. Sow seed at a depth a…	2	Tomatoesor Parsleyor Apples	Use raw in salads or as a mi…	0	1	50	86
25	Choko/Chayote	Chayote s…	Plant whole mature fruit when one pr…	39	ucumbers	Chokos can be peeled and cho…	0	1	59	86
26	Climbing beans	Pole bean…	Sow in garden. Sow seed at a depth a…	6	Sweetcornor spinachor lettuc…	Use young in salads - blanch…	1	1	61	86
27	Collards	Collard g…	Grow in seed traysor and plant out i…	18	Dwarf (bush) beansor beetsor…	Slice and steam or use in st…	1	1	46	86
28	Coriander	Cilantroo…	Sow in garden. Sow seed at a depth a…	18	Dillor Chervilor Aniseor Cab…	Use the leaves to flavour ho…	1	1	50	77

Figure 4.1: Main Database

4.2 Building the Arduino

For building the Arduino, we first decided what we wanted it to do. We wanted it to measure the light and temperature and, if needed, put down/up some blinds or turn on/off a fan. We also wanted to measure the water level in a tank we would have for watering the plants. It will turn on an alarm if it detects it has less than half of the water. A general view of the system can be seen in figure 4.2.



(a) Top view of the Arduino.

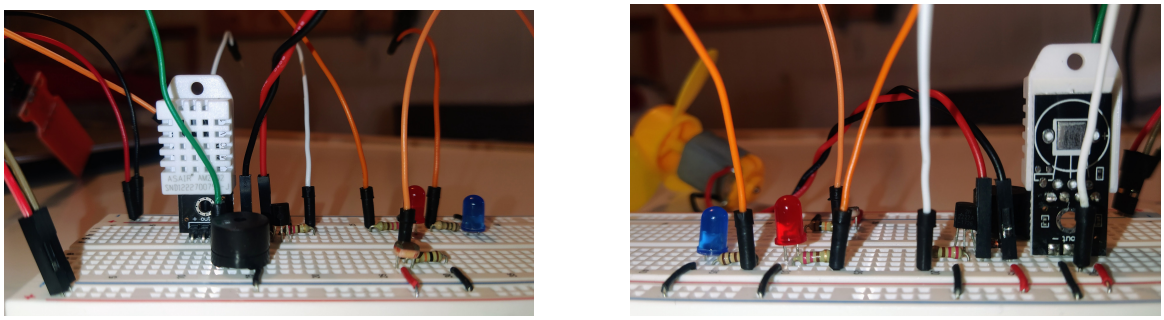
(b) Top view of the Breadboard.

Figure 4.2: Top view of the Arduino and Breadboard.

The connections of the Arduino shield were declared in the file ”cpp-py.ino”, listing 4.3.

Code 4.3: Ports declarations

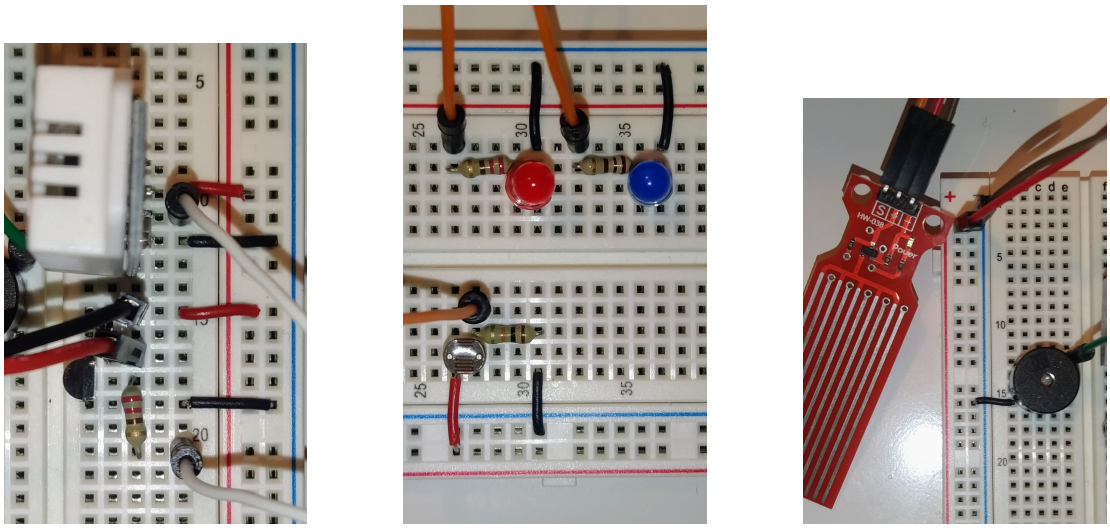
```
1  // DHT declaration
2  #include "DHT.h"
3  #define DHTPIN 2 // Digital pin 2 connected to the DHT sensor
4  #define DHTTYPE DHT22 // Sensor type
5  DHT dht(DHTPIN, DHTTYPE); // Initialize DHT sensor.
6  #define FANPIN 3 // Digital pin 3 connected to Fan motor
7
8  // Water level detection sensor module declaration
9  #define SENSORPIN A0 // Analog pin 0 connected to water level sensor
10 #define BUZZER 4 // Digital pin 4 connected to buzzer
11
12 // Photo sensor module declaration
13 #define PHOTOPIN A1 // Analog pin 1 at Photoresistor
14 #define REDLED 7 // Led pin at Arduino pin 7
15 #define BLUELED 8 // Led pin at Arduino pin 8
```



(a) Front view of the Arduino.

(b) Rear view of the Breadboard.

Figure 4.3: Front and back view of the Breadboard.



(a) Temperature and Humidity Unit top view on breadboard.

(b) Light Detection Unit top view on breadboard.

(c) Water Supply Unit top view on breadboard.

Figure 4.4: Units top view on breadboard.

Code 4.4: Temperature and Humidity Unit

```
33  void loop() {
34    while (!Serial.available());
35    String str(Serial.readString());
36    char str_array[str.length()+1];
37    str.toCharArray(str_array, str.length()+1);
38
39    if (strcmp(str_array, "readT") == 0) {
40      Serial.print(dht.readTemperature(true));
41      Serial.print(",");
42      Serial.print(dht.readTemperature());
43    }
44
45    if (strcmp(str_array, "hot") == 0) {
46      analogWrite(FANPIN, 150);
47      digitalWrite(BLUELED, LOW); // Stop turning up blinds
48      digitalWrite(REDLED, HIGH); // Turn down blinds
49    } else if (strcmp(str_array, "cold") == 0) { // If cold, open blinds
50      analogWrite(FANPIN, 0);
51      digitalWrite(REDLED, LOW); // Stop turning up blinds
52      digitalWrite(BLUELED, HIGH); // Turn up blinds
53    } else {
```

Code 4.5: Light Detection Unit

```
53    } else {
54      int light = analogRead(PHOTOPIN);
55      if (light > 900){ // Its bright AND not cold, red led on -> Close blinds
56        digitalWrite(BLUELED, LOW); // Stop turning up blinds
57        digitalWrite(REDLED, HIGH); // Turn down blinds
58      } else if (light < 500){ // Its dark, blue led on -> Open blinds
59        digitalWrite(REDLED, LOW); // Stop turning up blinds
60        digitalWrite(BLUELED, HIGH); // Turn up blinds
61      } else {
62        digitalWrite(REDLED, LOW); // Stop turning up blinds
63        digitalWrite(BLUELED, LOW); // Stop turning up blinds
64      }
65    }
```

Code 4.6: Water Supply Unit

```
67    // Check water level, if less than half, an alarm will turn on
68    int water = analogRead(SENSORPIN);
69    if ( water < 255){ tone(BUZZER, 2500); }// Send 1KHz sound signal...
70    else { noTone(BUZZER); }
71  }
```

4.2.1 Connecting the Arduino with the Cloud.

For connecting the Arduino with the database, we created a file called ”database.py”, we are showing the main code lines, where in listing 4.7 the python script gets the temperature from the Arduino, the listing 4.8, shows how we send a signal after reading checking temperature in the cloud.

