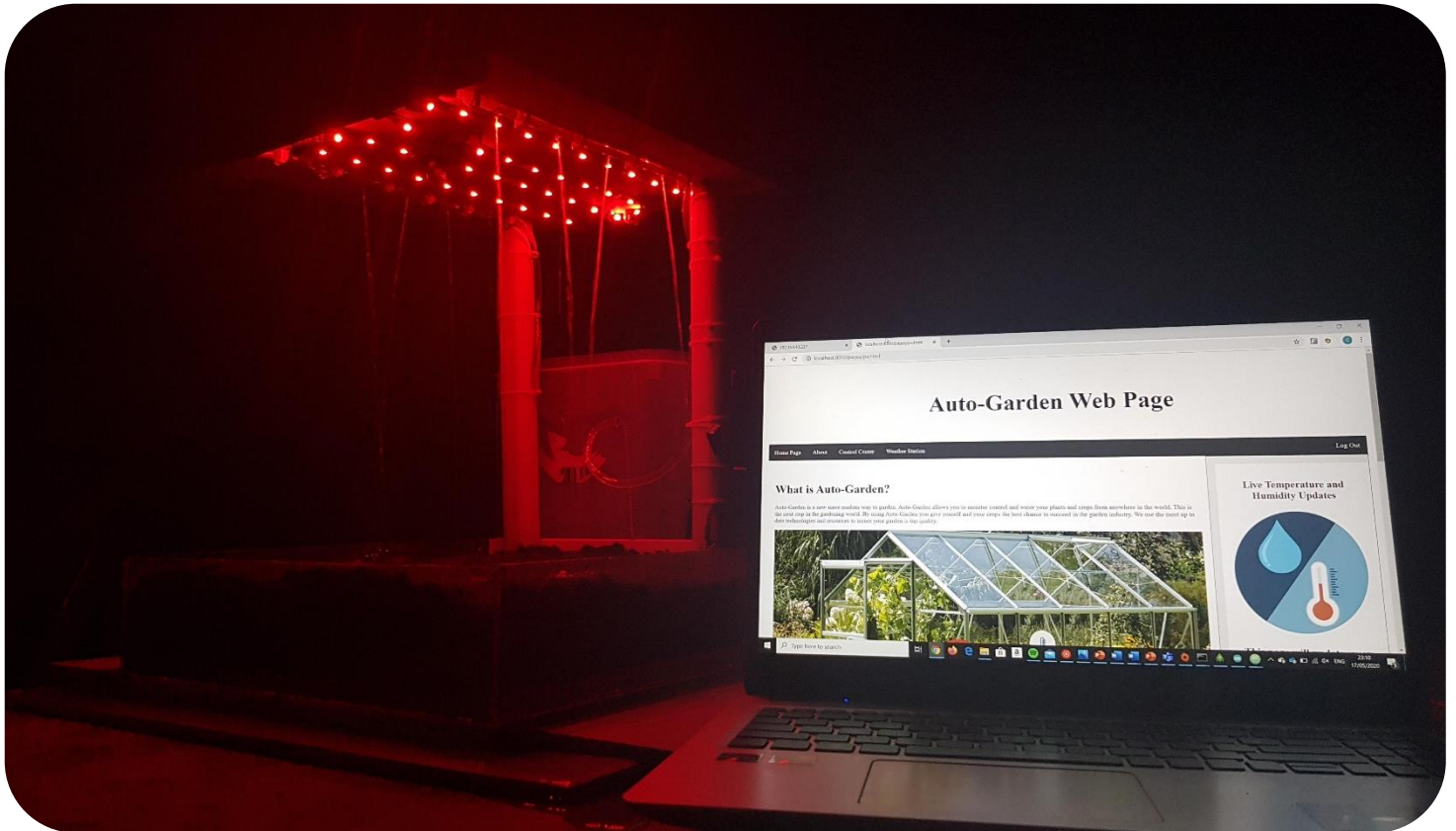


# AUTO - GARDEN

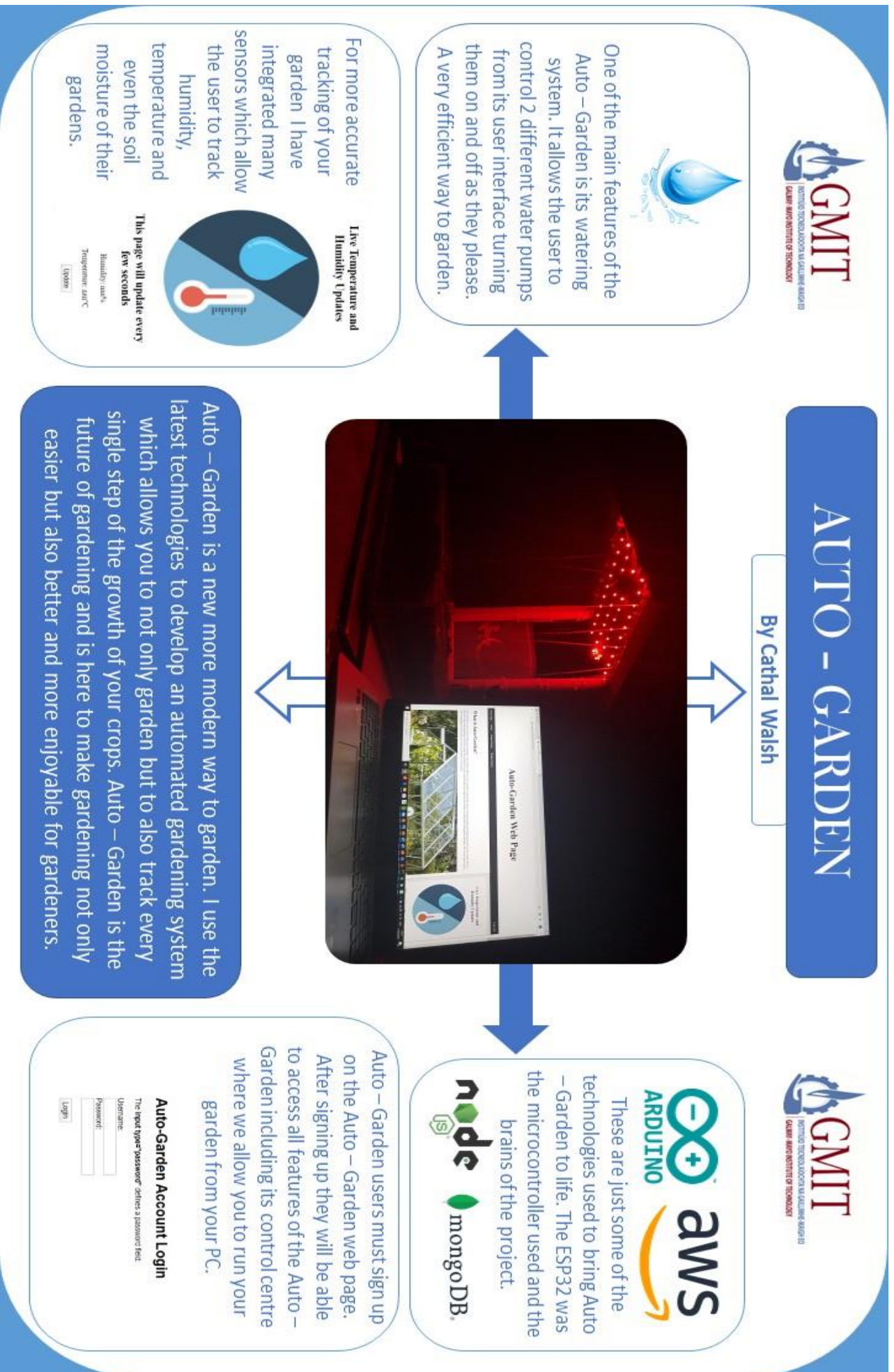


**Cathal Walsh - G00331916**

**Bachelor of Software & Electronic Engineering**  
**Galway-Mayo Institute of Technology**

# Table of Contents

<b>Poster .....</b>	<b>3</b>
<b>Declaration.....</b>	<b>4</b>
<b>Project Summary .....</b>	<b>5</b>
<b>Introduction .....</b>	<b>6</b>
<b>Project Architecture .....</b>	<b>7</b>
ESP32.....	8
NodeJs/MongoDB .....	9
AWS.....	10
<b>Hardware .....</b>	<b>11</b>
DHT11 .....	12
Soil Moisture Sensor .....	13
Blynk .....	13
Relay Module .....	14
Water Pumps.....	15
Lighting System.....	15
<b>Auto-Garden Web Page .....</b>	<b>16</b>
Login .....	16
Home Page .....	17
Control Centre .....	18
Weather Station .....	18
<b>Code .....</b>	<b>19</b>
Arduino Code.....	20
HTML Code.....	22
<b>Auto-Garden Build.....</b>	<b>23</b>
<b>Conclusion .....</b>	<b>24</b>
<b>Project Details .....</b>	<b>25</b>
<b>References.....</b>	<b>26</b>



# Declaration

This project is presented in partial fulfilment of the requirements for the degree of Bachelor of Engineering in Software & Electronic Engineering at Galway-Mayo Institute of Technology.

This project is my own work, except where otherwise accredited. Where the work of others has been used or incorporated during this project, this is acknowledged and referenced.



# Project Summary

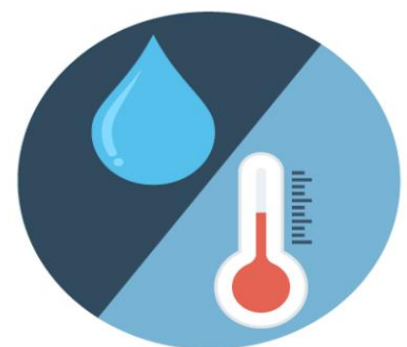
Auto – Garden is a new more modern way to garden. I use the latest technologies to develop an automated gardening system which allows you to not only garden but to also track every single step of the growth of your crops. Auto – Garden is the future of gardening and is here to make gardening not only easier but also better and more enjoyable for gardeners. The main goal of the project was to allow the user full access to their garden from their own smartphone or Pc anywhere at any time. With the Auto – Garden web page and control centre I



