Text, logo

Description automatically generated

Internet of Things Project

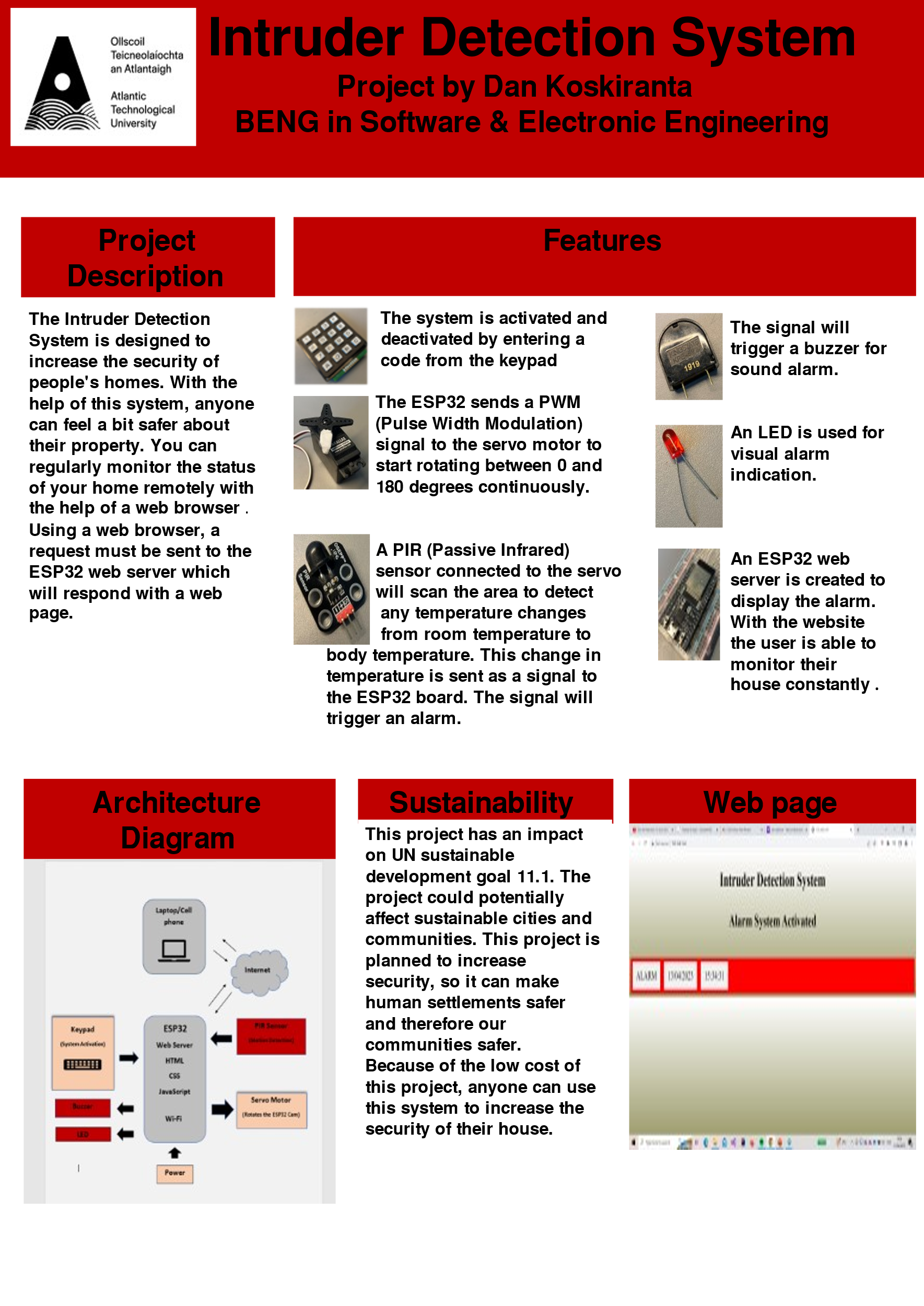
Intruder Detection System

Dan Koskiranta

Bachelor of Software & Electronic Engineering

Atlantic Technological University

2022/2023



Declaration

This project is presented in partial fulfilment of the requirements for the degree of Bachelor of Engineering in Software & Electronic at the Atlantic Technological 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.

\_\_\_Dan Koskiranta\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Acknowledgements

I would like to thank my mentors Natasha Rohan, Mairtin O’Conghaile, Brian O’Shea and Michelle Lynch for their guidance throughout this project. Their advice and expertise helped me a lot to complete this assignment the best way possible. I learned many new things and they gave me a lot of useful tips for this project. I sincerely appreciate their support for me.

Summary

The Intruder Detection System is designed to increase the security of your property whether it’s your home, business or whatever. With the help of this system anyone can feel a bit safer about their belongings.

The project will have reused components for all the main functions. A website is created to allow the user to monitor the status of their home. The project requirements are that it be completed no later than May 2023 and for under €200.

You can activate and deactivate the system with a keypad. The ESP32 microcontroller will send a PWM (Pulse Width Modulation) signal to the servo motor to start to rotate between 0 and 180 degrees continuously. The servo will rotate an ESP32 CAM which is used for video surveillance. A PIR (Passive Infrared) sensor will scan the perimeter for any temperature changes from room temperature to body temperature. Whenever a warm body passes by the sensor’s parameter, this is sent as a signal to the ESP32. This signal will trigger a buzzer, an LED, and an alert on a website.

