

Internet of Things Project

Smart Bedroom

Bartlomiej Drapinski

Bachelor of Software & Electronic Engineering

Atlantic Technical University

2024/2025

A screenshot of a computer

AI-generated content may be incorrect.

**Declaration**

This project is presented in partial fulfilment of the requirements for the degree of Bachelor of Engineering in Software & Electronic Engineering at the Atlantic Technical University, Galway campus.

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.

A close up of a sign

AI-generated content may be incorrect.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Bartlomiej Drapinski

**Table of Contents**

[1 Summary 6](#_Toc197626369)

[2 Project Plan 7](#_Toc197626370)

[3 Project Architecture 8](#_Toc197626371)

[4 Sensors 9](#_Toc197626372)

[4.1 Temperature and Humidity Sensor (DHT11) 9](#_Toc197626373)

[4.1.1 DHT11 Sensor Code 10](#_Toc197626374)

[4.2 Temperature and Humidity Sensor (AHT20) 14](#_Toc197626375)

[4.2.1 AHT 20 Sensor Code 16](#_Toc197626376)

[4.3 Light Sensor (DFRobot\_B\_LUX\_V30B) 18](#_Toc197626377)

[4.3.1 Light Sensor Code 19](#_Toc197626378)

[4.4 Motion Sensor (PIR sensor) 20](#_Toc197626379)

[4.4.1 Motion Sensor Code 21](#_Toc197626380)

[5 ThingSpeak 21](#_Toc197626381)

[5.1 ThingSpeak Code 22](#_Toc197626382)

[6 LED + 2 Buttons 24](#_Toc197626383)

[6.1 Button + LED Code 24](#_Toc197626384)

[7 Motors 25](#_Toc197626385)

[7.1 Servo Motor (FS90) 25](#_Toc197626386)

[7.1.1 Servo Motor Code 26](#_Toc197626387)

[7.2 Stepper Motor 27](#_Toc197626388)

[7.2.1 Stepper Motor Code 28](#_Toc197626389)

[7.2.2 Stepper Motor Power Requirements 29](#_Toc197626390)

[8 Web Server 30](#_Toc197626391)

[8.1 Wi-Fi 30](#_Toc197626392)

[8.1.1 Wi-Fi Code 30](#_Toc197626393)

[8.2 Web Server 31](#_Toc197626394)

[8.3 HTML & CSS Code 32](#_Toc197626395)

[8.3.1 Responsive Design 33](#_Toc197626396)

[8.4 Web Design for Accessibility 35](#_Toc197626397)

[9 Problem Solving 36](#_Toc197626398)

[10 Impact of Project on Sustainability 37](#_Toc197626399)

[11 Conclusion 38](#_Toc197626400)

[12 References 39](#_Toc197626401)

[Appendix 1: Code 40](#_Toc197626402)

[Appendix 2: Bill of Materials 41](#_Toc197626403)

# Summary

The goal of my project is to offer a straightforward way for someone to be able to control all the common features found in a bedroom like blinds and a ceiling light.

Key features of my project when looking at the UN’s SDGs are to reduce our electrical and energy consumption. In this project, I have a motion sensor that actively monitors for presence in the room and therefore turns on or off a light when needed, saving on electricity costs. The project also has a servo motor that controls a vent. It actuates according to sensor readings from 2 temperature and humidity, one placed inside and one outside. If it gets too warm, the vent opens and when it gets too cold, the vent closes, saving on heating costs.

This project started when I had the idea of how to enable myself to control things that are in my bedroom but in most well-fitted ones as well. I have found that it was a pain to go over to the window to operate my roller blinds. I then set up to solve my problem by getting a stepper motor connected to the pull string that would operate the blinds for me. I then set to enable myself to control them via my phone via a web server.

When I had one motor, I thought to myself, why not add a second motor. The second motor is a servo that can move from 0 degrees to 180 degrees and do it precisely. The servo is controlling a vent that I have on my window, but a vent built into a wall with a handle will still work. The aim of the vent is to actively regulate the temperature and humidity of the room while also saving on heating costs.

This project has incorporated an ESP 32 Development Board and ThingSpeak as the center of the project and a breadboard to connect all my sensors and peripherals together. Arduino IDE and Visual Studio Code coding software to put this project to life.

Also, a big contributor to code organization goes to GitHub for easy sharing and version control.

I think that I have ticked off all the main objectives that I had at the beginning of this project that slowly expanded with cool features and aesthetic display of the project.

# Project Plan

(Order of tasks may not correspond to actual sequence of events)

Start of Project

* Get a project description made and sent for review.
* Make a basic architectural diagram based on description.
* Create a detailed Bill of Materials for the project.

Ordering Components / Research Components

* Wait for components to arrive.
* Research on how to connect / code said components.

Testing of Components / Features

* When majority of components arrive, start testing each component individually for proper function to then add this code to the main code. Evidence on GitHub.
* Debug code.

When all Components are individually tested, code combining starts

* Combine code to work with each other.
* Lots of debugging.

Create simple Web Server in Visual Studio Code [11] for Christmas Demo

* Test ideas using live server in VS Code.

After first building the Web Server in VS Code, now running it from the ESP32

* Adapt HTML and CSS code.
* Use Web Server libraries WiFi.h and WiFiClient.h to make it work.

Prepare project for final Demo

* Project DONE

# Project Architecture

For my project I am using the ESP32 Development Board [1] and the Arduino IDE v2.3.4 [2].