Code 4.7: Gets the temperature from the Arduino

```
1  def arduino_read_T(arduino, unit):
2      x = datetime.datetime.now()
3      arduino.write(bytes("readT", 'utf-8'))
4      time.sleep(0.05)
5      ...
6      value = arduino.readline()
7      value = str(value).lstrip("b'")
8      value = str(value).rstrip("'")
9      f, c = str(value).split(",")
10     if unit == 1: txt_currentT = (f"Today, {x.day}/{x.month}/{x.year} at
           {x.hour}:{x.minute}:{x.second} it's {f} [U+FFFD] F")
11     elif unit == 0: txt_currentT = (f"Today, {x.day}/{x.month}/{x.year} at
           {x.hour}:{x.minute}:{x.second} it's {c} [U+FFFD] C")
12     else: txt_currentT = ""
13     return float(f), float(c), txt_currentT
```

Code 4.8: Sends signal to Arduino after reading temperature

```
1  def arduino_write_cold(arduino):
2      arduino.write(bytes("cold", 'utf-8'))
3
4  def arduino_write_hot(arduino):
5      arduino.write(bytes("hot", 'utf-8'))
```

4.3 Connecting the Personal Computer with the Database in the Cloud

As we wanted to automatize the project, our idea was to put down/up some blinds or turn on/off a fan. For that, we needed to check every certain amount of time, what the temperature and light there was on the greenhouse. After this, send this information to the database and wait for it to send us a response.

Among all the methods there are for making the connection between the Arduino and the DB, this one was made using a virtual machine in Azure, and setting up a Service Bus queue on the Azure VM using the Azure portal. This was because it was the method recommended and the simplest one, as

another method we saw was to create HTTP requests or use other cloud services like Azure IoT hub.

Azure IoT Hub is designed to work with IoT devices, and it is suitable for projects that involve large numbers of IoT devices, and it's useful for projects that need to handle large amounts of data and perform real-time data processing. On the other hand, Azure Service Bus queues are suitable for projects that involve a smaller number of devices, and projects that need to handle fewer messages.

We chose Azure Service Bus queues for sending data from an Arduino to an Azure VM as we wanted to check the temperature and light every certain amount of time and send this information to the database. It could be sent to the Azure Service Bus queue at regular intervals, which can then be picked up by the Python script running on the Azure VM. The Python script can then connect to the database and check the comfort temperature of each plant. Additionally, it also allows for the possibility of having a scalable solution, as you can increase the number of messages sent and received based on your needs.

The connection with the database was written in python, we used two main files `arduinodb_pc.py` (the one in our local machine and reading the Arduino) and `arduinodb_cloud.py` (the one in the cloud). We will be defining the most important lines of code in this section.

In order to read the "currentTinF" variable from a VM in Azure, we will need to set up communication between the two systems. We set up a Service Bus queue on the VM in Azure, and then have the Python script on the local side send the "currentTinF" variable to that queue. The script running on the VM in Azure can then read the value from the queue and read the values in the mySQL database.

Here is how we sent the "currentTinF" variable to a Service Bus queue using the azure-service bus library in Python, using the code in listing 4.9.

Code 4.9: Code to connect to the VM

```
1  while True:
2      # Connection string and namespace
3      servicebus_client =
           ServiceBusClient.from_connection_string("Endpoint=sb://servicebus-tfg2022.servicebus.windows.
4  ␣␣␣␣␣␣␣␣␣␣␣␣␣␣net/;SharedAccessKeyName=RootManageSharedAccessKey;SharedAccessKey=pwd")
5
6      global currentTinF, currentTinC,text_currentT
7      currentTinF, currentTinC, text_currentT = arduino_read_T(arduino, 1)
8
9      sender = servicebus_client.get_queue_sender(queue_name="myqueue")
10     # Create a Service Bus message and send it to the queue
11     message = ServiceBusMessage(str(currentTinF))
12     sender.send_messages(message)
13     print(f"{datetime.datetime.now()}␣->␣Local␣sent:␣{currentTinF}")
14
15     time.sleep(10)
```

On the Azure VM side, we add the same init of the service bus, but in this case, we set the script to read the variable sent from our machine. The piece of code can be seen in listing 4.10. A little disclaimer about the code is that in this version, the user can have plants in the same area that are hot and cold at the same time. This will be arranged in the future, where the user would have the plants separated depending on the temperature and zones.

Code 4.10: Code to connect to the local machine

```
1  while True:
2      # Connection string and namespace
3      servicebus_client =
           ServiceBusClient.from_connection_string("Endpoint=sb://servicebus-tfg2022.servicebus.windows.
4  ␣␣␣␣␣␣␣␣␣␣␣␣␣␣net/;SharedAccessKeyName=RootManageSharedAccessKey;SharedAccessKey=pwd")
5
6      receiver = servicebus_client.get_queue_receiver(queue_name="myqueue")
7      received_msgs = receiver.receive_messages(max_wait_time=15, max_message_count=20)
8      for msg in received_msgs:
9          print(f"{datetime.datetime.now()}␣->␣Cloud␣received:␣" + str(msg))
10         # Create a cursor object to execute SQL commands:
11         cursor.execute("SELECT␣*␣FROM␣my_plants")
12         rows = cursor.fetchall()
13         list_cold = []
14         list_hot = []
15         for row in rows:
16             if (float(str(msg)) < row[24]): list_cold.append(row)
17             elif (float(str(msg)) > row[25]): list_hot.append(row)
18             row = cursor.fetchone() # Move to the next row
19         if list_cold and list_hot: message = "hot_cold"
20         elif list_cold: message = "cold"
21         elif list_hot: message = "hot"
22         else: message = "ok"
23         # complete the message so that the message is removed from the queue
24         receiver.complete_message(msg)
25
26         sender = servicebus_client.get_queue_sender(queue_name="myqueue2")
27         # Create a Service Bus message and send it to the queue
28         back = ServiceBusMessage(message)
29         sender.send_messages(back)
30         print(f"{datetime.datetime.now()}␣->␣Cloud␣sent:␣{message}")
31
32     servicebus_client.close()
33     time.sleep(10)
```

The last step is to get the message ”msg” created by the VM listing 4.11, and depending on that, send (or not) a message to the Arduino, for it to turn on/off devices.

