# SecureScape

## Smart Deployable Security System


Group 15 Authors:

| Colin Kirby | Philip Murano | Dylan Myers | Jaxon Topel |
|---|---|---|---|
| Computer Engineering | Electrical Engineering | Electrical Engineering | Computer Engineering |

**Review Committee/Mentors:**

- Dr. Chinwendu Enyioha – UCF ECE Assistant Professor
- Dr. Mike Borowczak – UCF ECE Associate Professor

# Table of Contents

# List of Tables

# List of Figures

# List of Equations

# Chapter 1 - Executive Summary

Our Senior Design project “SecureScape” a smart deployable security system, aims to address the rising needs for increased security in rural areas with little to no access to the internet. To achieve our goal of developing a full-scale security system, we have designed 2 main hardware components and have split our software components into three main sections: machine learning/software architecture, embedded code, and front-end development. In short, our demo will consist of 3 nodes (our first hardware component) that contain 2 ESP32-Cam MCUs, an active IR sensor, and a passive IR sensor. The passive IR sensor will constantly monitor the environment if motion is detected. Once motion is detected the ESP32 on the respective node will capture an image and send it over the network hosted by our router to the Gadget (our second hardware component). The Gadget is a device that must be within the area and is used to interact with our system, receive alerts for any potential threats in the area, and allows additional control over our system to adjust features for the customer's needs. Once the image is sent to the gadget, image processing is done locally on the ESP32-S3 board that lives within the gadget. The algorithm that processes the image will simply determine if there is a threat within the image by deciding whether a person is inside of the image using an object detection algorithm. If our algorithm determines there is a threat in the area, our system will then alert the user via an alarm that will live inside each node and notifications will be sent to the iOS application and to the gadget. We also are developing switches on the gadget and a mobile app to make this a user interactive system. We plan to provide some features such as turning off/on the alarm, turning off/on the lights, and specifically for the iOS app, requesting images from the nodes.

# Chapter 2 - Project Description

This chapter is meant to provide the reader with a basic understanding of what the project is, and answer both why and how we are doing it. In detail we shall present our planning process, with an emphasis on how every part of our system will interact with each other.

## 2.1 Project Motivation and Background:

We would like to first present some background knowledge on our group and our project before sharing the motivation behind what we are going to accomplish. For starters we are a group of 2 Computer Engineers and 2 Electrical Engineers. We formed our group in the Spring of 2024 and had originally decided to pursue a drone project. After further discussion, we concluded that we will not be able to gain experience in all the areas we wished to with that project. Tying this into the main motivation behind our current project, we also wanted to be able to get behind an idea that we were all excited about. The areas we were interested in working with for Jaxon Topel were image processing and successfully collecting sensor data to make software computations in a real time system. Colin Kirby wanted some experience with Embedded Systems integration along with front end development. Dylan Myers and Phillip Murano both wanted experience with hardware integration and the development and design of a PCB that will facilitate functionality to multiple peripherals. Our project requires a second PCB with different functionalities, allowing both students to work adjacently with their own board. With all these areas in mind to gain experience, we recognized the need for a portable security system that you could take with you to areas that you wish to feel more secure in. For example, our deployable security system will have various applications for hikers that can set up around their tent. Hunters can use our product to gain more situational awareness even in the dead of night. We hope to provide reliable security for anyone that wishes to remain safe in an unknown territory where you can be sure that you are secure even without phone service.

## 2.2 Goals and Objectives:

### 2.2.1 Basic Goals:

Our basic objective would be to successfully develop a deployable security system that will ensure no one can enter or walk around your area without having our alarm system triggered if it is a potential threat. The design requirements for our project consist of developing and integrating multiple hardware and software applications. We will develop 3 different “nodes” that will be retractable sticks in the ground with an IR sensor halfway up the pole, and two ESP32-CAM microcontrollers atop each node that will have a camera built into each of the boards for a total of two cameras per node. Each node will live inside a computer network that will interact with our handmade device we are calling gadget. Our gadget will be fully designed on the hardware side of things by Phillip and Dylan and will serve as the central point of communication in our network, receiving and sending information about our system to/from the nodes. All our devices will be powered from batteries that can be easily replaced or recharged.  Requirements for hardware will consist of ensuring all our hardware is compatible together, for further details see below. Our software applications will consist of programming the computer network to send information between nodes and our gadget. An image processing algorithm that will be tripped by the IR sensor that detects movement and recognizes whether the movement outside our secure area is a threat. When the IR sensor detects movement or the image processing algorithm classifies the movement as a threat, communication signals will be sent from the node to the gadget and the node that detected the threat will enter an alarmed state along with the gadget alerting the owner of the threat. The node(s) in an alarmed state will have the capability to sound an alarm, flash an LED, and have the cameras operate in an excited state.