This project will have a positive impact on UN sustainable development goal 11.1. More specifically it will affect sustainable cities and communities. The project is designed to increase security in our neighbourhoods, so as a result, human settlements will be safer and communities as well.

Contents

[1 Introduction 7](#_Toc134538890)

[1.1 Project Goals 7](#_Toc134538891)

[1.2 Project Motivation 7](#_Toc134538892)

[1.3 Project Benefits 7](#_Toc134538893)

[1.4 Project Overview 7](#_Toc134538894)

[2 Project Architecture 8](#_Toc134538895)

[3 Hardware 10](#_Toc134538896)

[3.1 PIR (Passive Infrared) Sensor 10](#_Toc134538897)

[3.1.1 PIR sensor code 11](#_Toc134538898)

[3.2 Servo Motor 12](#_Toc134538899)

[3.3 Keypad 14](#_Toc134538900)

[3.3.1 Keypad Code 15](#_Toc134538901)

[3.4 ESP32 16](#_Toc134538902)

[3.5 ESP32 CAM 20](#_Toc134538903)

[4 Software 21](#_Toc134538904)

[4.1 Arduino IDE 21](#_Toc134538905)

[4.2 HTML (Hyper Text Markup Language) 21](#_Toc134538906)

[4.3 CSS (Cascading Style Sheet) 23](#_Toc134538907)

[4.4 JavaScript 25](#_Toc134538908)

[5 Problem Solving 26](#_Toc134538909)

[6 Conclusion 27](#_Toc134538910)

[7 References 28](#_Toc134538911)

[8 Appendices 29](#_Toc134538912)

# 1 Introduction

## 1.1 Project Goals

The Intruder Detection System will detect unauthorized access into a secured area. The cost of the project is low so anyone can use this system in various settings such as homes and businesses.

## 1.2 Project Motivation

The reason why I chose to do this project is because I always feel a bit concerned about my belongings. By creating this surveillance system, I can feel more relaxed.

## 1.3 Project Benefits

One of the main benefits is that some extra security is provided to protect against theft and vandalism. The system can also act as a deterrent since a visible security system can discourage the intruders from entering the property. This project can enhance the security of any property and the low cost makes it a good investment for anyone concerned about security threats.

## 1.4 Project Overview

The report will start with a project architecture which will briefly explain the development platform and development tools used in the project. Then there is the main body where the hardware and software are explained. This section will also address the challenges I had during the project. I will finish the report with a conclusion of the key findings and recommendations.

# 2 Project Architecture

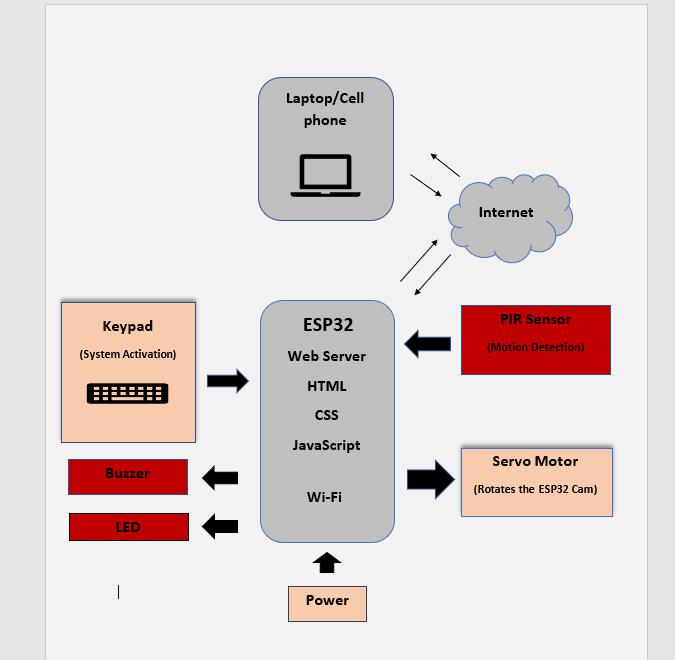
The development platform that was used in this project was the ESP32. The ESP32 is a microcontroller which has more features than the Arduino and it’s faster than the Arduino Uno. It’s a 32-bit device as opposed to Arduino Uno which is an 8-bit device. It has a multitude of versatile I/O ports, like digital and analog ports and capacitive touch inputs.

The version I used was ESP32-WROOM-32D. Here are the main features: It’s a dual core so it has two Xtensa 32-bit LX6 microprocessors. It has a clock frequency up to 240 MHz and multiple power modes. It also includes integrated WiFi and Bluetooth. The programming of the ESP32 is done using the Arduino IDE programming language.

The Arduino IDE stands for Integrated Development Environment. It’s a software that you can use to write, compile, and upload code in almost all Arduino modules/boards. You will write the main code on the Arduino IDE, and this will be uploaded on the controller of the board. The IDE has two basic parts: Editor and compiler where the editor is used to write the code and the compiler is used to compile and upload the code into a specific Arduino module. Once you’ve installed the Arduino IDE, you need to add the ESP32 board to it. Open the Arduino IDE, go to File > Preferences and paste the following URL in the Additional Boards Manager URLs field: <https://dl.espressif.com/dl/package_esp32_index.json>. Then, go to Tools > Board > Boards Manager, search for “esp32” and install the “ESP32” board. After installing the ESP32 board, you need to select it from the list of available boards: Tools > Board and select the specific board you have. Finally, you need to check from the device manager to which com port your ESP32 is connected to and then from Arduino IDE, select from under tools the port to which your board is connected to. Both C and C++ are supported in Arduino IDE.