believe I achieved this goal. The web page allows the user to sign in and enter the control centre which then allows the user to have complete control over their garden.

In this page they can control everything from watering to the lighting of their garden or green house. The user can also track very useful data such as temperature, humidity and soil moisture from the weather station section of the Auto – Garden web page. To allow the user to this I have implemented very handy small sensors such as the Arduino DHT11 and a soil moisture sensor to keep track of this data. A very helpful addition to the any garden. [\[1\]](#)

## Live Temperature and Humidity Updates



**This page will update every few seconds**

Humidity: 50%

Temperature: 14°C

[Update](#)

# Introduction

Deciding on my final year project was a difficult decision. I ended up going with the Auto – Garden as back in third year I developed a project called Auto – Water. This was an automated watering system which was controlled via



Bluetooth. After completing this project, I had so much more ideas on how to make it even better and develop it more. Therefore, I decided on this project the Auto – Garden. The Auto – Garden is much more than just a watering system, this is a completely automated gardening system. I work in a garden shop myself part time, so gardening had always interested me and probably where the motivation for a project such as this came from in the first place. For this project I wanted to go bigger and better than Auto – Water I knew I wanted a web page from the start to act as a focal point for the whole project.

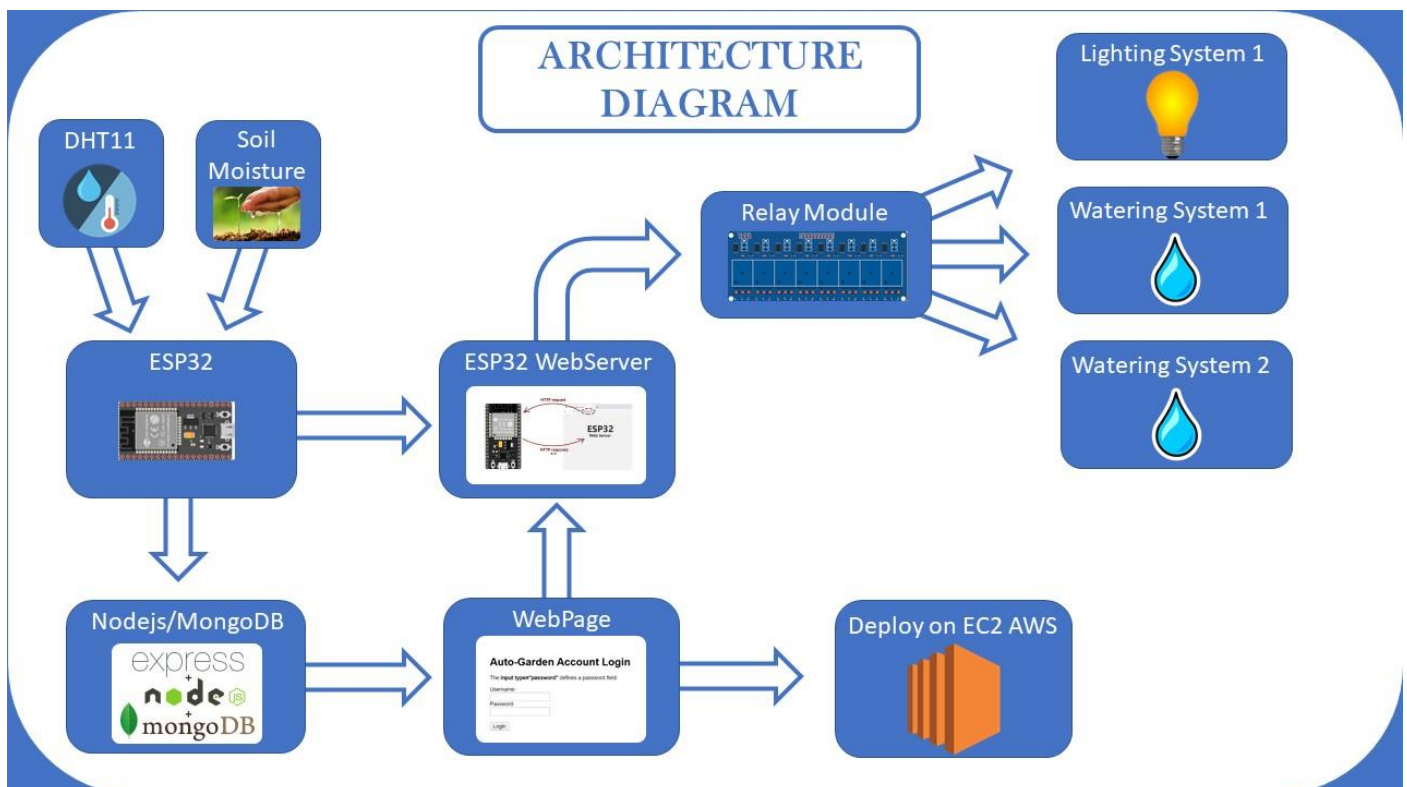
The ESP32 microcontroller came to mind straight away also due to its built in Wi-Fi module and how easy and convenient it is to use and implement into projects. I had previous experience with Arduino and its microcontrollers and found them very useful in projects just like this. So, after some research on it I decided the ESP32 was definitely the way I was going to go with this project.