The ESP32 Development Board is a versatile board that is primary used in IoT projects like my Smart Bedroom. It allows for fast prototyping as the board can be inserted into a breadboard for ease of connecting inputs like my temperature and humidity sensors and outputs like my stepper motor working the blinds.

To program the ESP32, I only used the Arduino IDE (Integrated Development Environment). This software supports many basic development boards like the Arduino Uno or Mega. The Arduino IDE uses C++ as the programming language.

I used Thingspeak [3] to graph my results and later make indicators and gauges with it.

At the start of this project, I did utilise Wokwi [4] to see how some of my components would connect to the ESP32.

To create this architecture diagram, I used the online tool called Draw.io [5]. It is easy to use and has many templated and guide on making a good diagram.

A close-up of a computer chip

AI-generated content may be incorrect.

Figure 3. Architecture Block Diagram

# Sensors

My project uses many sensors. I have 2 temperature and humidity sensors, the DHT11 and the AHT20. Both sensors function in a different way and use different communication protocols to talk to the ESP32.

## Temperature and Humidity Sensor (DHT11)

The DHT11 sensor comes in a blue plastic package that has a grid to allow airflow into the sensor. It has 4 wires coming out of it. When the sensor is lying flat, the pins from left to right are **Vcc**, **Data**, **NC** and **GND**.

**Vcc** is connected to the 5V power rail of the ESP32. 3.3V is sometimes used but unreliable.

**Data** is connected to any digital input on the ESP32.

**NC** is not connected to anything in the sensor and is used as mechanical support when soldering to a board.

**GND** is connected to the Ground rail of the ESP32.

When buying this sensor, it is often found that the package is soldered to a breakout board that has other components. They usually have a capacitor of decoupling and a 10k Ω pull-up resistor on the **Data** pin going to **Vcc**.

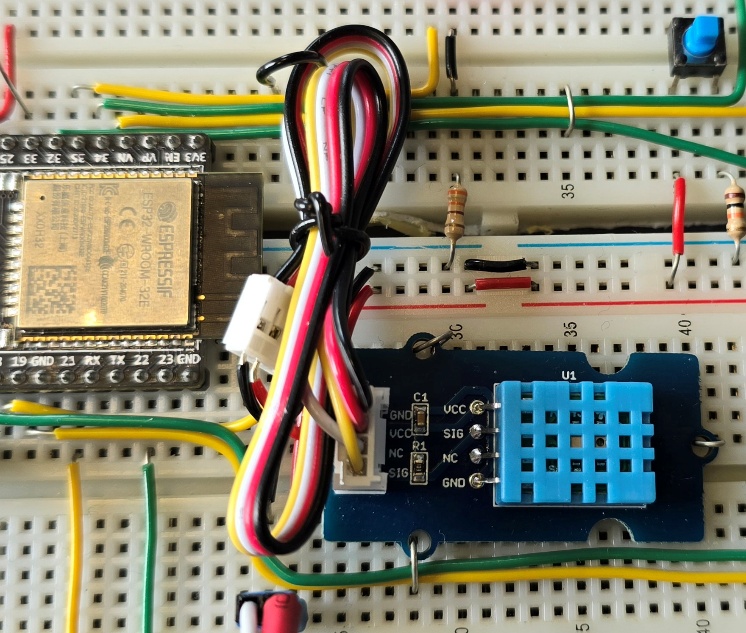


Figure 4..1 DHT11 Sensor Circuit

A diagram of a circuit

AI-generated content may be incorrect.

Figure 4.1.2 DHT11 Circuit Diagram

In my project the DHT11 sensor monitors the inside temperature and humidity. With these sensor values, it controls when a vent should open and close controlled by a servo motor.

### DHT11 Sensor Code

To program the DHT11 sensor, I found a library in the Arduino IDE that will simplify the amount of code I need to write to get the sensor to return the temperature and humidity values.

The library I found is the DHT.h. This library supports the few variants of the DHT like the DHT11 that I am using as well as DHT 12, 21 and 22. It also supports the AM2301 sensor which is a different version to the DHT 21.

To start programming the sensor, it is always important to properly include the library. All libraries or custom function should be located at the top of your code. Make sure that you have the library installed as different computers may not have them installed.

The library should look like this: #include <DHT.h>

After the library is added to your code, the ESP32 must know to what pin the sensor is connected to. This is achieved by defining a variable of name you choose but make it meaningful and assigning it a number, the pin number you connected the sensor to.

It should look like this: #define DHT11PIN 18 Note: No semicolon (;) is needed here

This is part of setup code that many sensors and peripherals need to function. Another key bit of code is the sensor initialization done by the library. Most libraries have a function that you need to pass value like the pin definition. In the case of the DHT11, it looks like this:

DHT dht(DHT11PIN, DHT11);

It names the function and passes the pin number defined earlier in the code and the name of the sensor you have. Note that the name of the pin definition does not have to include DHT or 11 or PIN, it is solely used to substitute the actual pin number to a name, this is useful when you need to change the pin number, you can do it from the top of the code where many other definitions live.

Now we can start coding for the sensor. When opening a new sketch, 2 functions appear, void setup() and void loop(). The library and pin definition goes above these functions.

Figure 4.1.3 shows a typical void setup() function. It includes the line “Serial.begin(115200)”. This line initialises the Serial Monitor so that we can see our sensor values. The big number is the baud rate of the connection from the computer to the ESP32. It sets the speed of transmission. As the ESP32 is faster than an Arduino Uno that uses 9600 baud, it can use the higher baud rates available.