### 2.2.2 Advanced Goals:

- Incorporate onboard memory for local data storage.
- Implement multiple alarm modes, allowing users to select between quiet and loud alerts.
- Installing a switch to the gadget that will allow the user to manually toggle a specific function within the node(s).
- Configure software to trigger an alarm if any security node is tampered with.
- Capture image button on app that will populate current images from the nodes to the app.

### 2.2.3 Stretch goals:

- Develop a fully integrated and comprehensive mobile application.
- Enable secure cloud storage for data backup and access.
- Introduce rechargeable peripherals for enhanced convenience.
- Design a control hub that also functions as a charging station for peripherals.
- Create software capable of distinguishing between human and animal threats, automatically adjusting the alarm's threat level accordingly.
- Develop a threat recognition system that can identify weapons, such as knives, and notify users of the corresponding threat level.
- Implement real-time updates of imagery on a dedicated website or app.
- Integrate battery level monitoring to ensure system reliability.

## 2.3 Project Functionalities:

Our project focuses on developing an advanced portable security system that seamlessly integrates motion detection with real-time alerts. The system is designed to provide continuous monitoring of secure areas, ensuring that users are always informed of any potential security breaches. Key features include a user-friendly interface for easy data integration, along with a versatile control gadget that allows comprehensive management of the entire security system. The features are detailed below:

### 2.3.1 Motion Detection:

Our system will utilize IR sensors to detect motion within the range and FOV, the detection will process in real time, sharing images to our processing algorithms, and potentially alerting users of threats.  Users will receive instant notifications through our mobile app and gadget whenever a threat is detected and will also hear the alarm from the node that detected the threat, informing the use of the location of the threat, giving an advantage to the user in the chance they will have to protect themselves.

### 2.3.2 Perimeter Security:

We will secure our system's perimeter by placing active IR sensors on each pole that will be set up to detect motion only along the perimeter of our secure area. This will be the ‘Achilles Heel’ of our system as the nodes will have to be set up in certain locations and lengths away from each other to ensure correct functionality. Users will receive notifications via our mobile app and our gadget we are building, along with all alarms being tripped sounding the highest level of threat.

### 2.3.3 Access and Control:

Users will be able to access the security system via mobile app or gadget. This will allow users to monitor the property and manage the system from anywhere. We will have to use secure cloud-based communication protocols and authentication methods to ensure safe remote access.

### 2.3.4 Response Integration:

This system can automatically alert pre-defined contacts in case of a specified level of threat detection. This will allow for a rapid response from the authorities and will enhance the security of our system.

### 2.3.5 Threat level alert system:

Multiple threat levels cases will be made to categorize threats into pre-defined areas. This feature will allow users to immediately and clearly identify what scenario they are in, enabling the user to take appropriate actions.

### 2.3.6 Feedback footage of detected motion/threats:

Available through the mobile app, users will be able to view in real time, recorded footage of detected motion/threats. This feature will allow users to respond appropriately to threats from distance or from a secure area.

In developing these features, we referenced input from customer reviews and conducted a marketing analysis of comparable products, such as the DIY home security system offered by Ooma [53], which emphasizes user-friendly installation and real-time threat alerts. Additionally, we considered insights from the best travel safety products, as highlighted by Travel and Leisure [54], to ensure our system offers portability, versatility, and effective perimeter protection.

## 2.4 Existing Products:

Existing products are a solar outdoor motion sensor alarm. The key differences between this product and ours will be: We will create a network of sensor alarm nodes providing a higher degree of security, and we will also have more advanced image processing algorithms providing a higher level of detail of security. Another key difference is that their product has a full 360-degree field of view while ours will have a 260-degree field of view from each node, this is because we are monitoring the area around the perimeter instead of monitoring from one point from within.

Figure 2.4: Image Sourced by HULPPRE [3]

## 2.5 Specifications:

| Specifications | Minimum | Maximum |
|---|---|---|
| Detection Accuracy | 75% | 100% |
| Detection ranges | 5 feet | 15-20 feet |
| Deployment time | 5 minutes | 10 minutes |
| Response time | > 5 sec | 10 sec |
| Scalability | 50-60% | 90- 95% |
| Installation time | Program Run Time | Program Run Time |
| Encryption level | 128 bits | 256 bits |
| Cloud latency | 10-50 ms | 50-100 ms |

Table 2.5: Specifications for Project Metrics

## 2.6 Hardware Planner:

| Work Items | Name | To be Acquired | Acquired | Investigating | Designing | Prototyping | Completed |
|---|---|---|---|---|---|---|---|
| Control Hub PCB | Phillip | P | | | D | | |
| Node PCB | Dylan | D | | | D | | |
| Control Hub power | Phillip | | P | | | | P |
| Control Hub housing | Dylan | D | | | | | D |
| Router | Dylan | | D | | | | D |
| Esp int. | Phillip | | P | | | | P |
| Esp cam int | Dylan | | D | | | | D |
| IR sensor (active) | Dylan | | D | | | | D |
| IR sensor (passive) | Dylan | | D | | | | D |
| LED/Alarm | Phillip | | P | | | | P |
| Node Housing | Phillip | P | | | | | P |
| Node Power | Phillip | | P | | | | P |
| System Integration | Dylan | D | | | | | D |
| Memory Integration | Dylan | D | | | | | D |
| Gadget Switches | Phillip | | P | | | | P |