Code 4.11: Code to manage the response of the VM

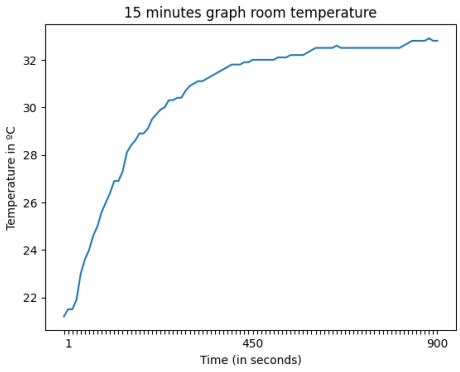
```
11
12      receiver = servicebus_client.get_queue_receiver(queue_name="myqueue2")
13      received_msgs = receiver.receive_messages(max_wait_time=15, max_message_count=20)
14      for msg in received_msgs:
15          print(f"{datetime.datetime.now()} -> Local received: " + str(msg))
16          if str(msg) == "cold": arduino.write(bytes("cold", 'utf-8'))
17          elif str(msg) == "hot": arduino.write(bytes("hot", 'utf-8'))
18          # complete the message so that the message is removed from the queue
19          receiver.complete_message(msg)
20      servicebus_client.close()
```

5

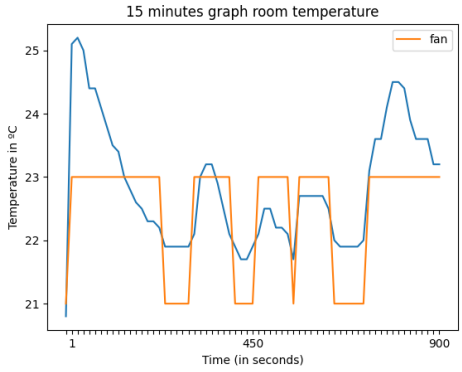
Results and Discussion

5.1 Results

The results of this project have shown that it is possible to create a climate control system for plants in the cloud using IoT and cloud computing technologies. The system was able to accurately monitor and control the growing conditions of plants in a greenhouse and was able to provide detailed data and insights into the growing conditions. The system was also designed to be easily expandable, allowing for the addition of new sensors and devices as needed. To further reinforce these findings we have included this figure 5.1 comparing the temperature reading before and after implementing the system.



(a) Graph of the temperature room with a heater on and without the fan implemented.



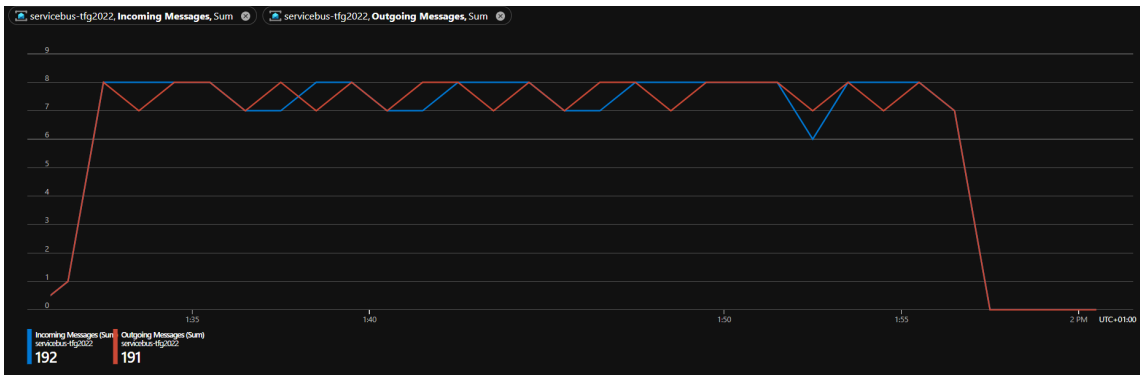
(b) Graph of the temperature room with a heater on and with the fan implemented.

Figure 5.1: Graph with and without fan

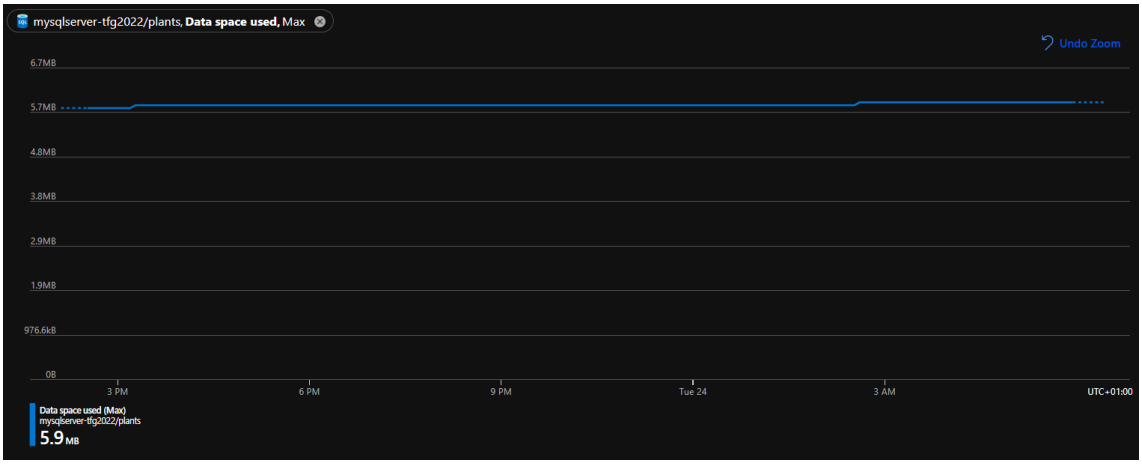
As we can see on figure 5.1(a), the temperature without having a fan in the room/greenhouse, we get temperatures of a minimum of 21.2º and a maximum of 32.9º Celsius. The temperature increase is steady but not linear, with slight fluctuations until it reaches a maximum. A different thing happens on figure 5.1(b), the temperature increases over the next several minutes, the fan turns on and the temperature reaches 25.2 ºC and remains on until the temperature decreases to 21.9℃ and then the

fan turns off. The temperature fluctuates between 21.9℃ and 24.5℃ for the rest of the data with the fan turning on and off accordingly. The fan is on when the temperature is 22℃ or higher. The orange line at 23º indicates when the fan was turned on, if it is at 21º, it indicates that the fan was off.

The complete solution for the climate control system for plants was created by integrating various system components. One of the key components was the use of the Azure service bus in figure 5.2(a) which acted as a container for all the queues we had, that were receiving and sending information from the sensors in the greenhouse. The data collected from the sensors and devices were then stored in the Azure MySQL server figure 5.2(b).



(a) Graph of the messages sent and received by both of the queues.



(b) Graph of the database storage in Azure.

Figure 5.2: Graphs of the messages Azure was receiving and sending.