The second line is custom to the DHT library I got, and it starts the sensor. In here the code runs only once on power up.

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 4.1.3 DHT11 void setup() code

Figure 4.1.3 shows the void loop() function. Here the code within will run forever if the ESP32 has power or it crashes for some reason.

Here the programmer can change how they want to receive the sensor values and what to do with them. To start off, the two variables that the sensor will return need to be defined first. In this code, both values are set a float. This means that the values will return to a decimal point, e.g., 21.49764. Both variables have a name that then equals a function that will return the specified value. Those functions are again set by the library. When the values have been received, they are printed to the Serial Monitor along with some descriptions as well.

To print to the Serial Monitor, the Serial.print or Serial.println functions can be used with text when using double quotation marks or just with the variable name.

The delay function of 1000ms or 1 second is there to not overwhelm the sensor and yourself with too many results.

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 4.1.4 DHT11 void loop() code

A screen shot of a black screen

AI-generated content may be incorrect.

Figure 4.1.5 DHT11 Serial Monitor Output

## Temperature and Humidity Sensor (AHT20)

The AHT20 [6] is the successor to the DHT11 as it comes with many improvements like using the I²C communication protocol and having a greater resolution. The sensor uses surface mount technology (SMT) that needs a breakout board to connect it to the ESP32. The breakout board has 4 pins to connect the sensor to A breadboard. From left to right, **Vcc**, **GND**, **SCL** and **SDA**.

**Vcc** is connected to the 5V power rail of the ESP32. 3.3V is sometimes used but unreliable.

**GND** is connected to the Ground rail of the ESP32.

**SCL** stands for Serial C lock. SCL is connected to pin 22 of the ESP32.

**SDA** stands for Serial Data. SDA is connected to pin 21 of the ESP32.

The I²C protocol is very interesting as you can connect up to 128 devices on the same 2 wires thanks to each device having a unique address. The sensors are called slaves and a ESP32 or Arduino Uno act as masters in the chain.

To communicate with sensors, the master sends out a request to return data from a specific device using its unique address and the device then sends that data on the same **SDA** wire.

As I²C can have many devices on the same 2 wires, signal integrity and ensuring a stable bus is crucial. To mitigate this, 10k Ω resistors are connected to the **SCL** and **SDA** and pulled to Vcc. This ensures that when no device is using the connection, both lines are at a known high state.

A circuit board with wires and a blue square

AI-generated content may be incorrect.

Figure 4.2.1 AHT20 Sensor Circuit

A black circuit board with wires

AI-generated content may be incorrect.

Figure 4.2.2 AHT20 Sensor Circuit Close-Up

A computer screen with a diagram

AI-generated content may be incorrect.

Figure 4.2.3 AHT20 Circuit Diagram

Figure 4.2.2 shows how the AHT20 is connected to the ESP32. As I also have a Light Sensor that uses I²C protocol, I have decided to create a I²C bus just like it was intended to use I²C for. So, from the ESP32, **SCL** is the green wire, and **SDA** is the yellow wire that use the power rail of my breadboard to make the I²C bus neat so that when a sensor needs to branch of it can.

In my project the AHT20 sensor monitors the outside temperature and humidity. I have found that it has greater accuracy which I liked. This sensor does not change other parts of the project, it is to have a history of the temperature and humidity readings that can aid in deciding on what to do outside, e.g., garden work.

### AHT 20 Sensor Code

To program the AHT20, I again used a library. I found the Adafruit AHTX0 library when searching online. Adafruit is a well-known sensor company, and their datasheet [7] had some good examples that I modified to suit the way I wanted to display the values.

The library is #include <Adafruit\_AHTX0.h> and it goes on top. Then some initialization code for the AHT20 which look like this. Adafruit\_AHTX0 aht; It does not define the pin numbers for **SCL** and **SDA** as the library already knows what are the I²C pins on the ESP32 as they are fixed.

In void setup() it has more features that make the sensor work better. In Figure 4-5, the library has the aht.begin function but now it is enclosed in an if statement, looking for when the sensor may fail or not get recognised from a poor connection or faulty unit.

The Serial Monitor also get initialised.

A computer code with white text

AI-generated content may be incorrect.

Figure 4.2.4 AHT20 void setup() code

Figure 4.2.4 shows the more complex void loop() compared to the DHT11 sensor. Although this code is from the manufacturer’s datasheet, I did research and see that temperature and humidity variables are defined as a struct variable that comes from the library. They are like an int or a float.

The code then calls a function that passes a reference to the above variables. When the function returns the values, they are printed onto the Serial Monitor.

A screen shot of a computer code

AI-generated content may be incorrect.

Figure 4.2.5 AHT20 void loop() code

A screenshot of a computer

AI-generated content may be incorrect.

Figure 4.2.6 AHT20 Serial Monitor Output

## Light Sensor (DFRobot\_B\_LUX\_V30B)

The light sensor is used to measure the light intensity of its surroundings. The sensor is mounted outside or facing outside to primarily detect when it gets dark and when the sun is rising in the morning. From these sensor readings, it will control blinds up and down that are actuated using a stepper motor.

This light sensor also utilizes the I²C protocol for ease of communicating with the ESP32. The sensor also has one pin to signal to the sensor to output data.

It is a round module that has a plastic dome to protect the sensor and allow for better light spread over the sensor for more accurate readings.