The website was created by using the HTML, CSS, and JavaScript programming languages. HTML (Hyper Text Markup Language) was used to structure the web page and its content. I used the CSS (Cascading Style Sheet) for the website design. JavaScript was used to make specific features functional on the web page.

**Architecture Diagram**



# 3 Hardware

## 3.1 PIR (Passive Infrared) Sensor

This project used an OSEPP PIR-02 sensor.

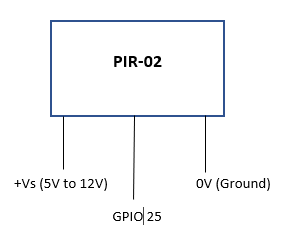
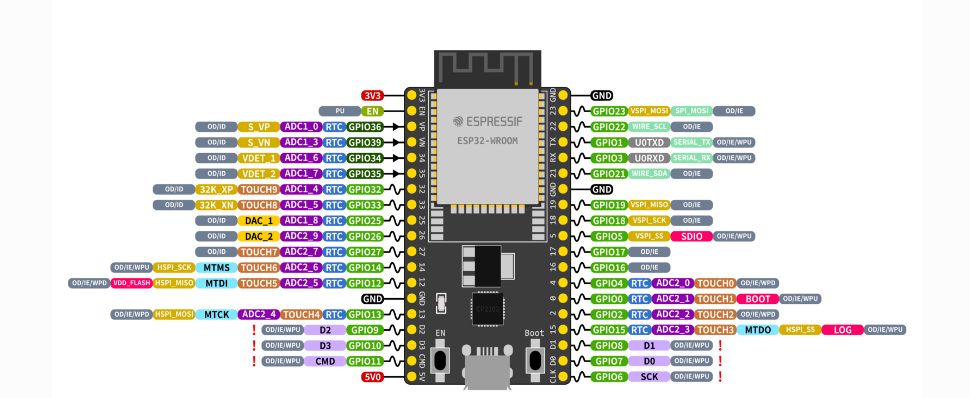
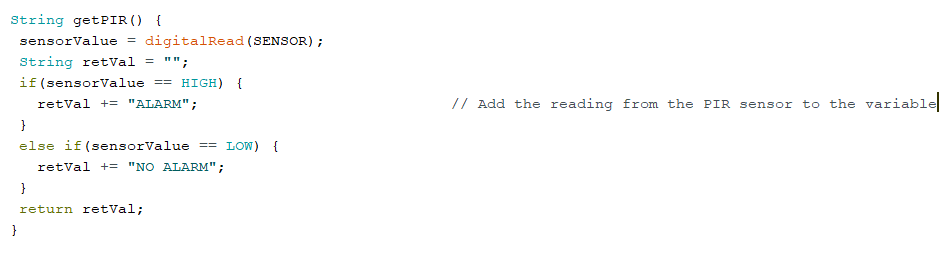


Figure 3-1 PIR sensor circuit

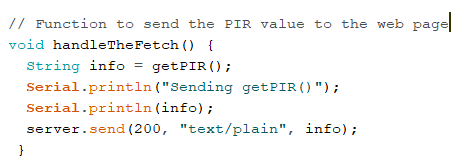
The sensor was used to detect motion. The PIR sensor contains pyroelectric elements that can measure infrared radiation emitted from objects in its field of view. Pyroelectric materials generate a voltage in response to temperature changes. Whenever the sensor detects temperature change in its field of view, that change in temperature translates into sensing movement and generates an electrical charge [1]. This is sent as a signal from the PIR’s output to the ESP32 pin 9 (GPIO 25) to trigger an alarm.

[2]

### 3.1.1 PIR sensor code



Functions are used when coding the PIR sensor. The reading from the sensor is added to a variable and the value is sent to the ESP32 web server which must display this value to the user. Using the Arduino function digitalRead, I’m able to check the status of the PIR sensor. I can store a message to a variable by reading the sensor, and then return the message to a specific source.



The getPIR function’s job is to read the status of the sensor pin. Then it must return the message that is generated to a different function called handleTheFetch which will assign a variable to the message and send it to the ESP32 web browser.

## 3.2 Servo Motor

The type of servo that was used was a Parallax continuous rotation servo motor [3]. The connection to ESP32 is established using pin number 13 (GPIO 12) on ESP32.

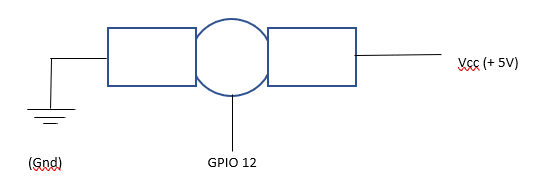


Figure 3-2 Servo Motor

The continuous servo has a rotation range of 360 degrees, but it was used to cover 180 degrees in this project. With help of a servo, you can rotate a camera for video surveillance.