On figure 5.3 we can see the plots Azure monitoring provided us, where we can see the number of messages compared with time. First, in the local file, figure 5.3(a) and figure 5.3(d), the number of incoming messages and outgoing messages is the same for the 30 minutes we left it running, indicating that the messages sent by the local PC are being received correctly by the cloud and the same number is being sent back. For myqueue2, figure 5.3(b) and figure 5.3(c), the incoming messages are lower than the outgoing messages, this means that the cloud is not receiving all the messages sent by the

cloud. We assume this is due to a delay in updating the metrics in Azure, as in table 5.1 we can see how the cloud and local are sending and receiving messages correctly.

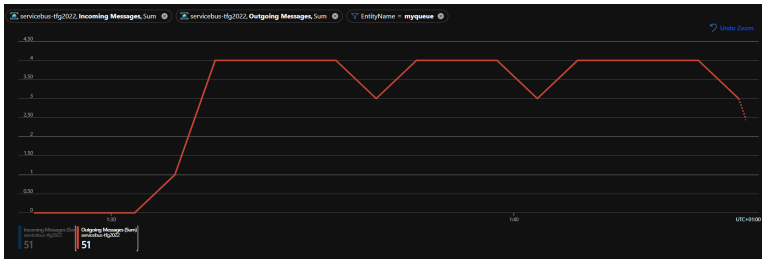
Time	Print
13:30:31.539094	Local sent: 65.3
13:30:43.850577	Local received: cold
13:30:47.644026	Local sent: 63.32
13:30:59.900718	Local received: cold
13:31:02.799246	Local sent: 63.86
13:31:14.981682	Local received: cold
13:31:17.943842	Local sent: 63.86
13:31:30.167591	Local received: cold
13:31:33.249010	Local sent: 63.86
13:31:45.631100	Local received: cold
...	...

(a) Local python output

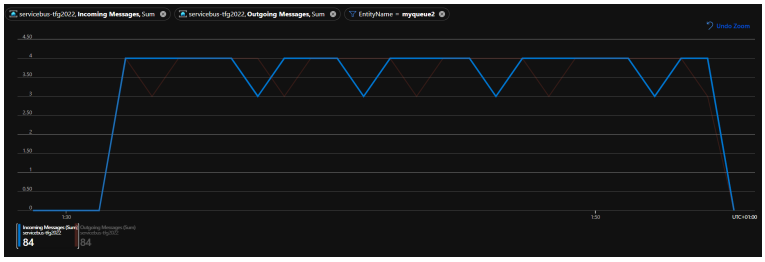
Time	Print
13:30:31.825181	Cloud received: 65.3
13:30:33.803240	Cloud sent: cold
13:30:47.902480	Cloud received: 63.32
13:30:49.853312	Cloud sent: cold
13:31:03.320804	Cloud received: 63.86
13:31:05.520761	Cloud sent: cold
13:31:18.727527	Cloud received: 63.86
13:31:20.743666	Cloud sent: cold
13:31:34.160295	Cloud received: 63.86
13:31:36.108277	Cloud sent: cold
...	...

(b) Cloud python output

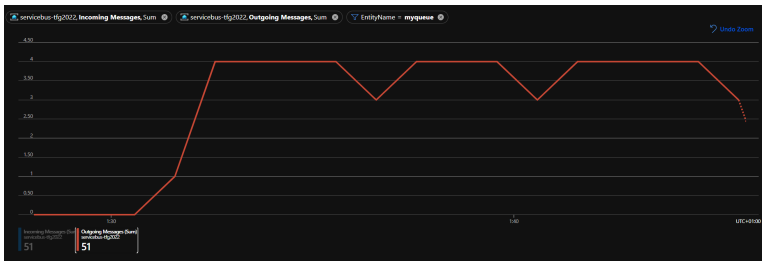
Table 5.1: Execution files output



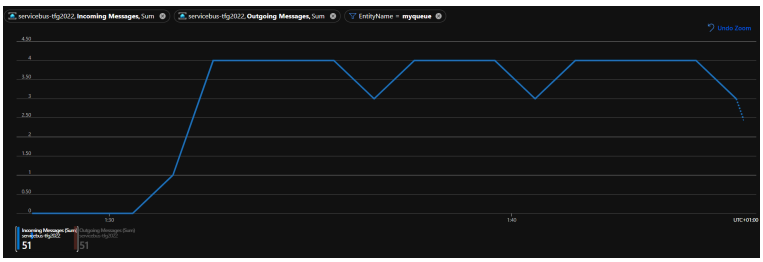
(a) Plot of the messages sent by the local machine.



(b) Plot of the messages received by the cloud.



(c) Plot of the messages sent by the cloud.



(d) Plot of the messages received by the local machine.

Figure 5.3: Graphs of the messages sent/received by the local machine and cloud.

5.1.1 User Interface (UI)

For making it user-friendly, so people not having to interact with the database in SQL code, we built a UI using the library Tkinter. The following images show the different actions a user could make inside the said interface. For the user having zero interaction with the base, we run the command

```
python -m PyInstaller -onefile -noconsole gui.py
```

This command is using the PyInstaller package to package ”gui.py”, into a single executable file. The flags ”–onefile” and ”–noconsole” are used to create a single executable file without a console window. This allows the user to run the program as if it were a regular application (gui.exe), rather than having to run it from the command line.

When the user first clicks on the app, this window (figure 5.4) shows up, for them to choose the temperature units they prefer to see in their window.

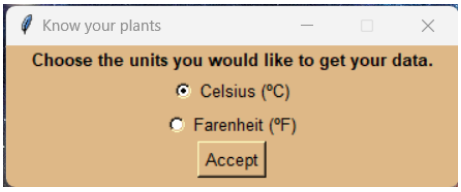


Figure 5.4: Window for choosing temperature units. After the decision, the user only has to press the button ”Accept” and the session will be showing, in this case, in ºC.

As shown in figure 5.5(a), figure 5.7(a) and figure 5.8(a), there is a standard part for the GUI, a part that does not change, the top and left frames. The first one shows the current date and hour, as well as the current temperature. The day the screenshots were taken, January 11th 2023 at 13:14h, it was 18.0ºC. The top frame is created in listing 5.1. We observe that the function `update_clock()` is the one that is in charge of updating the clock.

The left frame listing 5.2 is used for the user to choose which functionality the GUI wants to use. Each time the user presses a button, the right frame is deleted and creates a new one, is for not having an overlap. In the code, when we get to lines 33 to 34, a function with the same name as the function inside `.config(...)` is called. There is where we manage all the functionalities.