The sensor has a connector with pin numbers. From pin 1 it is **Vcc**, **SCL**, **SDA**, **GND** and **EN**.

**Vcc** is connected to the 5V power rail of the ESP32. 3.3V is sometimes used but unreliable.

**SCL** stands for Serial C lock. SCL is connected to pin 22 of the ESP32.

**SDA** stands for Serial Data. SDA is connected to pin 21 of the ESP32.

**GND** is connected to the Ground rail of the ESP32.

**EN** pin enables or disables the sensor. Logic High to enable and logic Low to disable the sensor.

As this sensor uses the I²C protocol, the **SCL** and **SDA** pins connect to my I²C bus where the AHT20 also connects to. The **EN** pin connects to digital input pin 19 on the ESP32.

If I had the chance using components that all use the I²C protocol it would greatly simplify the amount of wiring and make troubleshooting easier. Although having many I²C devices may complicate the code required to make everything work. It would provide a nice coding challenge.

A diagram of a light sensor

AI-generated content may be incorrect.

Figure 4.3.1 Light Sensor Circuit Diagram

### Light Sensor Code

For this sensor I used the DFRobot\_B\_LUX\_V30B library [8] for the model I have. Just like the other libraries, including the library is straight forward. The next step is initialising the sensor with the function call. Here is where that enable pin is needed.

This void setup() is simple with the Serial Monitor and sensor initialisation.

A screen shot of a computer code

AI-generated content may be incorrect.

Figure 4.3.2 Light Sensor void setup() code

The void loop() in this code is also like the AHT20 and DHT11. To get the light intensity values the code calls the function and the library does the rest.

A screen shot of a computer code

AI-generated content may be incorrect.

Figure 4.3.3 Light Sensor void loop() code

## Motion Sensor (PIR sensor)

The motion sensor I am using for my project is the HC-SR501. It is a generic module that is usually found to have a white dome over some circuitry. The dome is a Fresnel lens to increase the sensors wide-area range and sensitivity. The sensor detects the changed in infrared light that is emitted by our body heat. See Figure 4.4.1 for a closeup of the Fresnel lens.

The sensor I got is like the HC-SR501 but shaped differently and has 4 pins, although one says NC which might have been an optional feature on my board.

**Vcc** is connected to the 5V power rail of the ESP32. 3.3V is sometimes used but unreliable.

**SIG** is the data output of the sensor. It outputs a logic 1 when it senses movement and logic 0 when it is actively monitoring its surroundings.

**GND** is connected to the Ground rail of the ESP32.

The sensor has 2 potentiometers that control the sense and on time. They can be adjusted with a small Philips screwdriver (+). See Figure 4.4.2

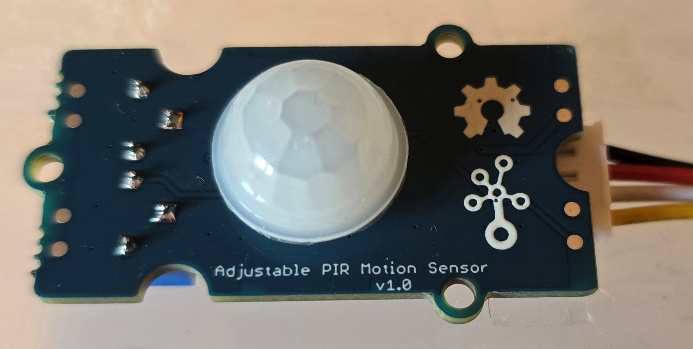


Figure 4.4.1 Motion Sensor - Top View

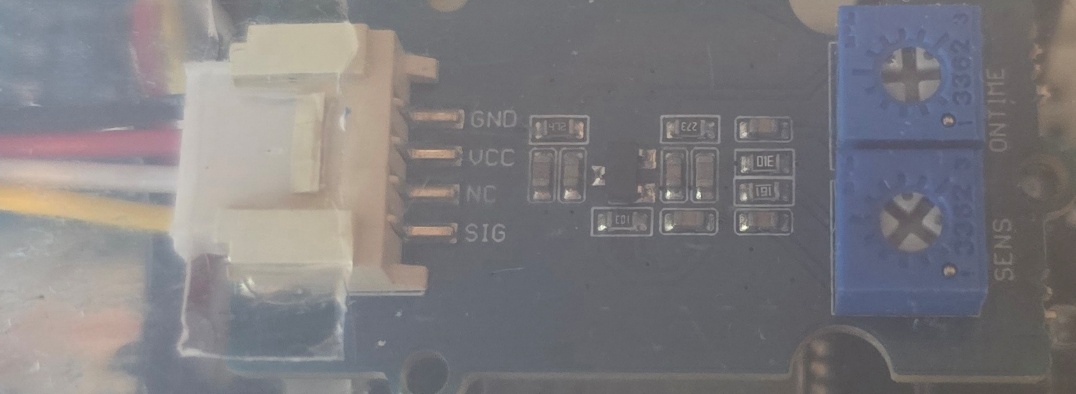


Figure 4.4.2 Motion Sensor - Bottom View

### Motion Sensor Code

The code to get the sensor working is not challenging nor long. All I am doing is polling the ESP32 pin the sensor is connected to using an if statement.

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 4.4.3 Motion Sensor Start Code

In Figure 4.4.3, I have defined the pins of the sensor and the LED. The const unsigned long variable sets the time the sensor will not trigger again.