Throughout this report I will explain how I developed my idea of an automated Gardening system and take you through some of the main technologies used in the creation of Auto – Garden.



# Project Architecture

In this Section I will take you through all the tools used in the development of the Auto – Garden project. I will discuss technologies such as Arduino, AWS, Mongo DB and Node js which all play vital roles in bringing this project to life. On this page you can see the Architecture diagram of the whole project which shows the main technologies used. I will then go into more detail about each one. [8]

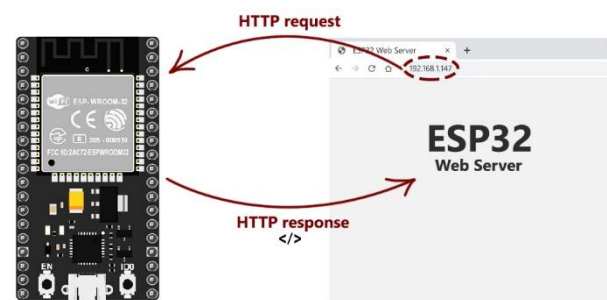


From this Diagram you can see some of the main features and identify some of the main tools used during development. Two sensors the DHT11 and Soil moisture sensor are the data suppliers to the data centre section of the page. Nodejs and Mongo DB render this data to the web page. On this Web Page you can access the Control centre which is to the ESP32 Webserver and allows full control over the relay module which is wired up to the watering and lighting system of the project. The Nodejs application is then deployed on AWS to allow access to everyone. [5][7][8]

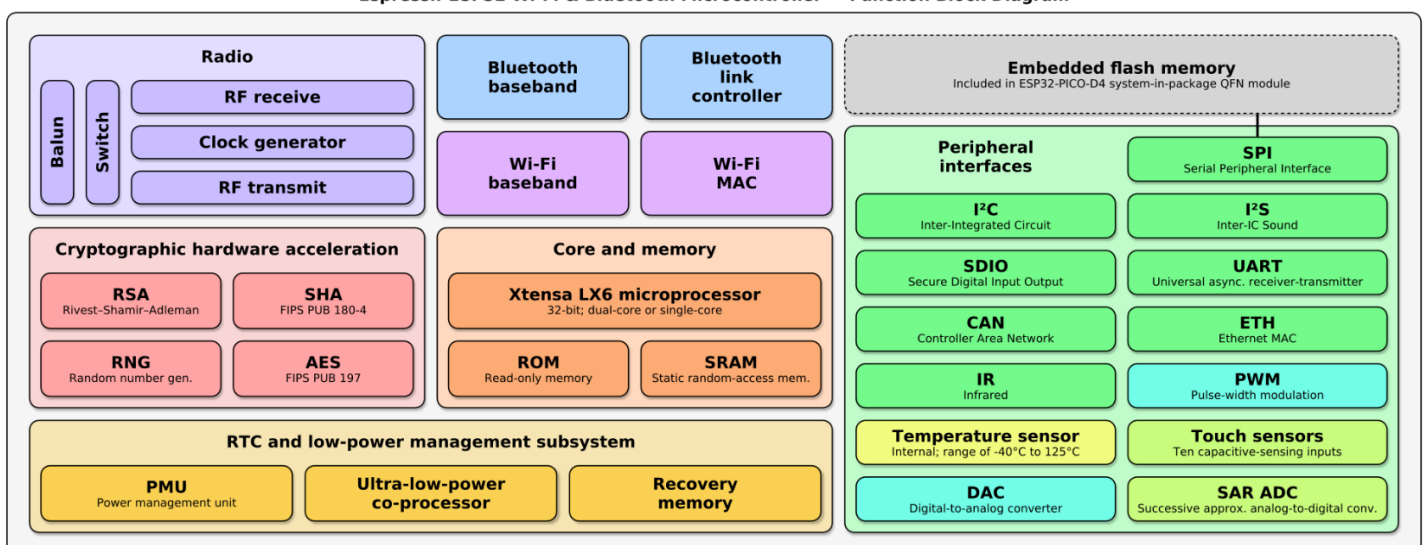


## ESP32

The main brains of the Auto – Garden is the ESP32 microcontroller by Arduino. The ESP32 was a very attractive microcontroller for me and my project as not only does it have integrated Wi-Fi and Bluetooth; it is also low power and very cost friendly. The ESP32 allows me to connect to an existing WiFi network it then gets the IP from the wireless router to which it is connected. With this IP address it sets up a webserver and delivers web pages to the connected devices under existing WiFi network. The control centre of the project is on one of these web pages and allows the user to have full control over the Auto – Garden features once connected to this web server. [2]



Espresif ESP32 Wi-Fi & Bluetooth Microcontroller — Function Block Diagram



As well as running the web server the ESP32 also allows you to control a relay module over WIFI. This is a key feature of the project as it is how the user is able to have full control over the 2 waterpumps and the LED lights integrated into the project through the web. The ESP32 also retrieves the temperature and humidity data from the DHT11 sensor attached to the project. [2]



## NodeJS/MongoDB

As I mentioned earlier one of the starting goals of this project was to develop some sort of web application that controlled all the features of this project. I wanted the user to be able to go online and sign into their own virtual garden in a way. Having full control over everything while sitting on their PC. To build this application I used technologies called NodeJS express and MongoDB. Node js is a useful tool for developing applications such as the Auto – Garden web page. It is an open source development platform for executing JavaScript code server-side. [\[8\]](#)

### Auto-Garden Account Login

The `input type="password"` defines a password field:

Username:

Password:

Login



In conjunction to Node js I use a database called MongoDB. These are often used together to build a lot of modern-day web applications, so I thought they were the perfect fit for the Auto – Garden Web application. In the creation of this web application we also use Express. Express is a lightweight web application framework for Node.js, which provides us with a robust set of features for writing web apps. Express allows us to do such things as route handling, template engine integration and a middleware framework, which allows us to perform additional tasks on request and response objects. Making Developing this application a lot easier!