Code 5.1: Code for creating the date hour and temperature frame

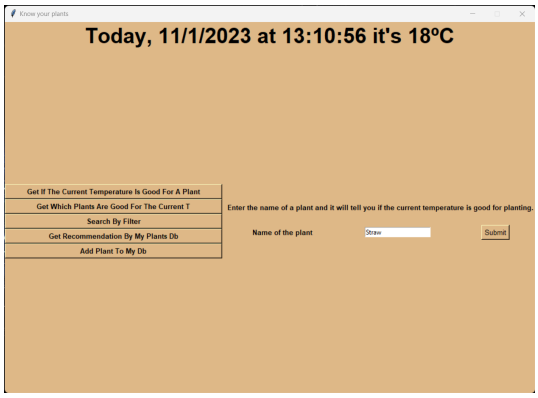
```
app = Tk() # Creating the main window
...
def update_clock(): # Updates the label where the time and temperature is shown
    currentTinF, currentTinC, text_currentT = arduino_read_T(arduino, unit)
    label_title.config(text = text_currentT)
    app.after(1000,update_clock) # Here is the change

top_frame = Frame(app, relief=FLAT) # Create top frame
top_frame.pack(side=TOP)
label_title = Label(top_frame, text="␣", font="Helvetica␣30␣bold", bg="burlywood", width=40) #
    Create label where time and temperature will be shown
label_title.grid(row=0, column=0)
update_clock()
```

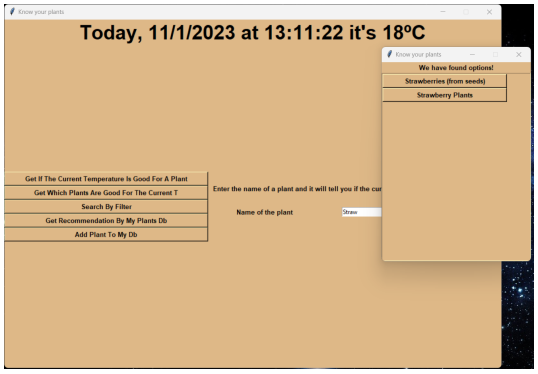
Code 5.2: Code for creating the buttons frame

```
left_frame = Frame(app, relief=FLAT) # Create left frame
left_frame.pack(side=LEFT)
buttons = ["Get␣if␣the␣current␣temperature␣is␣good␣for␣a␣plant",..., "Add␣plant␣to␣my␣DB"]
created_buttons = []
col = 0
for boton in buttons: # Add buttons to left frame
    boton = Button(left_frame, text=boton.title(), bg="burlywood", font="Helvetica␣10␣bold",
        width=50)
    created_buttons.append(boton)
    boton.grid(row=col, column=0)
    col+=1
created_buttons[0].config(command=frame_get_T_by_plant) # Add what to do when pushing the button
...
created_buttons[4].config(command=frame_add_plant_db) # Add what to do when pushing the button
```

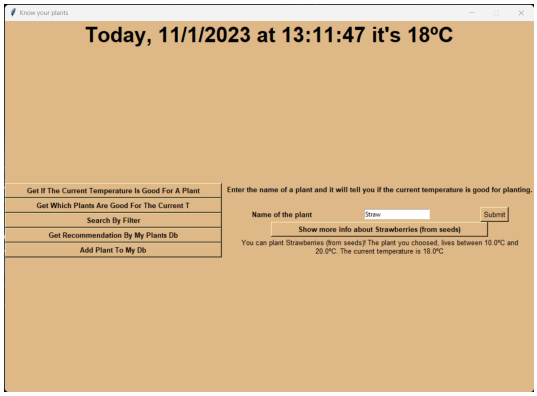
The following screenshots in figure 5.5 show an example of how a user can enter the name of a plant that is stored in the database, and, depending on the temperature and month, check if it is a good season for planting.



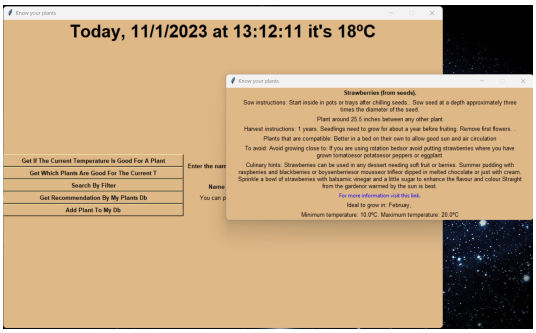
(a) The first step for checking if the temperature and season are good for planting a certain plant, consists of going to "Get if the current temperature is good for a plant", entering the full or partial name of a plant and pressing "Submit". In this example we will be looking for plants that contain "straw" in their name.



(b) After pressing submit, if it exists more than one coincidence in the main database, a list is shown for the user to choose which one is the one that wants.



(c) It is chosen the "Strawberries (from seeds)" so it is shown in the UI that they would survive with this temperature. There is also an option for pressing the button "show more info about ..." and get more information about how to sow them.



(d) Pressing the button for getting more information, we will get a window like the one above, where all the information necessary is shown ("Strawberries (from seeds)"). We can see that the temperature is between 10ºC and 20ºC, which is ideal for growing in February. As we are in January, the season coincides.

Figure 5.5: Steps for checking if the temperature and season are good for planting a certain plant.

If the case of not knowing what to plant but wanting to plant something, there is also a button ("Get which plants are good for the current T") for getting which plants could you sow depending on the temperature and season. It is shown in figure 5.6.

We also implemented a way of searching for a plant more specific. There are some ways, Entering the name and not checking the boxes will return only the coincidences of the plant name. If you only checkboxes, without putting a name in the box, the app will return the plants that coincide with the box checked. An example is shown in figure 5.7.

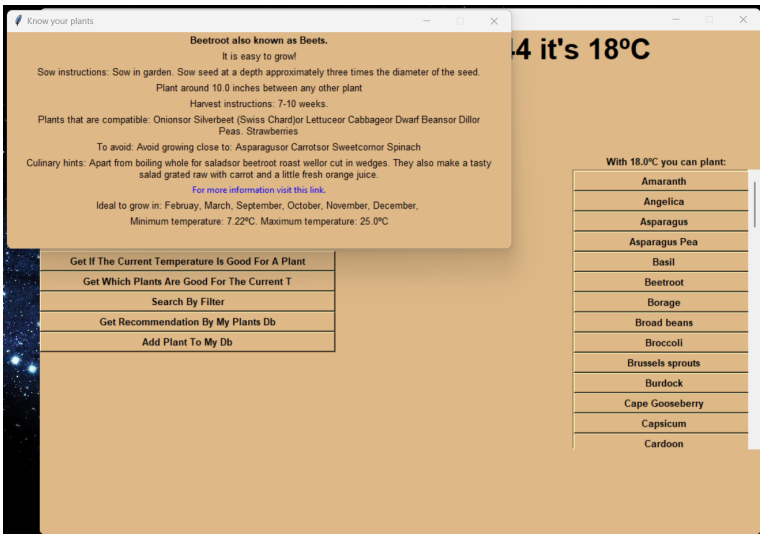
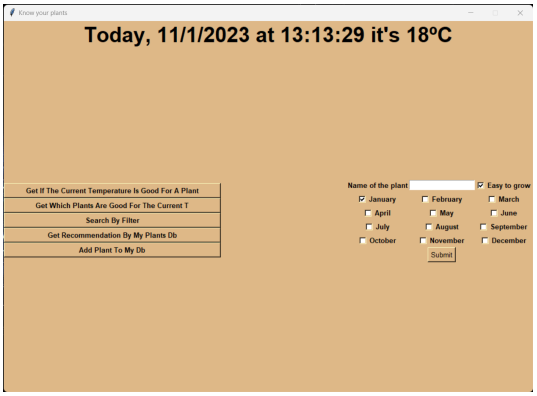
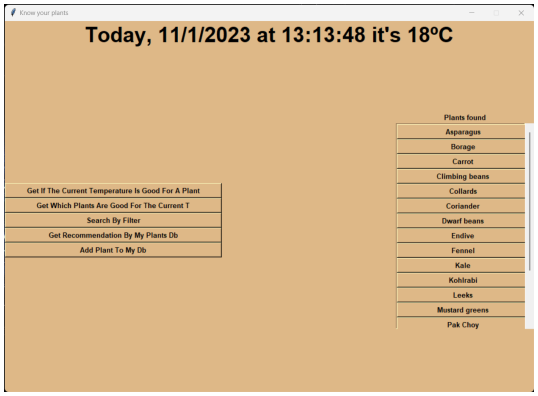