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 4.4.4 Motion Sensor void loop() code

In the void loop, I have the millis() function that counts from zero and will determine how long the sensor is triggered and subsequently turn the LED on or off.

# ThingSpeak

This tool enabled me to create user friendly and easy to read graphs and later some big colour indicators to show status of my project and some gauges to display the temperature and humidity readings.

It is a free service that was offered to us to enable us to store sensor data and generate graphs and gauges that I used in my project.

ThingSpeak [3] is owned by MathWorks, and they offer a free account that allows for data to be sent to them every 20 seconds. This service’s limitation created a problem as my project has a PIR motion sensor that updates every second and so it may not send the actual motion state to ThingSpeak which can change in the 20 seconds the ESP32 is waiting.

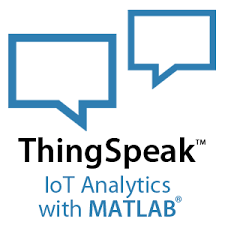


Figure 5.1 ThingSpeak Logo

## ThingSpeak Code

To send any data to ThingSpeak [3] there are a few parameters to set. First are your Wi-Fi credentials so that the ESP32 can talk with ThingSpeak servers. Then there is your Channel ID number, and an API write key. This is all the private information that must be defined somewhere in the Arduino IDE sketch. As my project was going to be connected to GitHub for version control and ease of sharing (to the public), these variables cannot be inside a normal file that then is uploaded to GitHub.

I have put all this sensitive data into a header file I called secrets.h that I could then mask from the uploads and only keep them on my computer. Inside this file I have 6 #defines. They are SECRET\_SSID and SECRET\_PASS, SECRET\_CH\_ID and SECRET\_WRITE\_APIKEY twice for both of my channels.

The API key allows the key holder to authenticate and authorize the transmission of data.

In Figure 5.2, is a small part of the code required to send any data to ThingSpeak. First, I assign each ThingSpeak field with the data variable and to what field I want the data to go to. Then the ThingSpeak library uses the Channel ID and API key defined earlier to send the data. This function can return a value describing the status of ThingSpeak. The value is defined as x, and so the code can check for HTTP code 200 which indicates successful data transmission. If any other number like 301 shows up, this means the ESP32 lost its internet connection and communication with ThingSpeak.

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 5.2 Code to send Data to ThingSpeak

# LED + 2 Buttons

My project uses 2 momentary push buttons that I use as physical buttons for controlling my blinds. Both buttons have a 10k Ω pull down resistor. So, when pressing the button, I send a logic 1 to the ESP32. I have made the code so that when you press the button once, the LED turns on and when you press the button again, the LED turns off. The LED has a 330 Ω current limiting resistor and is active logic 1.

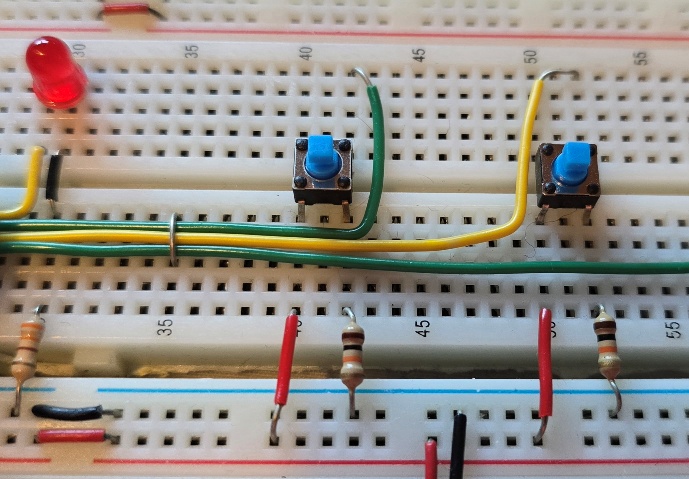


Figure 6.1 LED + 2 Buttons Circuit

The code for the button and LED can be combined as I have tested them together anyways.

## Button + LED Code

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 6.2 LED + 2 Buttons code

In Figure 6.2, with the 2 buttons, I have implemented a latching feature that compares the current logic state of the buttons to the last. I included a delay of 50 ms which is the typical debounce time for the button. It is there to prevent false triggers that can occur when 2 metal plates do not fully connect and tend to bounce up and down on a microscopic level, which to the ESP32 means the button was pressed multiple times in a short amount of time.

The code then drives the LED accordingly.

# Motors

My project has 2 types of motors, a servo motor and a stepper motor. They both control moving things in my project. The servo is low power and low torque, but the stepper motor delivers a big amount of torque so that it can move big blinds up and down.

## Servo Motor (FS90)

For my Demo and Video Presentation, I unfortunately had to remove the servo motor that I had planned to use for opening and closing a vent according to the inside temperature and humidity. I removed it as to not clutter the project presentation with a component that does not work the way I wanted it.

I deemed it necessary to still describe it and explain the trouble I was having with it.

The library I used is the ESP32Servo [9]. With the name saying ESP32, it must work without a problem, I thought. I have worked with this model of servo on the Arduino Uno, and it worked there.



Figure 7.1 FS90 Servo Motor

The servo has 3 wires. Yellow for SIG, Red for Vcc and Brown for GND.

At the start of the project, I had the servo SIG pin directly connected to the ESP32. The servo was moving but not in the way that I expected. Later, I found that I would need a 220 Ω resistor in between the SIG pin and the ESP32. Therefore, I broke the first servo motor probably a long time ago and didn’t realise. The motor may have been moving but the logic inside may have been damaged by the ESP32’s voltage on the SIG pin. I cannot confirm that this is true but with a new servo and the resistor, it did work. This was about before the Easter holidays when I got the servo working but then I decided, I won’t have enough time to get this part finished.