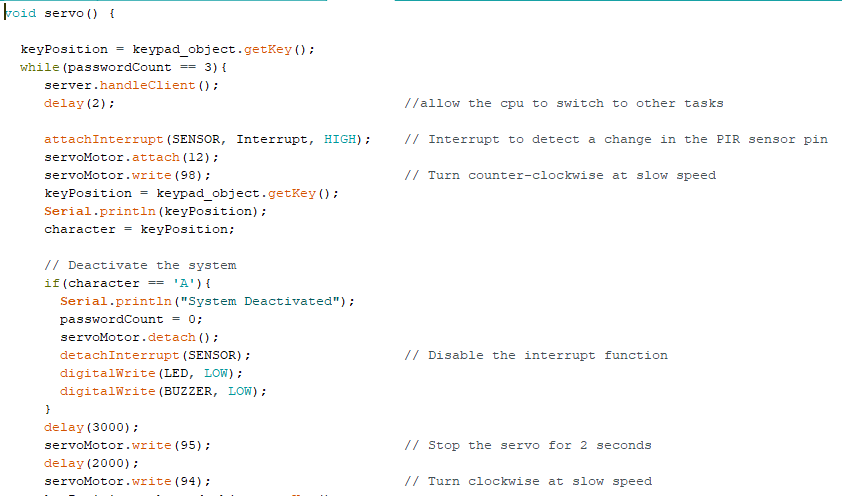
The servo motor is controlled by sending a PWM (Pulse Width Modulation) signal to the motor. The speed is determined by the length of the pulse with longer pulses resulting in faster speeds. When the pulse is longer there is more current flowing through the motor coil.

Most continuous servo motors have a neutral position where the motor is not rotating. If you send a pulse with a length equal to the neutral position, the motor will remain stationary. To control the direction of a continuous servo, you need to send a pulse that is either longer or shorter than the neutral position pulse.

The direction is controlled with pulses varying from 1 to 2 milliseconds that are sent every 20 milliseconds. So, the period is 20 milliseconds, and the frequency is 50 Hz. = 50[3]**.**

One millisecond pulse will make the servo to rotate clockwise at full speed and two milliseconds counterclockwise. The neutral position is 1.5 milliseconds where the servo is stationary.

**Servo Motor code**



In this project there is a function **servo()** for the servo motor that is called when the system is activated. The ESP32Servo.h library allowed me to use the attach and detach methods for servo activation and deactivation. The **write()** method controls the direction and speed of the servo. The **servomotor.write(0)** represents the 1 ms pulse width and moves the servo clockwise at full speed. The .write(180) represents 2 ms and moves counterclockwise at full speed. You need to write 95 to stop the servo.

## 3.3 Keypad

The project included a keypad to activate and deactivate the system. The specific keypad model that was used was the ECO.16250.06 | Keypad ECO 4x4| EOZ [4].

X1 X2 X3 X4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Y1 | 1 | 2 | 3 | F |
| Y2 | 4 | 5 | 6 | E |
| Y3 | 7 | 8 | 9 | D |
| Y4 | A | 0 | B | C |

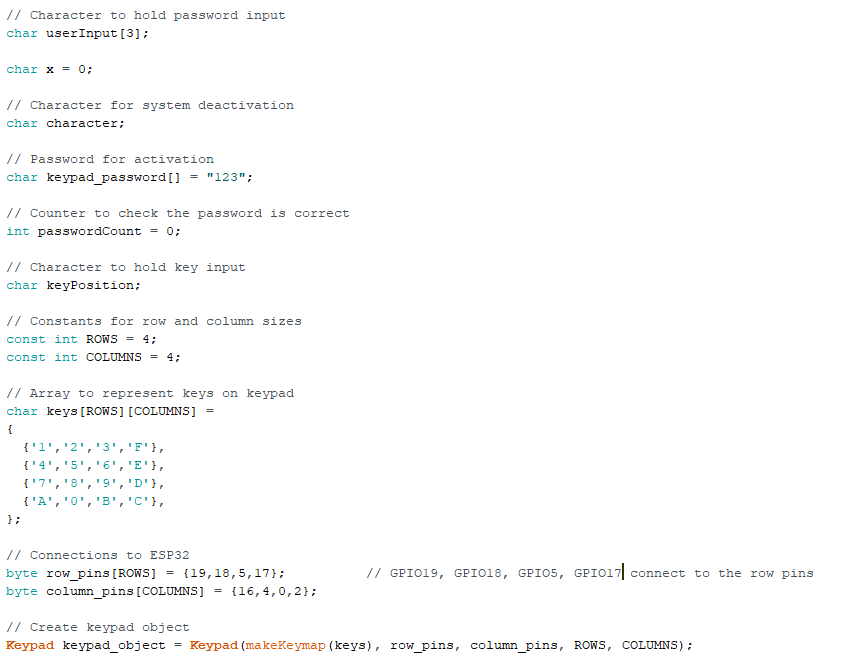
Matrix XY contacts

**Y4, Y3, Y2, Y1** represent pins to the rows. **X4, X3, X2, X1** represent pins to the columns.

Y4 Y3 Y2 Y1 X4 X3 X2 X1

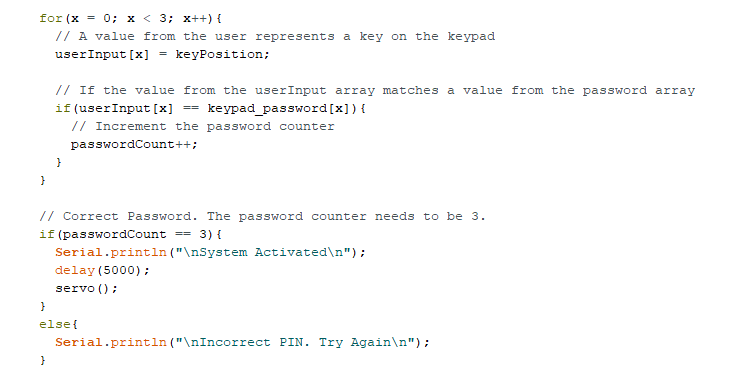
Each key on the keypad is a unique input. Whenever you press a key, it completes a circuit on the keypad, and it will send a signal to the connected device which in this project is the ESP32. When the keys are in their natural state, they don’t touch the conductive contacts beneath them. When the key is pressed, the conductive contact of the key will touch the conductive contact beneath it, so it will complete the circuit.