## Amazon Web Services (AWS)

AWS has a huge part to play in the Auto – Garden project. Amazon Web Services is a secure cloud services platform, offering compute power, database storage, content delivery and other functionality to help businesses scale and grow. In Auto Garden we use AWS IoT to publish our data from our DHT11 sensor to our Auto – Garden Node JS web application. This allows our page to give the user live data updates on what is going on in their garden. In the images below you can see AWS receiving the Temperature and Humidity data from the ESP32. It then publishes this data to Node JS Express and Mongo DB who render it to the web application.

COM5

The screenshot shows the AWS IoT console. On the left, a terminal window displays the following log output:

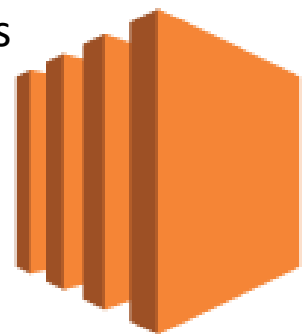
```

11:09:53.565 -> Publishing:-
11:09:53.565 -> Temperature: nan°C Humidity: nan %
11:09:53.600 -> Failed!
11:09:53.634 ->
11:09:54.625 -> Publishing:-
11:09:54.625 -> Temperature: nan°C Humidity: nan %
11:09:54.659 -> Success
11:09:54.693 ->
11:09:55.618 -> Publishing:-
11:09:55.651 -> Temperature: nan°C Humidity: nan %
11:09:55.684 -> Success
11:09:55.684 ->
11:09:56.706 -> Publishing:-
11:09:56.706 -> Temperature: nan°C Humidity: nan %
11:09:56.741 -> Success
11:09:56.741 ->
11:09:57.694 -> Publishing:-
11:09:57.694 -> Temperature: nan°C Humidity: nan %
11:09:57.762 -> Success
11:09:57.762 ->
  
```

On the right, the 'Subscriptions' tab is active, showing a topic '\$aws/things/AutoGarden/shadow/update'. The 'Publish' section shows a message: 'Hello from AWS IoT console'. Below this, a table lists recent publications:

Topic	Message	Timestamp	Actions
\$aws/things/AutoGarden/shadow/update	Temperature: nan°C Humidity: nan %	Mar 31, 2020 11:09:09 AM +0100	Export Hide
\$aws/things/AutoGarden/shadow/update	Temperature: nan°C Humidity: nan %	Mar 31, 2020 11:09:07 AM +0100	Export Hide

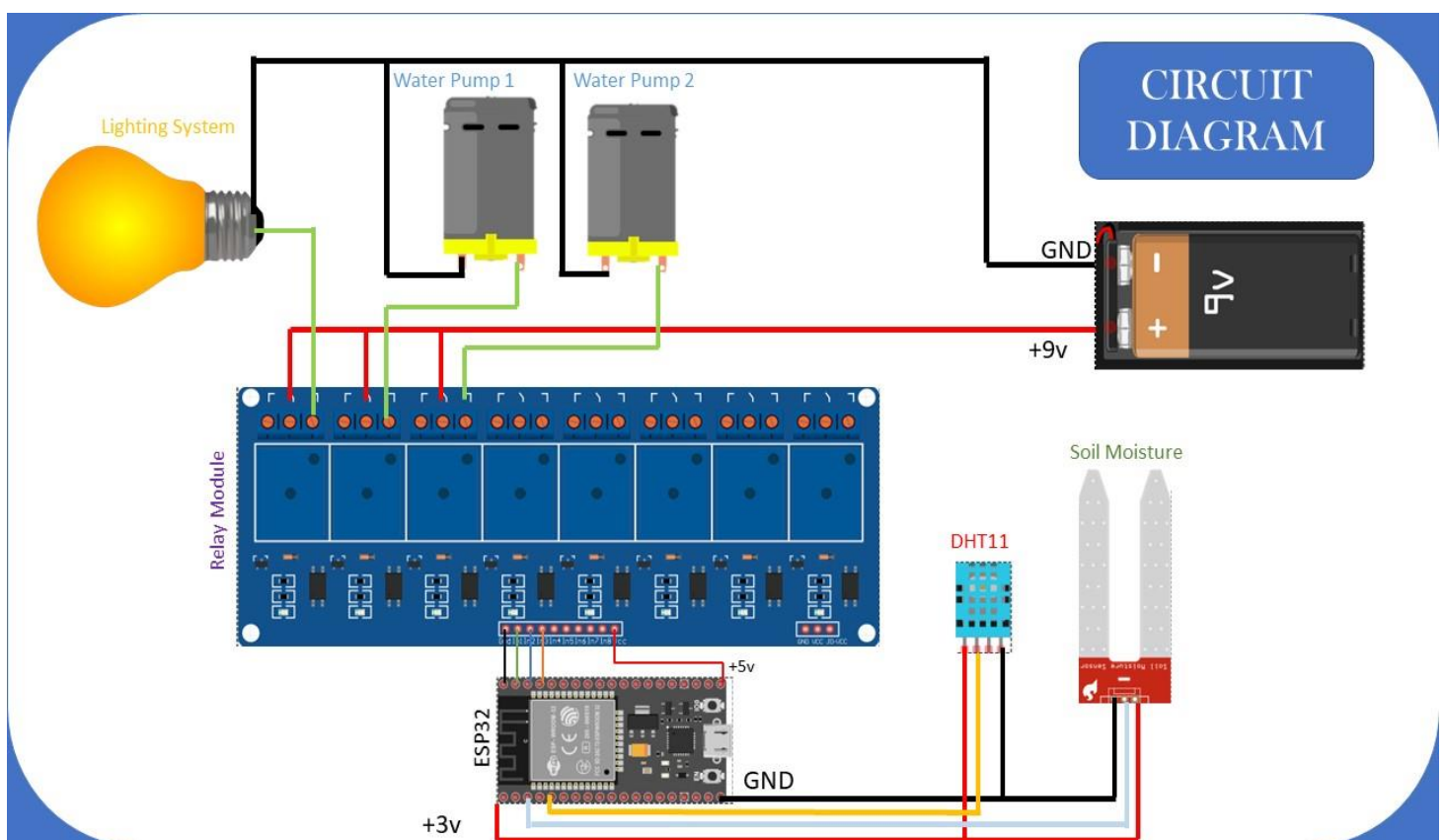
I also use AWS a second time in this project for deployment of the web application. As the NodeJs application is running locally I need to deploy it to grant everyone access to the Application. For this part of the project I use AWS EC2 instance. An EC2 instance is a virtual server in Amazon's Elastic Compute Cloud for running applications on the Amazon Web Services infrastructure. This allows us to run the Auto – Garden web application on AWS allowing everyone access to the web page. The instance can be turned on and off as you wish making the application live or keeping it local. [7]



Amazon  
**EC2**

# Hardware

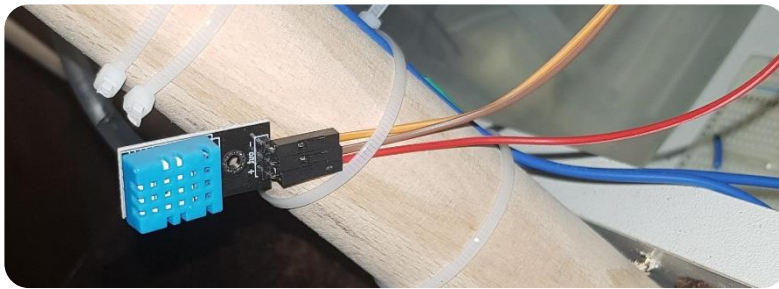
In this section I will run through all the hardware used in the project, what each piece of equipment does and why I chose to use it. Below you can see the circuit diagram of the gardening system itself. It includes the power supply which is a 9-volt battery. I chose the 9 volts as it was the perfect amount to run both water pumps and the lighting system at the same time to good a standard. The 9-volt battery just powers the 3 outputs of the circuit.