### Servo Motor Code

For testing of the servo, I wrote code to move the servo left and right with keyboard input. The keyboard input would have eventually been controlled by a threshold for temperature minimum and maximum.

The servo library [9] gets a degree value between 0 and 180 and the function myServo.write(90) moves the servo to that position.

A computer screen shot of a program code

AI-generated content may be incorrect.

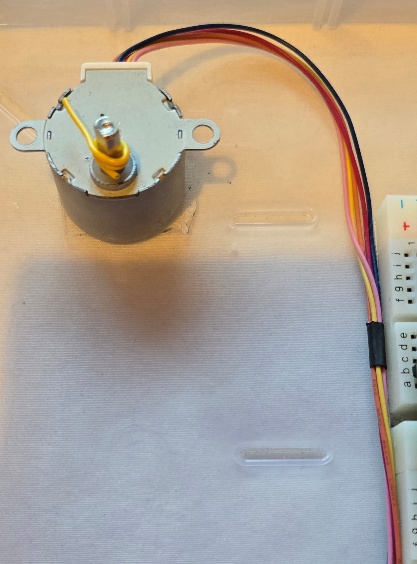
Figure 7.2 FS90 Servo Motor code

## Stepper Motor

I am very happy that my stepper motor is working fully. It has a dedicated stepper motor driver, the UNL2003. It is paired with the 28BYJ-48 stepper motor.

It needs the dedicated driver as the ESP32 cannot drive the 4 coils of the stepper motor directly. Average max current per pin on the ESP32 is about 40mA.

The ULN2003 gets a 4-bit code on the IN1, IN2, IN3, IN4 pins. The 4-bit code gets power boosted. The power for the ULN2003 is from an external power supply.



A circuit board with wires

AI-generated content may be incorrect.

Figure 7.3 Stepper Motor + Driver Circuit

### Stepper Motor Code

I used the Stepper library [10] that need one value to be set about the motor you have. It is the stepsPerRevolution value. It is number of steps my motor needs to make a full 360 degrees. This also sets the accuracy of the motor. To get greater accuracy just use a servo motor.

The IN1 to IN4 pins of the driver are defined. Pins 4, 16, 17 and 5 are all together so that the 4 wires can form a custom 4 pin connector.

In void setup(), using the library, I can set the speed of the motor.

In void loop(), the function myStepper.step(stepsPerRevolution) drives the motor. To reverse the direction, I put a minus in front of the stepPerRevolution value.

Although the library sets the 4-bit code for the movement. I found the sequence it may use depending on what you need, less torque and less power or more torque and more power.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | LESS TORQUE, LESS POWER | | | |  | MORE TORQUE, USES MORE POWER | | | |
| Step | IN1 | IN2 | IN3 | IN4 |  | IN1 | IN2 | IN3 | IN4 |
| 1 | 1 | 0 | 0 | 0 |  | 1 | 1 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 |  | 0 | 1 | 1 | 0 |
| 3 | 0 | 0 | 1 | 0 |  | 0 | 0 | 1 | 1 |
| 4 | 0 | 0 | 0 | 1 |  | 1 | 0 | 0 | 1 |

A computer screen shot of a program

AI-generated content may be incorrect.

Figure 7.4 Stepper Motor code

### Stepper Motor Power Requirements

As the stepper motor is a bigger motor compared to the servo and the stepper can deliver more torque, it also requires more power and because of that my project has a dedicated USB port for external power that I can connect to a powerful wall adapter. I use an external power supply and not my laptops USB ports as they can only deliver up to 500mA of current and the motor can draw up to 1A of current when stalled.

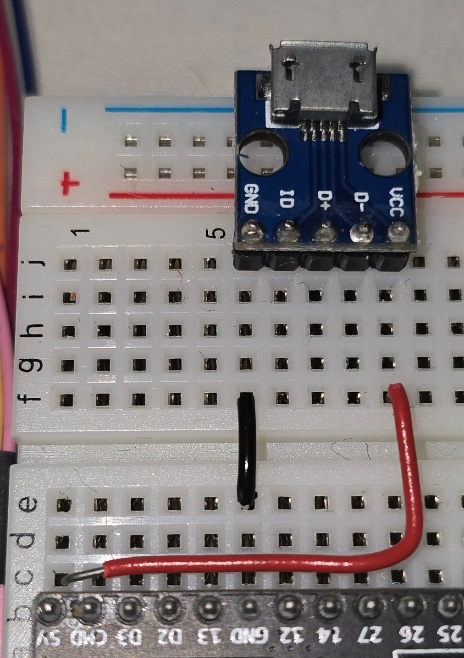


Figure 7.5 External Power Supply USB Connector

# Web Server

I had a simple Web Server running from Visual Studio (VS) Code 1.99.3 [11] at the end of the first semester. I used this Web Server for my Christmas Demo. When starting to work on for the second semester, I had a rough sketch of the layout that I wanted to achieve, I don’t have that drawing but I went for giving the user lots of information from both of my temperature and humidity sensors and big status indicator for all the features of my project.

## Wi-Fi

To get Wi-Fi for my project I have used my laptops hotspot. It works just like the one found on phones, just on a laptop which has its advantages and disadvantages.