Figure 5.6: In this image is shown how the recommended plants are in a list taking into account the 18ºC temperature and season. When pressing any name of a plant, an information window will be opened.



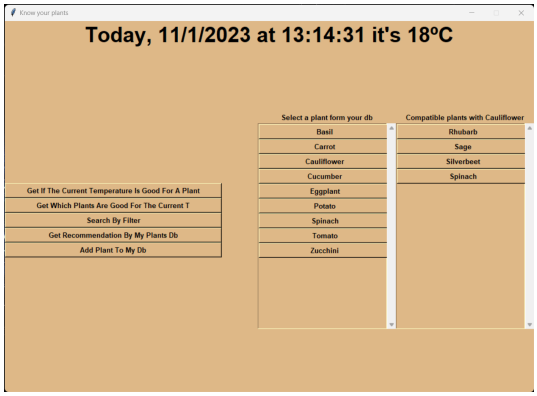
(a) In this figure we show how the interface looks after pressing ”Search by filter”. In this case, we are going to search for *ANY* plants in the main database that are *easy to grow* and are needed to be *planted in January*. The user is also able to check more than one month-checkbox at the same time and to introduce a full or partial name of a plant, as in figure 5.5(a).



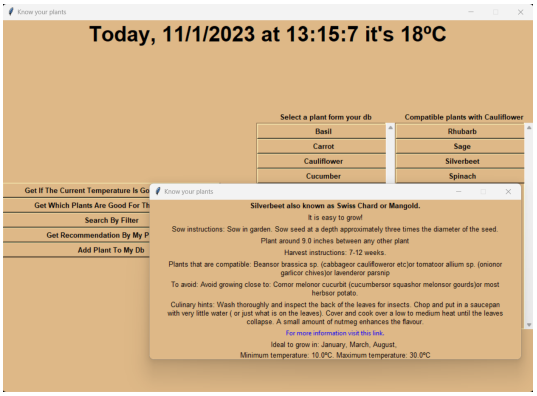
(b) Once the user presses the button ”submit” a panel with all the plants that meet this requirement, will be shown in ”Plants found”. When choosing the plant, a window as in figure 5.5(d) and figure 5.6 with information about the plant will be open.

Figure 5.7: Steps to follow if a user wants to do a more specific search.

As some plants are better to plant together as in that way they grow better thanks to the symbiosis (animals, plants and microorganisms relationship) [50], there is also an option that recommends the user what to plant depending on which plants you have in your greenhouse.



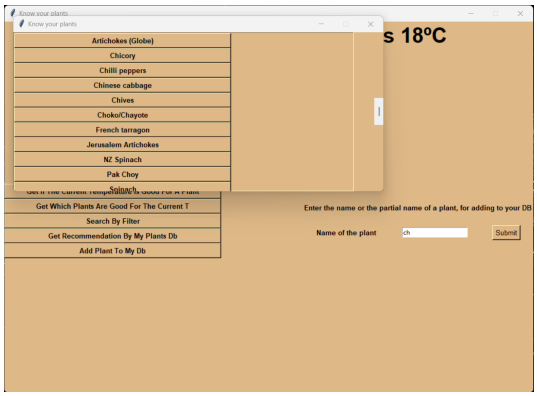
(a) The first column "Select a plant from your DB" is shown, which shows the user the plants that are stored in their database. They press any of the plants, in this case "Cauliflower". The second column ("Compatible plants with ...") is open, with all the plants that are compatible to grow next to the cauliflower.



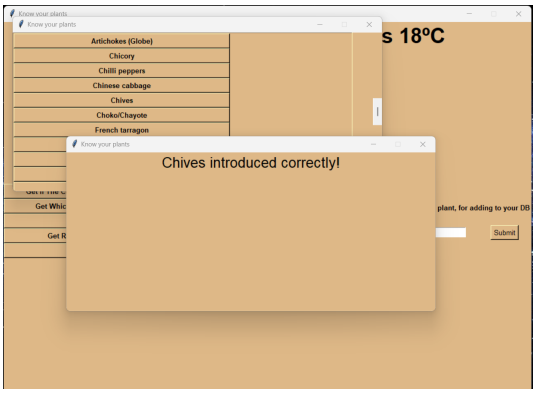
(b) If the user presses any button, a window as in figure 5.5(d) appears.

Figure 5.8: Get recommendations about what can you plant depending on what you have in your database plant

Finally, the option of adding a plant to the user database is implemented. It will need to be upgraded on a future version if we want to add different fields with different plants shown in figure 5.9. Each field has plants with similar temperatures, so a fan, the sensors and a different database would be specific to each field. The idea would be for the app to tell the user which field is better to plant the new plant, and then add it to the corresponding database.



(a) The user will find a space to enter a name of a plant. They will press the button "Submit" and if there is more than one coincidence, a list with all the names will open.



(b) After pressing a name of a plant of the list in the window in figure 5.9(a), an informative window will be open, if there was an error, or if the introduced plant was already in the database.

Figure 5.9: In this figure it is shown how to add a new plant to the user database

5.2 Discussion

The system was built using an Arduino board as the main controller, with temperature and light sensors. These sensors collected data and sent it to a cloud-based database, where it was analyzed and used to make decisions about adjusting the growing conditions. The system was also designed to be cost-effective and easy to use, using low-cost and widely available hardware components and open-source software.

The results of the project show that the system was able to accurately monitor and control the temperature in the growing area. The system was able to make adjustments to the environment based on the data collected, turning on fans to cool the plants down. The system also provided detailed data and insights into the growing conditions, which can be used to optimize the growth of plants and improve yields.

One of the key challenges of this project was to create a system that is both cost-effective and easy to use. To achieve this, we used low-cost and widely available hardware components, such as the Arduino board, and open-source software. We also used a cloud-based database, which allowed us to easily store and analyze large amounts of data.

6

Conclusions and Future work

6.1 Conclusions

The proposed climate control system for plants in the cloud is a practical and effective solution for implementing IoT and cloud computing in our greenhouse. In this project, we successfully achieved all of our goals in designing and implementing a personal climate control system for plants in the cloud. The system consisted of temperature and light sensors placed in a greenhouse, an Arduino board for controlling the sensors and sending data to the cloud, and a cloud-based database for storing and analyzing the data.