The relay module is then powered by the 5 volts from the ESP32 microcontroller. The inputs of the circuit are the sensors. In Auto – Garden I use two different sensors, DHT11 sensor and a Soil moisture sensor which I will talk about more later in this section. These are both powered by the 3-volt output from the ESP32 and provide all important garden data for the user to use in the monitoring of their plants. I power the ESP32 itself using a power bank or PC when applying new features to the system.

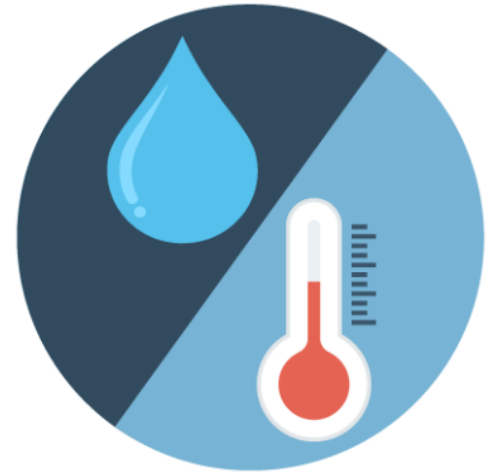
## DHT11

The DHT11 is an Arduino Sensor I have implemented into the data centre of the project to collect the temperature and humidity of the user's garden. I chose the DHT11 as I had used it before and was familiar with applying it to this project. It is also very low cost. How it works is it uses a capacitive humidity sensor and a thermistor to measure the surrounding air and spits out a digital signal onto the data pin which is wired up to the ESP32. [\[1\]](#)



The code to read the temperature and humidity is simple when using the DHT11 with Arduino. I needed to install the DHT11 Arduino library to allow it to work first. I set up two functions one to read the temperature (`readDHTTemperature()`) and the other to read humidity (`readDHTHumidity()`). Getting sensor readings is as simple as using the `readTemperature()` and `readHumidity()` methods on the DHT object. [\[4\]](#)[\[5\]](#)

### Live Temperature and Humidity Updates



**This page will update every few seconds**

Humidity: 50%

Temperature: 14°C

Update

```
String readDHTTemperature() {
  Celsius (the default)
  float t = dht.readTemperature();
  // Read temperature as Fahrenheit (isFahrenheit = true)
  //float t = dht.readTemperature(true);
  // Check if any reads failed and exit early (to try again)
  if (isnan(t)) {
    Serial.println("Failed to read from DHT sensor!");
    return "--";
  }
  else {
    Serial.println(t);
    return String(t);
  }
}
```

```
String readDHTHumidity() {
  float h = dht.readHumidity();
  if (isnan(h)) {
    Serial.println("Failed to read from DHT sensor!");
    return "--";
  }
  else {
    Serial.println(h);
    return String(h);
  }
}
```



## Soil Moisture Sensor

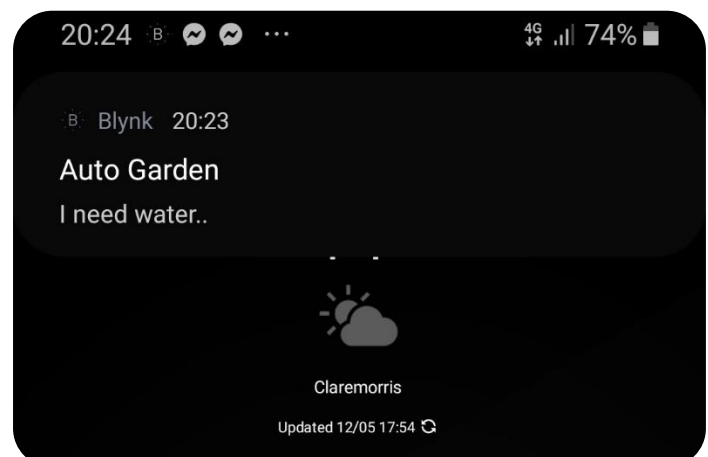
The second of the sensors I have implemented into Auto – Garden is a soil moisture sensor. The Soil Moisture Sensor uses capacitance to measure dielectric permittivity of the surrounding medium. In soil, dielectric permittivity is a function of the water content. The sensor creates a voltage proportional to the dielectric permittivity, and therefore the water content of the soil. Once the soil moisture is low in the Auto – Garden the user will receive a notification from their phone saying the soil needs water. I have linked the soil moisture sensor up to a phone application called Blynk. [9]



## Blynk

Blynk is a new platform that allows you to quickly build interfaces for controlling and monitoring your hardware projects from your iOS and Android device. After downloading the Blynk app, you can create a project dashboard and arrange buttons, sliders, graphs, and other widgets onto the screen. I have linked the Blynk application to the ESP32, setting it up to send a notification once the soil moisture sensor is activated as you can see in the images below.

```
void loop()
{
  Blynk.run();
  sensorValue = digitalRead(sensorPin);
  if (sensorValue == 1) {
    Blynk.notify("I need water..");
  }
}
```



## Relay module

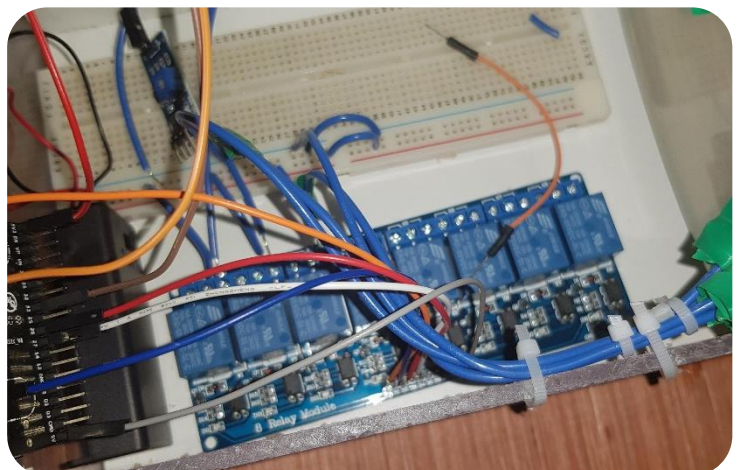
To allow control over the watering and lighting system in the Auto – Garden I have implemented an 8-channel relay module into the project. Relays are switches that open and close circuits electromechanically or electronically. Relays control one electrical circuit by opening and closing contacts in another circuit. As relay diagrams show, when a relay contact is normally open (NO), there is an open contact when the relay is not energized. In Auto – Garden I control the relays using the sliders in the control centre, once the slider is set to on the relay is (NO) and therefore energized.

```
for(int i=1; i<=NUM_RELAYS; i++){
  pinMode(relayGPIOs[i-1], OUTPUT);
  if(RELAY_NO){
    digitalWrite(relayGPIOs[i-1], HIGH);
  }
  else{
    digitalWrite(relayGPIOs[i-1], LOW);
  }
}
```