### 3.3.1 Keypad Code



The Keypad library “Keypad.h” for ESP32 was used in this project which allowed me to use the library functions. There was a keypad object created which used the pins 19, 18, 5 and 17 as row inputs and 16, 4, 0 and 2 as column pins. I used an array to store all the keypad characters. With the keypad library I was able to use the constructor Keypad(makeKeymap(userKeymap)) to create the keypad object.

I used an array for the user inputs which holds 3 characters. The reason why the value for the userInput array is 3 is because the password array contains 3 characters. The two arrays need to match because I need to check that the value in the userInput array matches the corresponding value in the password array. When the value matches, the password counter will increment by 1. The password counter needs to add up to 3 for the password to be correct.



## 3.4 ESP32

The main development board for the project was the **ESPRESSIF ESP32-WROOM-32D** microcontroller. The main benefits of an ESP32 are that the microcontroller is a low-cost and low-power consuming device. It has features like deep sleep mode to save power. It’s a powerful microcontroller with built in WiFi and Bluetooth that allow it to communicate wirelessly with other devices. It also has a wide range of I/O (Input/Output) peripherals which make it ideal for many projects.

The ESP32 is a dual core, so it has two CPU (Central Processing Unit) cores which can be controlled individually. You can adjust the CPU clock frequency from 80 MHz to 240 MHz. It also has a slower co-processor that can be used to perform smaller tasks while the dual core CPU is in sleep mode.

The microcontroller has a lot of memory which means that it’s suitable for tasks like connecting with cameras, recognizing speech, and streaming data from the internet.

**ESP32 Memories**:

* **ROM** (Read-only memory): 448 KB: This section cannot be reprogrammed. The Bluetooth, WiFi, core functions and booting are handled from the ROM.
* **SRAM** (Static Random Access Memory): 520 KB: This is the memory where the data and instructions are saved.
* **RTC SRAM** (Real-time communications): 16 KB: The co-processor uses this memory while the device is in sleep mode.
* **Efuse**: 1 Kbit: The system uses 256 bits of this memory, and the remaining 768 bits are reserved for other applications.
* **Flash embedded**: The application code is stored in this memory. The amount of memory depends on which chip is used.

**WiFi:**

You can connect the ESP32 to a WiFi network or you can create a WiFi wireless network (access point mode) for it. When it works in station mode, the ESP connects to an available WiFi network [5].

In access point mode, the chip will create a WiFi field around it so other devices can connect to it. The ESP32 doesn’t have internet access by itself. When in access point mode you can only display few web pages that are programmed in the ESP32 memory.

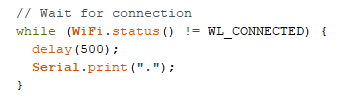
The 3rd option is called Combined AP-STA mode. In this mode the ESP32 connects to an existing WiFi network and at the same time it’s also creating its own WiFi field so other devices can connect to it.

For this project, the ESP was set to work in station mode. The **WiFi.h** library for ESP32 provides functions that allow you to connect to a WiFi network and to communicate over the network.

**ESP32 set as a WiFi station**



**Wait for the board to connect to WiFi**

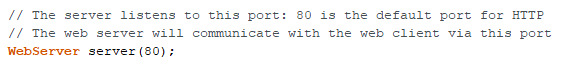


**Return the IP address of ESP32**

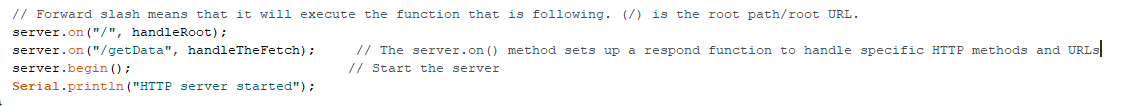


The **WebServer.h** library was used in this project to set the ESP32 as a web server and to respond to HTTP requests.

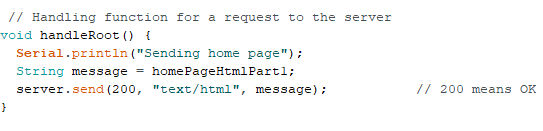
**Create an instance of WebServer class**



**Start the server**



**Send an HTTP response to the web client**



**ESP32 Pins**:

The ESP32 has a total of 38 pins and most of them have multiple functions.