The Arduino board served as the main controller of the system, collecting data from the temperature and light sensors and sending it to the cloud. The cloud-based database was used to store and analyze the collected data, and the system was designed to be easily expandable to allow for the addition of new sensors and devices as needed.

We were able to create a system that was both cost-effective and easy to use, using low-cost and widely available hardware components and open-source software. The use of a cloud-based database allowed us to easily store and analyze the data we had anywhere, and the system provided detailed data and insights into the growing conditions, which could be used to optimize the growth of plants and improve yields.

In terms of technical implementation, we utilized IoT and cloud computing technologies to accurately monitor and control the growing conditions of plants. The system was designed with a feedback mechanism for controlling environmental conditions such as light and temperature based on the data collected from the sensors, and it was able to provide real-time data and insights into the growing conditions. We had also sensors that were there to warn the user that, for example, the water tank was half empty.

Finally, we conducted a thorough analysis of the potential benefits and challenges of using cloud computing and IoT technology for personal gardening. Our examination of the potential cost savings, increased efficiency, and improved yields that can be achieved through automation confirmed the

viability and usefulness of the system we created.

In conclusion, we successfully achieved all of our goals in this project and were able to create a cost-effective, easy-to-use, and highly functional personal climate control system for plants in the cloud.

6.2 Future work

In our current system, the Arduino is connected to the cloud using a bridge (our personal computer). In the future, it is possible to use Azure IoT Hub as a bridge between the Arduino boards and the cloud. This will allow for a more efficient and reliable connection, as well as the ability to use a Raspberry Pi or other WiFi microprocessor to connect to the internet as we can see in figure A.1.

Nowadays we only have one temperature sensor as well as a light and water level sensor, but we plan to add more sensors in different locations in order to have a more comprehensive view of the growing conditions like in figure A.2.

Our goal with this is to have in each field, plants that are compatible and have similar maximum and minimum temperatures. One of the problems resulting from this solution could be that having so many sensors, the service bus or the IoT hub could collapse. For this problem, we have thought that the best solution would be to create a local network and send the messages through it. In any case, we don't think that this is a problem that will develop in the near future, as it is a private greenhouse, for our own use, and will not have such a large expansion.

Additionally, further work could be done to improve the system's performance and address any security concerns related to cloud computing and IoT technology. It could also be interesting to add more sensors to the system to measure factors such as soil moisture, and air quality, or incorporate machine learning algorithms to make real-time adjustments to the climate control settings based on the collected data.

Bibliography

[1] L. Qian, Z. Luo, Y. Du, and L. Guo, "Cloud computing: An overview," in *IEEE international conference on cloud computing*, pp. 626–631, Springer, 2009.

[2] W. Chai and S. J. Bigelow, "Definition cloud computing." https://www.techtarget.com/searchcloudcomputing/definition/cloud-computing, 2022.

[3] A. W. Services, "What is cloud computing?." https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-a-private-cloud, 2022.

[4] Azure, "What is iaas?." https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-iaas/, 2022.

[5] Azure, "What is paas?." https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-paas/, 2022.

[6] Azure, "What is saas?." https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-saas/, 2022.

[7] Azure, "Saas, paas, iaas," 2022. [Online; accessed December 09, 2022].

[8] Oracle, "What is iot?." https://www.oracle.com/in/internet-of-things/what-is-iot/, 2023.

[9] S. Li, L. D. Xu, and S. Zhao, "The internet of things: a survey," *Information systems frontiers*, vol. 17, no. 2, pp. 243–259, 2015.

[10] M. M. Sadeeq, N. M. Abdulkareem, S. R. Zeebaree, D. M. Ahmed, A. S. Sami, and R. R. Zebari, "Iot and cloud computing issues, challenges and opportunities: A review," *Qubahan Academic Journal*, vol. 1, no. 2, pp. 1–7, 2021.

[11] A. Botta, W. De Donato, V. Persico, and A. Pescapé, "Integration of cloud computing and internet of things: a survey," *Future generation computer systems*, vol. 56, pp. 684–700, 2016.

[12] M. Aazam, I. Khan, A. A. Alsaffar, and E.-N. Huh, "Cloud of things: Integrating internet of things and cloud computing and the issues involved," in *Proceedings of 2014 11th International Bhurban Conference on Applied Sciences & Technology (IBCAST) Islamabad, Pakistan, 14th-18th January, 2014*, pp. 414–419, IEEE, 2014.

[13] Y. Liu, K. A. Hassan, M. Karlsson, O. Weister, and S. Gong, "Active plant wall for green indoor climate based on cloud and internet of things," *IEEE Access*, vol. 6, pp. 33631–33644, 2018.

[14] Camponectado, "Camponectado, ¿qué es?." http://www.camponectado.com/, 2022.

[15] Postscapes, "Smart greenhouse climate control systems." https://www.postscapes.com/greenhouse-climate-and-control-systems/, 2022.

[16] Gardenate, "View your climate zone details." https://gardenate.com/zones/#zone-US, 2012.

[17] S. Brener, "What to plant and when to plant it." https://data.world/sharon/what-to-plant-and-when-to-plant-it, 2017.

[18] Gardenate, "Planting in usa - zone 8a regions." https://www.gardenate.com/zones/USA%2B-%2BZone%2B8a, 2012.

[19] Gardenate, "Planting in usa - zone 10a regions." https://www.gardenate.com/zones/USA%2B-%2BZone%2B10a, 2012.

[20] W. Abroad, "Spain – geography and climate." https://www.workingabroad.com/travel/spain-geography-and-climate/, 2023.

[21] A. Khan, "10 best microcontroller boards for engineers and geeks [updated 2021]." "https://www.engineeringpassion.com/10-best-microcontroller-boards-for-engineers-and-geeks/", 2021.

[22] Y. Fernández, "Arduino y raspberry pi: qué son y cuáles son sus diferencias." https://www.xataka.com/basics/arduino-raspberry-pi-que-cuales-sus-diferencias, 2018.

[23] T. Point, "Microcontrollers - overview." https://www.tutorialspoint.com/microprocessor/microcontrollers_overview.htm.

[24] G. Learning, "Arduino vs raspberry pi: What's the difference?." https://www.mygreatlearning.com/blog/arduino-vs-raspberry-pi/, 2022.

[25] Y. A. Badamasi, "The working principle of an arduino," in *2014 11th international conference on electronics, computer and computation (ICECCO)*, pp. 1–4, IEEE, 2014.

[26] S. Writer, "Microcontroller vs microprocessor - what are the differences?." https://www.totalphase.com/blog/2019/12/microcontroller-vs-microprocessor-what-are-thedifferences, 2019.

[27] Elegoo, "About us." https://www.elegoo.com/en-es/pages/about, 2022.

[28] Chris, "Arduino vs. elegoo: How to choose what's best for you." https://chipwired.com/arduino-vs-elegoo/, 2022.

[29] ELEGOO, *ELEGOO UNO R3 datasheet*.

[30] Azure, "What is a public cloud?." https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-a-public-cloud/, 2022.