As you can see in the image, I have created a for loop that states if the relay is (NO) the ESP32 will set the correct GPIO pin to high. Allowing me to send power to the watering and lighting system.

```
<h2>Auto - Garden    Control and Data Center</h2>
%BUTTONPLACEHOLDER%
<script>function toggleCheckbox(element) {
  var xhr = new XMLHttpRequest();
  if(element.checked){ xhr.open("GET", "/update?relay="+element.id+"&state=1", true); }
  else { xhr.open("GET", "/update?relay="+element.id+"&state=0", true); }
  xhr.send(); }
```

As mentioned earlier this relay is fully controlled by the control centre in the Auto – Garden web page. On the control centre page, you will find 3 slider buttons which are connected to the pins 2, 26, and 27 of the ESP32. These pins are then connected to Relay 1, 2 and 3 on the relay module. Once the GPIO pins are activated they activate the Relays which creates a contact within the circuit allowing power to the outputs in this case the lighting and water pumps. [\[4\]](#)[\[5\]](#)



## Water Pumps

Obviously one of the most important aspects of gardening is the watering of your plants. The watering system in the Auto – Garden was always going to be the most vital part to get right. For this watering system I have used 5v water pumps to suck water out of a basin located in the back of the Auto – Garden and up through two tubes which are connected to the roof. I have placed little holes along these tubes to allow the water to escape through them and act as a sprinkler.



## Lighting System

For the lighting system in the Auto – Garden I have used red LED strip lights to act as infrared garden lighting. These lights are also 5 volts and powered by the 9-volt battery installed in the project. I have stuck the strip lights onto the roof also facing down on the garden as you can see in the images. I chose the strip lights as they were very handy to use and only required 5 volts which was important for me and how I would power them and the water pumps together.





# Auto-Garden Web Page

As mentioned already in this report the Auto – Garden web page was a main objective for the whole project. I wanted to create a page where the user could sign up and access all the features of my web application. In this section I will discuss the features of the Auto – Garden web page. I will provide images showing the layout and other cool aspects of the web page. I used NodeJS express and MongoDB to create this web application and used a development tool called Atom to write the code.

## Login

Like most web pages or applications Auto – Garden requires a log in to access all its features. The first page you will be greeted with on Auto-Garden is the log in page below. You will be asked to enter a username and password. If your details are correct you will be granted access if incorrect you will receive an error message. Once you sign in your data will be saved and you will be aloud to continue to the home page of Auto – Garden.

### Auto-Garden Account Login

The `input type="password"` defines a password field:

Username:

Password:

Login

Your Username is: cathal74

Your Password is: ireland1997

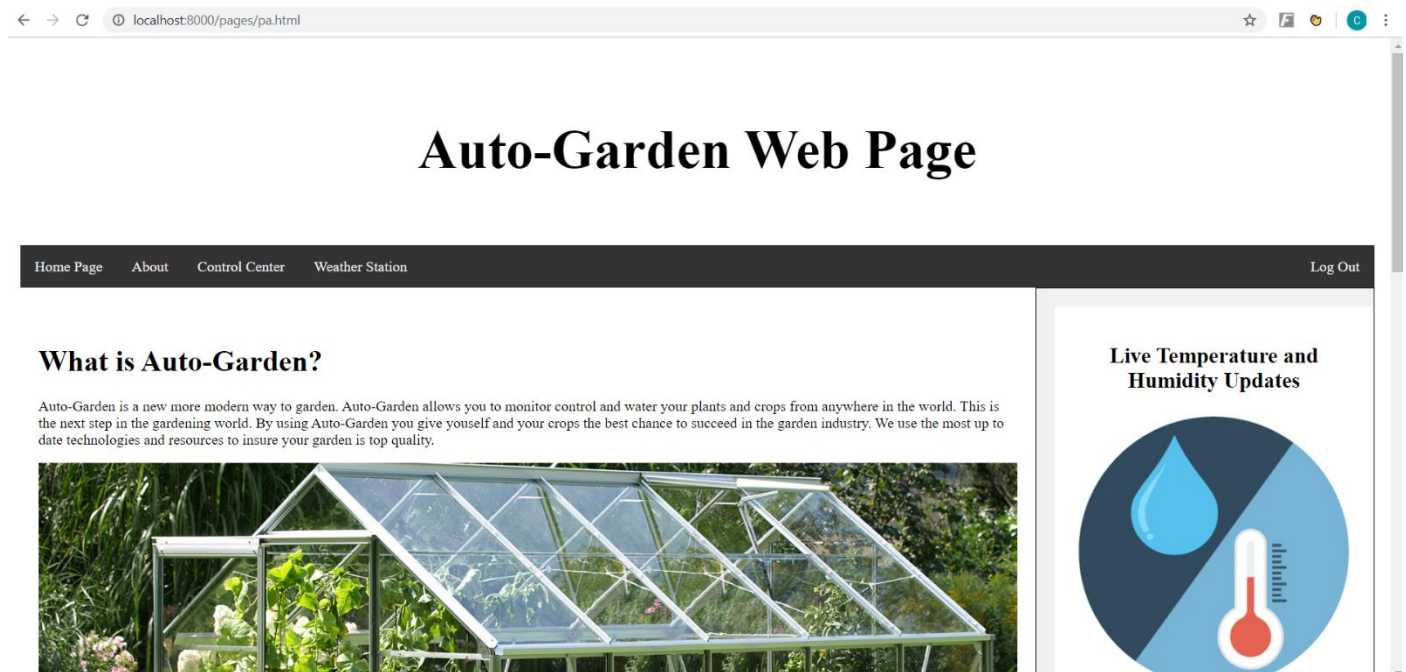
Continue To Home Page

On the home page of the Auto garden web page you will also see a log out button. Once you click this you will be redirected back to the Auto – Garden Account Login page and will have to enter your personal data once again to sign back in.



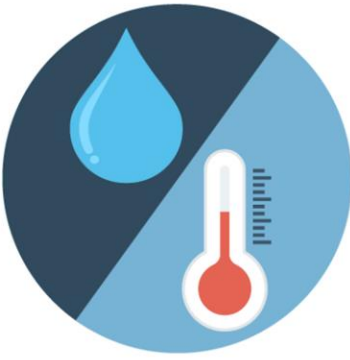
## Home Page

Once past the log in you will be brought to the Auto- Garden home page. The first thing you will notice on the home page is 4 modules on the left Home, About, Control Centre and Weather Station and one Logout module on the right.