* GPIOs 34 – 39 are input only pins.
* GPIO 6 to 11 are connected to the integrated SPI flash on the ESP-WROOM-32 chip and should not be used [6].
* The ESP32 has 10 internal capacitive touch sensors [6]. These GPIOs can sense variations in anything that holds an electrical charge [6]. The internal touch sensors are connected to the following GPIO pins:
* GPIO 4
* GPIO 0
* GPIO 2
* GPIO 15
* GPIO 13
* GPIO 12
* GPIO 14
* GPIO 27
* GPIO 33
* GPIO 32
* There are 18 ADC (Analog-to-Digital Converter) input channels. The channels have a 12-bit resolution. So, you can get readings that range from 0 to 4095, 2^12 = 4095. The reading 0 corresponds to 0 V and 4095 corresponds to 3.3 V [6].
* ESP32 has 2 DAC (Digital-to-Analog Converter) channels. These can change digital signals into analog voltage signal outputs [6].
* There are 16 independent channels that can generate PWM signals.
* The chip has two I2C channels. These can be used for communication with devices such as sensors, LCD displays and other peripherals.
* There are three UART (Universal Asynchronous Receivers and Transmitter) interfaces. These are used for serial communication. UART’s main purpose is to transmit and receive serial data [7].

Graphical user interface

Description automatically generated

Timeline

Description automatically generated

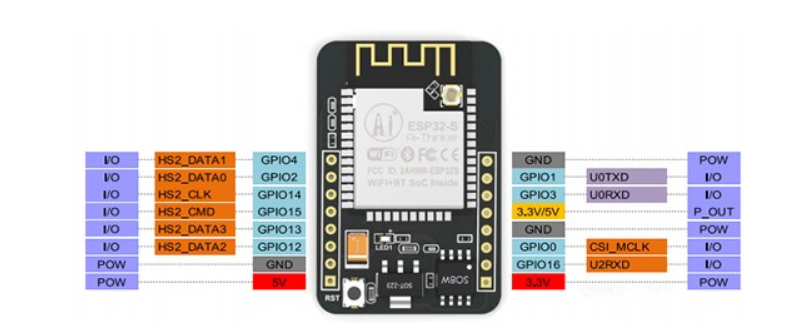
[2]

## 3.5 ESP32 CAM

The ESP32 CAM is a small camera module with an ESP32-S chip. It includes an OV2640 2-megapixel camera. It has a slot for the microSD card to store images taken with the camera. The ESP32 CAM does not have a USB (Universal Serial Bus) connector, so you need an FTDI programmer to upload code through the U0R and U0T pins [8].

The board has components on both sides of the module so it’s difficult to breadboard. On the bottom side is the ESP32-S chip: 2 high-performance 32-bit LX6 CPUs. The reset button and voltage regulator chip are also located on the bottom side. On the top side, we have TF Card Holder: microSD card holder, FPC connector and Flash Light.

**ESP32-CAM Pinout**

**[8]

The ESP32 CAM module has 16 pins in total. It includes 2 power pins: 5V and 3.3V, it has 3 ground pins and 10 GPIO pins.

You can assign a variety of peripheral duties to the GPIO pins, such as UART, SPI, ADC, and touch. GPIO 1 and 3 are the serial pins which are used to upload code to your board. Also, the ESP32 CAM needs to be in flashing mode so you can upload the code. To set the ESP32 in flashing mode, you need to connect GPIO 0 to GND [8].

The connection between the ESP32 and ESP32 CAM can be done in the following way:

|  |  |
| --- | --- |
| **ESP32 CAM** | **ESP32** |
| 5V | 5V |
| GND | GND |
| U0R (GPIO 3) | RX (GPIO 3) |
| U0T (GPIO 1) | TX (GPIO 1) |
| GPIO 0 | GND |

# 4 Software

## 4.1 Arduino IDE

The Arduino IDE was used to program the ESP32. The code was written in C++. The Arduino IDE is an open-source software that provides an interface for writing, compiling, and uploading code to Arduino boards. The Integrated Development Environment provides a text editor to write the code and a compiler to compile the code. An Arduino program is called a sketch.

Here are some key features of the Arduino IDE:

* Library Manager: The Arduino Integrated Development Environment includes a library manager that allows the user to search and install thousands of libraries. With the help of these libraries, the user can add some functionality to projects, such as adding motor functions, connecting sensors and display readings from sensors and to use a WiFi module.

To include a library to a project, you need to include it at top of the sketch.

* Serial Monitor: The Serial Monitor tool allows the user to view the data that is streamed from the board. You need to use the **Serial.print()** command to view the data. The Serial Monitor can also be used as a debugging tool. In order to communicate with the ESP32, you need to set the baud rate, which is done using the **Serial.begin(115200);** command. The 115200 represents the baud rate, which is the maximum bits per seconds that can be transferred[9].
* Board Manager: This tool allows you to install different cores to your computer. A core is written and designed for different microcontrollers[10]. You need to add the ESP32 boards from Arduino IDE Board Manager before you can use the Arduino IDE on ESP32.

When uploading the code into the ESP32, the correct port needs to be chosen. When you have the programming environment ready, you can start to experiment with ESP32. You can test the WiFi library functions, for example, scan the nearby WiFi networks within its WiFi range. This can be done using the **WiFi.scanNetworks()** function.

## 4.2 HTML (Hyper Text Markup Language)

The web page was created using the HTML language. HTML allows you to describe the content of the web page like headings and paragraphs. The HTML code for this project was written to the IDE sketch, which made it easier to display the page from the ESP32 web server. The code was created inside a raw string literal in order to include characters like quotation marks and backslashes which normally act like delimiters and escape sequence starts.

**HTML Code**

A screenshot of a computer code

Description automatically generated with low confidence

**<!DOCTYPE html>** at the top declares the HTML document. **<html>** tag represents the entire HTML document and contains all other HTML elements, including **<head>** and **<body>** sections. The **<head>** tag provides the metadata about the document such as the title of the page, CSS design section for the web page or some other information that is not displayed on the page.