The advantage of using my laptop as the source of internet for the ESP32 is that I can view the Web Server on the laptop and don’t need my phone. Some projects may like having a small screen but for my project, I needed the large screen to display everything clearly.

The disadvantage of using this hotspot compared to the one on any phone is it constantly changed the IP address. It was the only frustrating part about using this hotspot.

Not related to my hotspot, the Wi-Fi in and around Room 828 is horrible. I think there is an Access Point down the corridor. This is an electronics lab, that with current trends for IoT projects and more, will require some good coverage. And on the topic of Wi-Fi, I would be nice if that area would have their own dedicated IoT Wi-Fi that the computers in the lab could access.

### Wi-Fi Code

The code that sets up the connection is simple as everything is managed by 2 libraries, Wifi.h and WifiClient.h. [12] They both contribute to getting the ESP32 connected to the internet.

In Figure 8.1, WiFi.begin gets the details stored in that secrets.h header file where the ThingSpeak variables are stored as well. Then the code will wait until it gets connected to the network you specified the details for. It will print a dot to the Serial Monitor making a soft of timer for how long the ESP32 waits for the connection.

A screen shot of a computer code

AI-generated content may be incorrect.

Figure 8.1 Wi-Fi Connect code

## Web Server

My final Web Server was hosted on the ESP32 itself, but it wasn’t done like that from the beginning. I used VS Code [10] to write my HTML and CSS code and use a tool in VS Code to run a Web Server. This way I had the advantage of working on the Web Server and see my changes live. While on the finished product, the ESP3 running the Web Server, must be reprogrammed every time I made a small change to either the HTML or the CSS code.

To get the Web Server running on the ESP32 I used sample code of the WiFi and WiFi Client libraries.

A computer screen shot of text

AI-generated content may be incorrect.

Figure 8.2 Web Server code

Figure 8.2 shows two functions required to set up a Web Server on the ESP32. In the handleRoot() function, the webpage that is stored in a header file just like the Wi-Fi secrets, gets ready to be sent. The variable String message encloses all the HTML and CSS code and the server.send function sends it as text, indicating that it is HTML code. Code 200 is the HTML code for successful transmission. ThingSpeak also uses code 200 to show that it sends its data successfully.

A computer screen shot of white text

AI-generated content may be incorrect.

Figure 8.3 Web Server void setup(void) code

In Figure 8.3, it shows how the sends the Web Server and how it handles any errors that can occur using the handleNotFound() function in Figure 8.2.

## HTML & CSS Code

HTML or Hyper Text Markup Language is the start of any website. To make it looks nice, CSS or Cascading Style Sheets is where that HTML code can be manipulated to create interactive designs, add colour to your website and more.

For simplicity, here is only the HTML and CSS code that is running on the ESP32.

To continue from Chapter 8.2 Web Server, the HTML and CSS code is separated into a header file I called homepage.h but this can be anything. Within the file there is the String of the HTML code followed by this:





It looks confusing but is a vital piece of code that needs to be in between all the HTML and CSS code.

First the ‘F’ means that the contents within will be placed in the Flash Memory of the ESP32 and not ROM which is the Read Only Memory and that is where the program is stored until it is executed. Then the R”====()=====) means that this is a raw string literal. A raw string literal is a place where you can write code like HTML or CSS that the ESP32 will not try to execute and just send it as is. The many equals signs is a delimiter to prevent any problems with the HTML or the CSS code in the string.

### Responsive Design

In my HTML and CSS code I have implemented Flex Box that is a tool built into CSS that allows the browser window to adjust its size according to the available space given to it.

It is possible by including this line in the <head> section of your HTML code

<meta name="viewport" content="width=device-width, initial-scale=1, viewport-fit=cover">

This code allows the Web Server to always know what the current screen dimension is.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 8.4 Web Server Full Screen

Aside from the HTML code, some CSS is still required. The HTML code only provides the required data for the Flex Box feature to work. The code that does the changes is display: flex; I have this code everywhere so that everything gets adjusted when needed.

Figures 8.4 and 8.5 are screenshots of my Web Server. 8.4 is full screen and displays all the content while 8.5 is where the screen is shrunk to about a phone size screen and their Flex Box automatically puts 2 scroll bars to move the webpage up and down and left and right.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 8.5 Web Server ¼ Screen

## Web Design for Accessibility

As my project focuses on showing lots of information to the user without any input, I added code to my Web Server that allows screen readers to read the screen.

I used Ctrl + F to test if the screen reader function works as seen in Figure 8.6. This feature is on about any website that has text to make it easy for finding a word or phrase in a long PDF document or an online newspaper.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 8.6 Web Server Screen Reader Example

To make screen readers work with my Web Server, I have included this line at the very top of my HTML code: <html lang=”en”>. This line tells any screen reader what language to use.

# Problem Solving

In my project, I have encountered many problems that I was able to find a fix for it. I have broken down the list from simple problems with easy fix to problems that I had to work hard to get what I wanted.