On the home page itself there is some basic information on Auto – Garden itself and what it does. There is also a section with information on the ESP32 and a help centre that redirects you to a few helpful pages on how to set it up. To the right you will see a Live Temperature and Humidity Update centre. This receives live updates from the DHT11 which is in the garden and prints them out on the

**Live Temperature and Humidity Updates**



**This page will update every few seconds**

Humidity: 50%

Temperature: 14°C

[Update](#)

**HELP AND SUPPORT**

[Setting up ESP32](#) [Setting up Nodejs](#)

page so you can always keep track of the climate. The logout button redirects you back to the login page.

## Control Centre

Once you go to the control centre in the Auto – Garden web page you will be redirected to the ESP32 web server. On this web server there will be 3 different sliders which allow you to turn on and of the lighting and watering systems in the garden as you please. These sliders are connected to the GPIO pins of the ESP32. So once the slider changes to on it will send the data to the pins which will change the state of the relay it is controlled of. [\[4\]](#)[\[5\]](#)

# Auto – Garden Control and Data Center

Lights - Switch 1

WaterPump Switch 2

2nd WaterPump Switch 3



## Weather Station

The weather station section is connected to the DHT11 sensor. On this page you can see all the updates from the DHT11 and a live temperature and humidity feed which updates every few seconds allowing the user to monitor closely. [\[1\]](#)

Temperature -- °C

Humidity -- %

# Code

The main build tools used in this project was Arduino and Atom. I use Arduino to upload all the code onto the ESP32. So, all the control of the lighting system and watering system are done through Arduino. I also connect Arduino to AWS to allow AWS to publish my DHT11 Data to the NodeJs web application. All the code for the Web page itself is then done it Atom. I have written a lot of HTML in Atom to render each section on the web page. In this section I will go through some of the main bits of code used in the development of the Auto – Garden project and what it does.



ATOM

I also use Git and Github throughout the project to commit and push my code. Git is a distributed version-control system for tracking changes in source code during software development. GitHub is a Git repository hosting service, it also provides access control and several collaboration features, such as a wikis and basic task management tools for every project. These are very handy tools to use in the development of projects such as this as it allows you easily track changes in your code throughout the year and when the changes may have been made. After any changes to your code you can commit and push it to your repo.



git



## Arduino Code

Arduino is where most of the code for the hardware is written. In this code I have created an ESP32 web server. This is where the control centre runs. Up the top of the code you will see me connecting the ESP32 to the WIFI and AWS. This is essential to allow the ESP32 to run the web server and send the DHT11 data to AWS.

```
AWS_IOT hornbill;    // AWS_IOT instance

char WIFI_SSID[]="Cathals Hotspot";
char WIFI_PASSWORD[]="cathal74";
char HOST_ADDRESS[]="arn:aws:iot:us-east-1:452570201764:thing/AutoGarden";
char CLIENT_ID[]="AutoGardenClient";
char TOPIC_NAME[]="AutoGarden";
```

To collect the temperature and humidity data I have then set up two functions to read this data from the DHT11.

```
String readDHTTemperature() {
  Celsius (the default)
  float t = dht.readTemperature();
  // Read temperature as Fahrenheit (isFahrenheit = true)
  //float t = dht.readTemperature(true);
  // Check if any reads failed and exit early (to try again).
  if (isnan(t)) {
    Serial.println("Failed to read from DHT sensor!");
    return "--";
  }
  else {
    Serial.println(t);
    return String(t);
  }
}

String readDHTHumidity() {
  float h = dht.readHumidity();
  if (isnan(h)) {
    Serial.println("Failed to read from DHT sensor!");
    return "--";
  }
  else {
    Serial.println(h);
    return String(h);
  }
}
```

I have then a small bit of HTML to print the temperature and humidity readings to the web page. This prints out the temperature in degrees and the humidity as a percentage in the weather station.

```
</script>
<p>
  <i class="fas fa-thermometer-half" style="color:#059e8a;"></i>
  <span class="dht-labels">Temperature</span>
  <span id="temperature">%TEMPERATURE%</span>
  <sup class="units">&deg;C</sup>
</p>
<p>
  <i class="fas fa-tint" style="color:#00add6;"></i>
  <span class="dht-labels">Humidity</span>
  <span id="humidity">%HUMIDITY%</span>
  <sup class="units">%</sup>
</p>
</body>
</html>
\rawliteral":
```



To control the relay module from the ESP32 I first define the relay configuration, the number of relays being used and then assign the relay pins.

```
#define RELAY_NO    true
#define DHTPIN 33    // Digital pin connect

// Set number of relays
#define NUM_RELAYS 3
#define DHTTYPE     DHT11    // DHT 11
```

I then have a for loop to update the state of the relay module by sending high and low signals to the relay pin. I can stop the current flow by sending a high signal.

```
for(int i=1; i<=NUM_RELAYS; i++){
    pinMode(relayGPIOs[i-1], OUTPUT);
    if(RELAY_NO){
        digitalWrite(relayGPIOs[i-1], HIGH);
    }
    else{
        digitalWrite(relayGPIOs[i-1], LOW);
    }
}
```

I then have some Html that updates the state of the relay depending on what condition the button slider is. If the button slide is on it updates the relay state to true allowing the current to flow and sending power to the outputs of the project.

```
<h2>Auto - Garden    Control and Data Center</h2>
%BUTTONPLACEHOLDER%
<script>function toggleCheckbox(element) {
    var xhr = new XMLHttpRequest();
    if(element.checked){ xhr.open("GET", "/update?relay="+element.id+"&state=1", true); }
    else { xhr.open("GET", "/update?relay="+element.id+"&state=0", true); }
    xhr.send();
}
```

To print the button sliders, I have some HTML in the Arduino code to print them to the web server. [\[4\]](#)[\[5\]](#)

```
body {min-width: 200px; margin-top: 40px; padding-bottom: 20px;}
.switch {position: relative; display: inline-block; width: 120px; height: 68px}
.switch input {display: none}
.slider {position: absolute; top: 0; left: 0; right: 0; bottom: 0; background-color: #FF0000; border-radius: 34px}
.slider:before {position: absolute; content: ""; height: 52px; width: 52px; left: 8px; bottom: 8px; background-color: #fff; -webkit-
input:checked+.slider {background-color: #008000}
input:checked+.slider:before {-webkit-transform: translateX(52px); -ms-transform: translateX(52px); transform: translateX(52px)}
```

## HTML Code

All the HTML for the web page is then written in Atom. I have about 5 or 6 pages of HTML for the login, homepage, control centre, weather station and about section of the web page. These can all be found in my Github repository which I will include at the end of the Report.