The ESP32 web server can send the HTML page to the web client using the

**server.send(200, “text/html”, variable)** function.

## 4.3 CSS (Cascading Style Sheet)

The CSS was used for the design of the web page. How the HTML elements are displayed is done with CSS section.

**CSS code**

A screenshot of a computer code

Description automatically generated with medium confidence

The code:

**body {**

**background: linear-gradient (to bottom, #ffffff 0%, #999966 100%)**

will set the background of the web page to a mixture of two colors: white and grey. Colour code for white is #ffffff. Grey is the more dominant colour.

**h1{**

**padding: 2vh;**

**text-align: center;**

**font-size: 50px;**

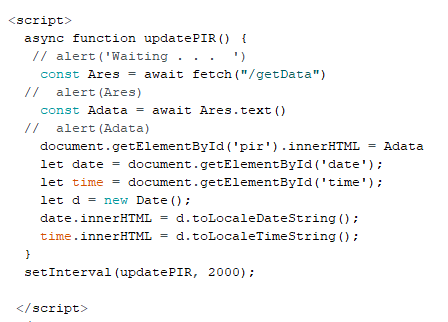
**}**

This line of code sets a padding of 2% of the viewport height on all four sides of the h1 element (Main heading). It centres the h1 heading horizontally and sets the font size to 50 px. The font size is 0.52 inches. 1px = 1/96 of an inch. = 0.52.

## 4.4 JavaScript

The JavaScript language was used to code the behaviour of the web site. The JavaScript code in this project includes one async function which is used to return a promise

**JavaScript code**



In this function, the fetch method is fetching the data from the **getData** resource from the ESP32 server. The keyword await only works inside async functions. The await makes JavaScript to wait until the result has been retrieved. While it waits for the result, JavaScript can execute other scripts in the meantime. The result obtained from the fetch will be assigned to the Ares variable.

The **text()** method returns the text content of the selected elements. The **getElementById()** method returns an element with a specified value [11]. The **innerHTML** property sets or returns the HTML content of the element [11]. In this project it’s the message that is generated when the PIR sensor is high (“Alarm”). The getElementById method is also used to get current date and time.

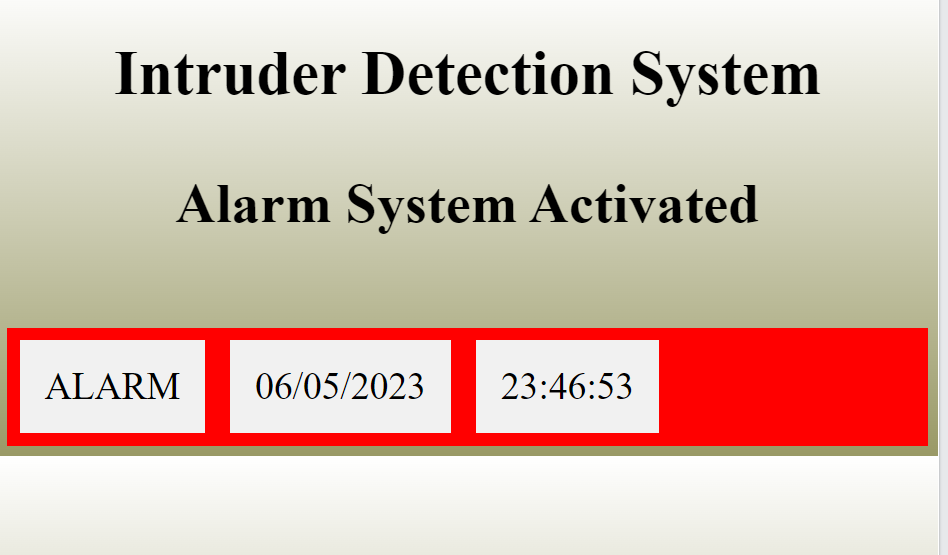
The **setInterval()** method will call the updatePIR function every 2 seconds, 2000ms = 2s.

# 5 Problem Solving

The big challenge for this project was that there were a lot of new things to learn and then trying to implement these skills. To learn how the ESP32 works and all the functions it has. To create a web page using HTML, CSS and JavaScript was new but interesting. Then trying to display readings from sensors on the web page was the most challenging part.

One example was when I was trying to display the warning message from the PIR to the web page. With the help from my lecturers and by going through the PowerPoint examples, I was able to figure out how to build the web page into the IDE sketch. I spent like two weeks on how to get the reading from the sensor to the web site and when I tested it nothing was displayed. I went through the code line by line, and I saw that I forgot to configure the pin as an input.

Next problem was to display the warning message inside a flexbox. I spent approximately three weeks on this problem. I kept testing different things and nothing was working. With some guidance and doing research online, I saw that there was one unnecessary line of code **console.log** inside the JavaScript async function that was causing the issue. After removing this, the readings were displayed in the correct sections.



The integration of the subsystems was the final challenge. The way it was achieved was by using functions. For example, using a function for the servo motor operation.

The keypad code was in the loop function where the user had to enter the correct pin to activate the system. Once it was activated, a servo() function was called. Inside this function, besides the servo operation, there was an attachInterrupt method for the PIR sensor pin. Whenever a specific change was detected in the pin, an interrupt function was called immediately to set the buzzer and LED high. With interrupts, it was possible to handle the event immediately without blocking the main program’s execution.