1. Where was I going to store my project through out the year. I wanted to take the project home to work on it their as well as in the ATU Labs. So, I was looking for a flat box that would house all my components. I eventually found the best solution, a box from assorted components I had empty lying around. I now had a solid box to keep all my components safe as the ESP32 is expensive and the AHT20 sensor is hard to find.
2. For the project, I used my laptops hotspot for the ESP32 to connect to, and it connected all the time, so I could view my Web Server. But in the last few months, my laptop has been losing connectivity with the college Wi-Fi constantly. It is annoying as ThingSpeak would disconnect and my Web Server will not show live information. My hotspot also changed the IP Address of the Web Server every time it got connected. My phones hotspot does not do that as it remembers the device and uses the same one. It was not a big problem, probably just some limitation of the way the hotspot is made on the laptop.
3. When uploading code to the ESP32, it sometimes works first time but a lot of the time I had to attempt the upload multiple times. It happens on my laptop and rarely on the Lab computers. I found that a button labelled ‘boot’ on the ESP32 must be pressed when the code it preparing to upload. Only after that, the code uploaded. I could not pinpoint the specific circumstances when it would fail or work without any problems.
4. The most challenging problem I faced was with my Servo Motor that I have removed from my project. It was frustrating to work with as it wouldn’t do what the code indicated, overall, its movement was erratic and unpredictable. After some trouble with it, I was shown that a 220 Ω resistor should be in between the ESP32 pin and the servo Signal pin. That reduced the wild movement I was expecting. As the servo motor has caused me so much hassle that I decided to just remove it from the project. I am still happy with the number of components I have but may revisit getting the servo working fully on the ESP32.

# Impact of Project on Sustainability

I have aimed to make this project for an audience that wants to see information. Lots of information, that does not require any input from them to see it.

I feel that my project has positively impacted the United Nations, Sustainable Development Goals.

**SDG 7: Affordable and Clean Energy** **[13]**

This goal describes the importance of reducing the total energy consumption per household. Electricity is scarce in some parts of the world. Target 7.1 and 7.2 describes how society needs stronger energy conversion measures as there is insufficient progress in the heat and transport sectors.

In my project I will have an LED light in my room that with a PIR motion sensor will monitor the room for human presence and if it detects nothing, it will turn off the light to reduce electricity consumption. I have a Lux meter measuring if it is sunny outside so that it turns off the light to reduce electricity consumption of a home. This feature is implemented in every school hallway or office space.

Most of my components will be given back to the college for redistribution for next year as new students will be starting their IoT projects and may have similar ideas to what I have done, reusing the AHT20 or the light sensor. Components like the wires, resistors and breadboards will either get recycled or I will use them for my own projects.

# Conclusion

This is the end of the line, I have covered every aspect of my project that I could think of. This was a very fun activity that has though me many important skills, not only for any future big projects like the one in 4th Year but also for when I will be in work placement.

I have learned that tasks take way longer than originally planned and to make room for extensions as some of my components needed some extra research on top of the original idea research to find how to code for the AHT20 sensor for example.

I see myself expanding this project towards higher standards. I would be using platforms like the Raspberry Pi and MQTT to assist the ESP32 in getting more advanced features and looks on the Web Server side. I would run the Web Server in a way that can be accessed by anyone on the Internet.

# References

|  |  |
| --- | --- |
| [1] | Espressif, "ESP32\_Datasheet," 2024. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32\_datasheet\_en.pdf. [Accessed 7 October 2024]. |
| [2] | Arduino, "Arduino IDE 2.3.6," [Software]. Avaiable: https://arduino.cc/en/software. [Accessed 7 October 2024]. |
| [3] | Mathsworks, "IoT Analytics - Thingspeak Internet of Things," [Online]. Available: https://thingspeak.com/. [Accessed 11 November 2024]. |
| [4] | CodeMagic, "www.wokwi.com," [Online]. [Accessed 4 November 2024]. |
| [5] | JGraph, "www.draw.io," [Online]. [Accessed 10 March 2025]. |
| [6] | Adarfuit, "DHT11\_Datasheet," [Online]. Available: https://cdn-learn.adafruit.com/downloads/pdf/dht.pdf. [Accessed 21 October 2024]. |
| [7] | Adafruit, "AHT20\_Datasheet," [Online]. Avaiblable: https://cdn-learn.adafruit.com/downloads/pdf/adafruit-aht20.pdf. [Accessed 21 October 2024]. |
| [8] | DFRobot, "Light Sensor\_Datasheet," [Online]. Available: https://wiki.dfrobot.com/Ambient\_Light\_Sensor\_0\_200klx\_SKU\_SEN0390. [Accessed 23 October]. |
| [9] | Kevin Harrington and John K. Bennett, "ESP32Servo Library," [Online]. Available: https://docs.ardunio.cc/libraries/esp32servo. [Accessed: 4 November 2024]. |
| [10] | Arduino, "Stepper Motor Library," [Online]. Available: https://www.arduino.cc/en/Reference/Stepper. [Accessed 6 November 2024]. |
| [11] | Microsoft, "Visual Studio Code 1.89.0," [Software]. Available: https://code.visualstudio.com. [Accessed 28 October 2024]. |

[12] Espressif, "ESP32 Arduino Core: WiFi and WiFiClient Libraries," [Online]. Available: https://github.com/espressif/arduino-esp32. [Accessed 17 February 2025].

[13] United Nations, "Sustainable Development Goal 7: Affordable and Clean Energy," [Online] Available: https://sdgs.un.org/goals/goal7. [Accessed 5 May 2025].

# Appendix 1: Code

All my code can be accessed through my GitHub repository.

My GitHub is <https://github.com/CircuitScupltor/IoT-Project>

For ease of access, here is a QR Code that you can scan.

A qr code with a logo

AI-generated content may be incorrect.

# Appendix 2: Bill of Materials

A screenshot of a computer

AI-generated content may be incorrect.