[31] Azure, "What is a private cloud?." https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-a-private-cloud/, 2022.

[32] ProjectPro, "Aws vs azure-who is the big winner in the cloud war?." https://www.projectpro.io/article/aws-vs-azure-who-is-the-big-winner-in-the-cloud-war/401, 2022.

[33] E. Jones, "Aws vs azure en 2021 (comparación de los gigantes de la computación en la nube)." https://kinsta.com/es/blog/aws-vs-azure/, 2022.

[34] F. Palumbo, G. Aceto, A. Botta, D. Ciuonzo, V. Persico, and A. Pescapé, "Characterization and analysis of cloud-to-user latency: The case of azure and aws," *Computer Networks*, vol. 184,

p. 107693, 2021.

[35] E. Jones, "Aws vs azure en 2021 (comparación de los gigantes de la computación en la nube)." https://www.znetlive.com/blog/aws-vs-azure-vs-google-cloud-full-database-comparison-2022/, 2022.

[36] W. Al Shehri, "Cloud database database as a service," *International Journal of Database Management Systems*, vol. 5, no. 2, p. 1, 2013.

[37] S. Madakam, V. Lake, V. Lake, V. Lake, *et al.*, "Internet of things (iot): A literature review," *Journal of Computer and Communications*, vol. 3, no. 05, p. 164, 2015.

[38] D. Ucuz *et al.*, "Comparison of the iot platform vendors, microsoft azure, amazon web services, and google cloud, from users' perspectives," in *2020 8th International Symposium on Digital Forensics and Security (ISDFS)*, pp. 1–4, IEEE, 2020.

[39] Aosong Electronics Co.,Ltds, *Digital-output relative humidity temperature sensor/module DHT22 (DHT22 also named as AM2302)*.

[40] Adafruit Industries, *DC Toy / Hobby Motor - 130 Size*.

[41] KEYES, *Water Sensor Module User's Manual*.

[42] TDK, *Piezoelectronic Buzzers(without circuit) PS Series(Pin Terminal/Lead)*.

[43] KLS Electronics, *CdS Photoconductive cells*.

[44] Kingbritght, *FULL COLOR LED LAMP*.

[45] Fairchild, *1N4001 - 1N4007*.

[46] Fairchild, *PN2222r*.

[47] Royal ohm, *Precision Metal Film Fixed Resistor*.

[48] circuito.io, "Circuito design." https://www.circuito.io/app?components=512,9088,9590,10167,11021,11372,11696,752394,956215, 2023.

[49] S. Brener, "Vegetables, herbs, and edible flowers." https://data.world/sharon/vegetables-herbs-and-edible-flowers, 2017.

[50] E. Rosenberg and I. Zilber-Rosenberg, "Symbiosis and development: the hologenome concept," *Birth Defects Research Part C: Embryo Today: Reviews*, vol. 93, no. 1, pp. 56–66, 2011.

APPENDICES

A

Future work diagrams

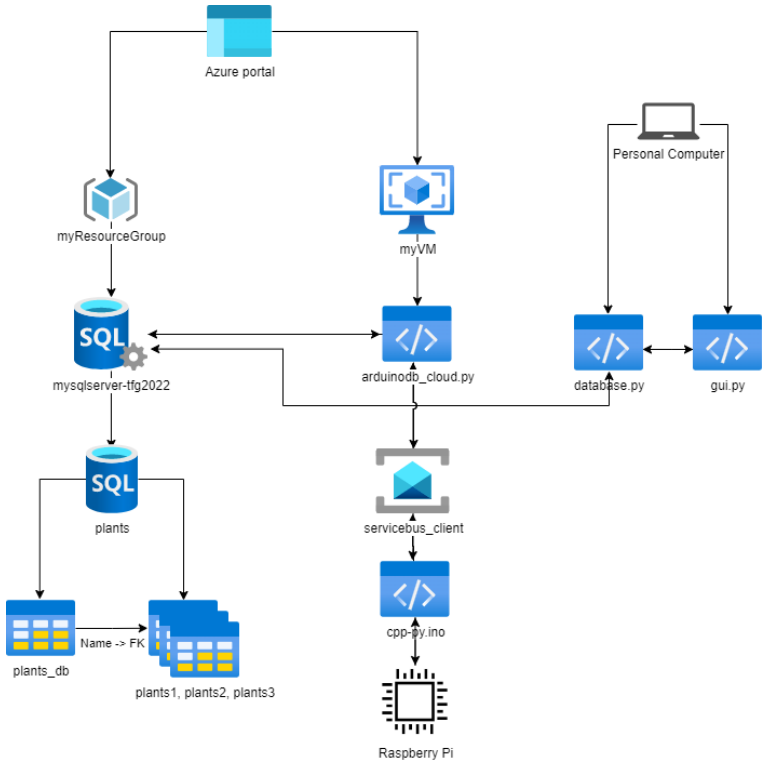


Figure A.1: Diagram of the system as future work

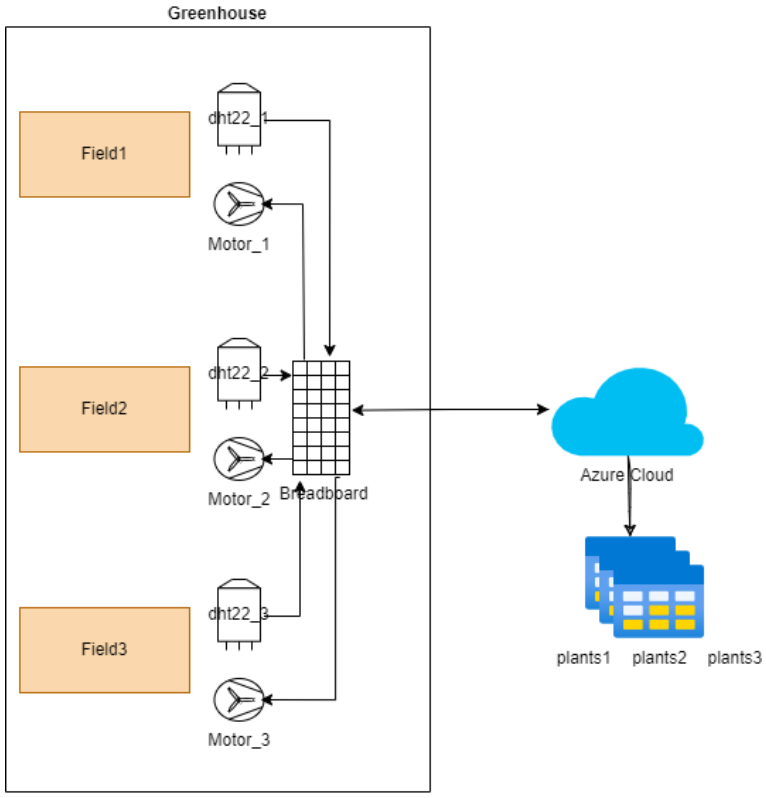


Figure A.2: High-Level diagram of the system as future work


UAM
UNIVERSIDAD AUTONOMA
DE MADRID