To allow the user to access the ESP32 web server I have printed the IP address of the server into the HTML code. Once the control centre module is then clicked on it will redirect the user to the ESP32 web server.

```
<div class="header">
  <h1>Auto-Garden Web Page</h1>
</div>

<div class="topnav">
  <a href="/pages/pa.html">Home Page</a>
  <a href="/pages/pb.html">About</a>
  <a href="http://192.168.43.237/">Control Center</a>
  <a href="http://192.168.43.237/">Weather Station</a>
  <a href="/" style="float:right">Log Out</a>
</div>
```

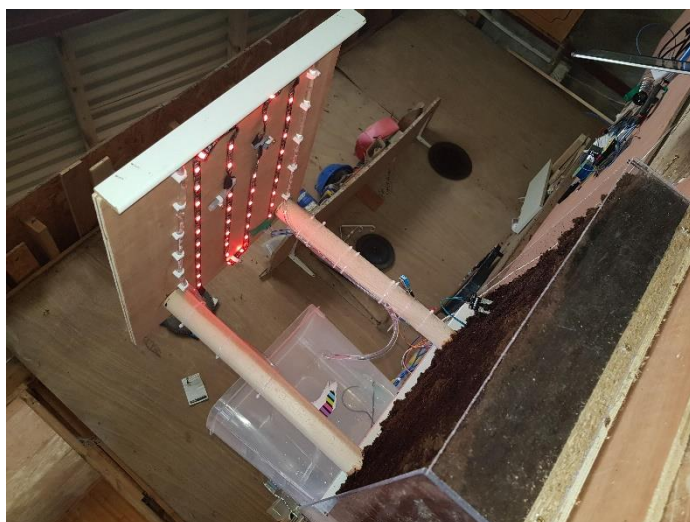
To allow AWS to publish the data I must connect AWS to the atom web application. To do this I have used the code shown below, linking all the certs and the host address.

```
var device = awsIot.device({
  keyPath: "C:\\nodejs\\MiniProject\\certs\\df7b8df7ef-private.pem.key",
  certPath: "C:\\nodejs\\MiniProject\\certs\\df7b8df7ef-certificate.pem.crt",
  caPath: "C:\\nodejs\\MiniProject\\certs\\AmazonRootCA1.pem",
  clientId: "qwerty",
  host: "a3tpzmz206oll8-ats.iot.us-east-1.amazonaws.com"
});
```

Aws publishes the DHT11 data to node js which then renders it to the web application.

# Auto - Garden build

The building of the Auto – Garden itself was quite a tricky task. Most of the garden itself is build out of wood. I started buy building a small rectangular box with a wooden base and used Perspex to go around the sides to allow you to see the soil. I split the rectangular box with a wooden plank splitting up the garden section and the electronic sections for safety as there will be a lot of water about. In the back section I have a large plastic basin to hold the water with two pumps laying in it and then all the electronics located beside it isolated from the rest of the project.



The lighting and watering system are then brought up two wooden poles cut off a shovel acting as two columns for the roof itself. On the roof piece I have stuck on two tubes going across from the water pumps in the plastic container at the back section and stuck the LED strip lights here too shining down on the garden itself. I then cut multiple holes in the tubes to allow water to sprinkle down on the garden section.

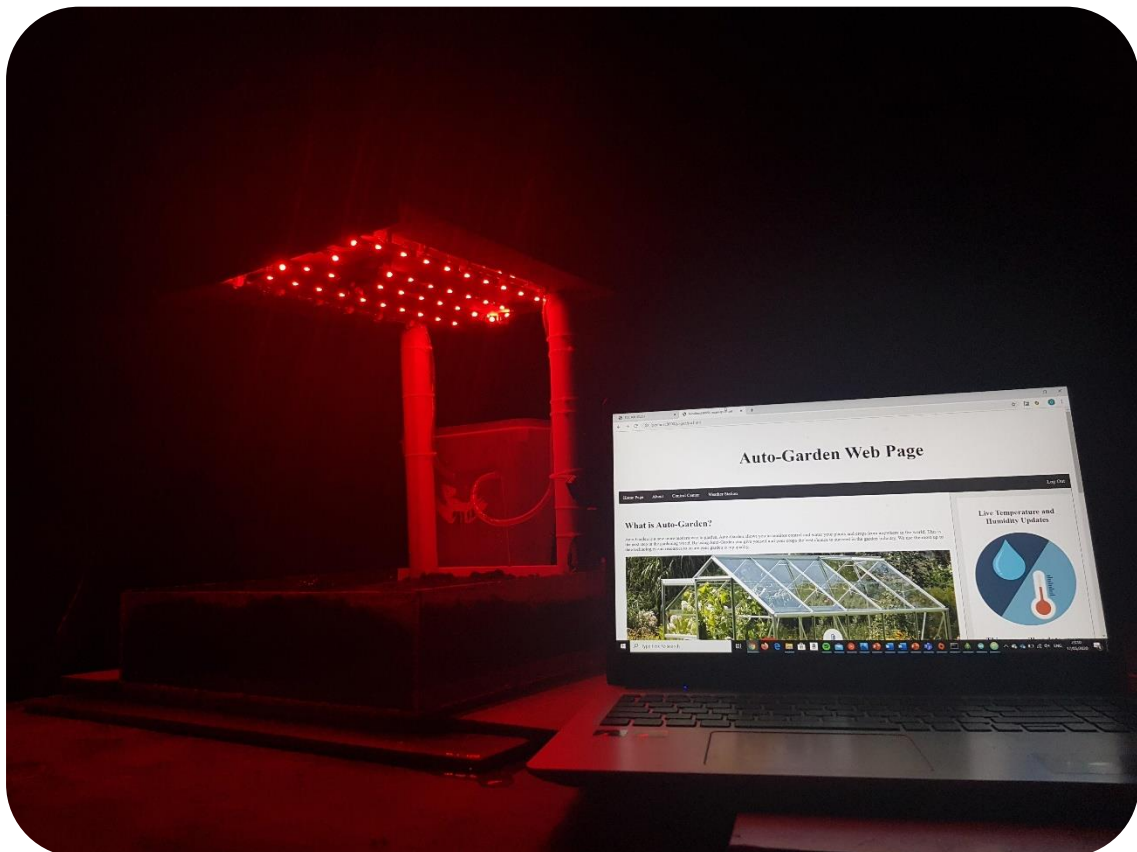


# Conclusion

Overall, I am very happy with how Auto – Garden turned out. When I started this project last year, I knew it would be a massive challenge and that it was and caused me some great problems along the way. However, I got through these and it was a great learning experience. I have learnt so much from this project and developed so much as an engineer.

The goal at the start was to have a fully functional web page which allows the Auto – Garden users full control over their garden from their own PCS. I think I have achieved this goal and am very happy with the outcome.

There is so much more I can still add to this project and endless opportunities to upgrade. Due to time restrictions all of these were not possible, but I look forward to adding to Auto – Garden in the future and who knows where it might end up.





# Project Details

<https://github.com/cathal74/Project>



# GitHub

<https://www.youtube.com/watch?v=lwPjZq3l6OI&t=1s>



# References

- [1] <https://github.com/adafruit/DHT-sensor-library>
- [2] <https://aws.amazon.com/iot/>
- [3] <https://en.wikipedia.org/wiki/ESP32>
- [4] <https://randomnerdtutorials.com/esp32-relay-module-ac-web-server/>
- [5] <https://randomnerdtutorials.com/esp32-web-server-arduino-ide/>
- [6] <https://www.w3schools.com/html/>
- [7] <https://aws.amazon.com/getting-started/tutorials/deploy-code-vm/>
- [8] <https://www.sitepoint.com/build-simple-beginner-app-node-bootstrap-mongodb/>
- [9] [https://www.youtube.com/watch?v=NN5c0eO\\_dnM&t=128s](https://www.youtube.com/watch?v=NN5c0eO_dnM&t=128s)