The server.handleClient() method was placed inside the servo function to handle incoming HTTP requests. The method needed to be called constantly to handle the client requests in real-time and to provide a responsive web server.

# 6 Conclusion

To conclude, the objective was to develop a surveillance monitoring system with the help of a microcontroller, sensors, camera module and a web site. Overall, I got about 60% done what I wanted. The web page is displaying the alarm and most of the components are working. Unfortunately, I was unable to integrate ESP32 CAM module into the project. I ordered the part too late, and I ran out of time to get it working.

The research of the project was done on ESP32 function, how to configure the ESP32 web server and how to display the sensor readings on the website. There was research done on the servo motor, especially how the continuous servo works. Also, the PIR sensor was a vital part of this project which required research.

The testing phase included testing the application, the various methods used in the application and the different components to ensure that everything meets the objectives of the project.

The features of the application such as data input and output were tested, testing code for the component functionality and integration testing to see how the components worked together.

There were issues with delays in my project. For example, it took some time for the buzzer and LED to go high when the PIR sensor went high. I fixed this by using an external interrupt. After component integration, there was a delay in displaying the alarm message on the web page. I tried to resolve the issue by calling the **server.handleClient()** method more frequently.

The project delivered the servo function, PIR function, keypad, buzzer, and LED. All these were demonstrated along with a web page. ESP32 CAM was undelivered, and it was one of the most important features for the project. To make the project better, the camera needs to be added and web accessibility needs to be improved.

# 7 References

|  |  |
| --- | --- |
| [1] | “PIR Motion Sensor,” Adafruit Industries, [Online]. Available: https://cdn-learn.adafruit.com/downloads/pdf/pir-passive-infrared-proximity-motion-sensor.pdf. [Accessed 6 May 2023 May 2023]. |
| [2] | “ESP32-DevKitC V4 Getting Started Guide,” Espressif Systems (Shanghai) Co., Ltd, 2016. [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/esp32/get-started-devkitc.html#get-started-esp32-devkitc-board-front. [Accessed 4 May 2023 May 2023]. |
| [3] | “Parallax Continuous Rotation Servo (#900-00008),” Parallax Inc, 24 October 2011. [Online]. Available: https://docs.rs-online.com/870b/0900766b8123f8a1.pdf. [Accessed 5 April 2023 April 2023]. |
| [4] | “Keypads ECO,” eao, 10 2010. [Online]. Available: https://docs.rs-online.com/e17e/0900766b800338ed.pdf. [Accessed 20 April 2023 April 2023]. |
| [5] | “WiFi on ESP32,” Tutorials Point India Private Limited, 4th Floor, Incor9 Building, Plot No: 283/A, Kavuri Hills, Madhapur, Hyderabad, Telangana, INDIA-500081, [Online]. Available: https://www.tutorialspoint.com/esp32\_for\_iot/esp32\_for\_iot\_on\_wifi.htm. [Accessed 18 April 2023 April 2023]. |
| [6] | “Random Nerd Tutorials,” RandomNerdTutorials.com, 2013. [Online]. Available: https://randomnerdtutorials.com/esp32-pinout-reference-gpios/. [Accessed 28 April 2023]. |
| [7] | S. C. |. D. E. |. 79, “BASICS OF UART COMMUNICATION,” Circuit Basics, [Online]. Available: https://www.circuitbasics.com/basics-uart-communication/. [Accessed 26 April 2023]. |
| [8] | “ESP32-CAM Video Streaming and Face Recognition with Arduino IDE,” RandomNerdTutorials, 2013. [Online]. Available: https://randomnerdtutorials.com/esp32-cam-video-streaming-face-recognition-arduino-ide/. [Accessed 15 April 2023]. |
| [9] | “Using the Serial Monitor tool,” Arduino, 2023. [Online]. Available: https://docs.arduino.cc/software/ide-v2/tutorials/ide-v2-serial-monitor. [Accessed 5 May 2023 May 2023]. |
| [10] | “Using the Board Manager,” Arduino, 2023. [Online]. Available: https://docs.arduino.cc/software/ide-v2/tutorials/ide-v2-board-manager. [Accessed 5 May 2023 May 2023]. |
| [11] | “HTML DOM Document getElementById(),” W3Schools, 1999. [Online]. Available: https://www.w3schools.com/jsref/met\_document\_getelementbyid.asp. [Accessed 27 April 2023 April 2023]. |

# 8 Appendices

**BOM (Bill of Materials)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Item** | **Quantity** | **Manufacturer** | **Sourced From** | **Cost Euros** |
| ESP32 WROOM-32D | 1 | ESPRESSIF | ATU Galway | Estimated  13.50 |
| Parallax Continuous Rotation Servo Motor | 1 | Parallax Inc | ATU Galway | Estimated  14.00 |
| OSEPP PIR-02 Sensor | 1 | OSEPP Electronics | ATU Galway | Estimated  9.00 |
| KINGSTATE  KPE-522 Buzzer | 1 | KINGSTATE | ATU Galway | Estimated  0.50 |
| LED  (Light-Emitting Diode) | 1 | Uxcell | ATU Galway | Estimated  0.50 |
| 1kΩ Resistor | 1 | Honeywell | ATU Galway | Estimated  0.50 |
| ESP32-CAM-MB 2640 | 1 | ESPRESSIF | Amazon.de | 13.43 |

**Schematic**