Table 2.6: Hardware Planner Distributing Tasks Amongst Members

## 2.7 Hardware Diagram:

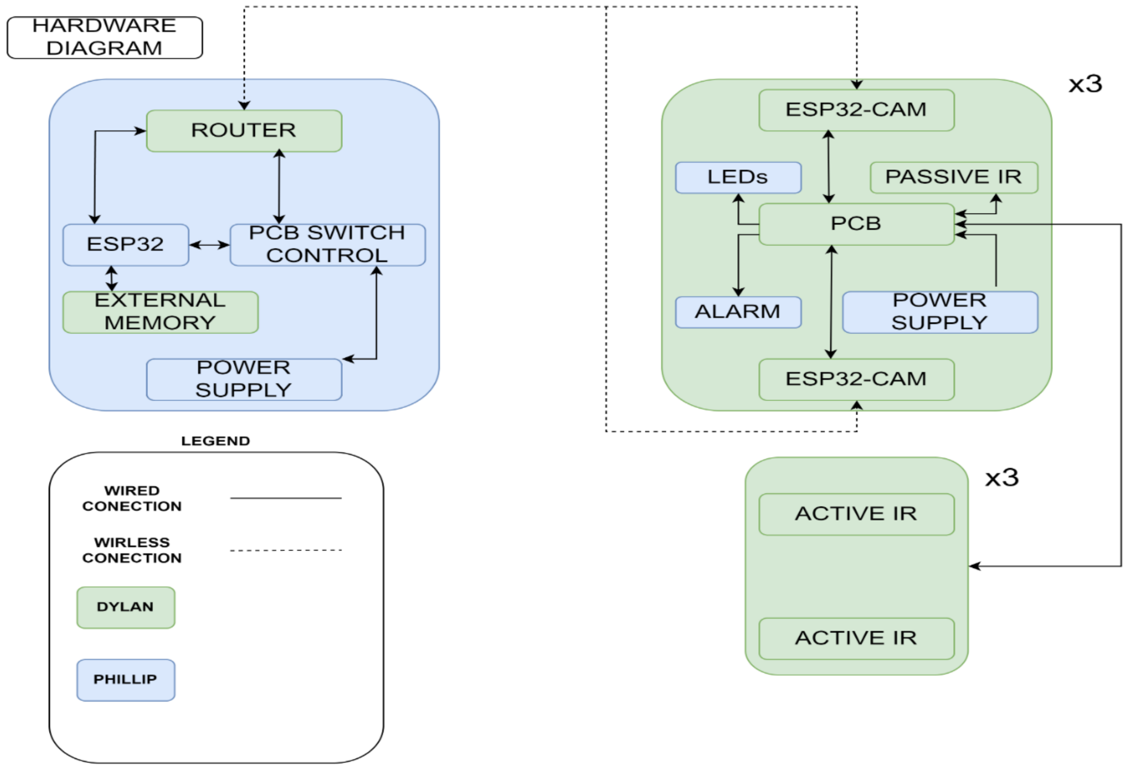



Figure 2.7: Hardware Diagram

### 2.7.1 Hardware Overview:

For the hardware of this assignment, we will be designing two unique pieces of hardware. A control hub that is referred to as "gadget", and then "nodes" that survey the area for movement. We will call the collective functionality of every piece our "system". The gadget will be in control of housing the resources capable of creating a wireless LAN, the control unit in charge of powering on and off the system, and additional memory to hold imagery data. The main method of communication between nodes and the gadget will be done through a wireless LAN. We have advanced and stretch goals mentioned above specifically calling for more capability of the gadget like having a quite mode or manually turning on the alarm mode. For both Electrical Engineers, we will need to design a PCB board for the gadget that will regulate power and house the switches for system power as well as alarm modes. There will be three nodes built to give an active perimeter. With three nodes we can have the capability to have complete coverage without any blind spots. The nodes will contain all the sensors and alarm hardware. Each node will have its own PCB built and will control the signals to and from the passive and active IRs respectively, power regulation to the node, and power on and off the alarm hardware. Our focus for selecting hardware is to save as much power as possible without neglecting the user's security. Our nodes will all have a low power mode for the peripherals that aren't always on. The devices that are always on are our passive and active IR sensors which will need to have a low power usage. The cameras and other alarm devices will be active once the system triggers into an alarmed state. This endeavor will be difficult to achieve with the project's time limit, but with efficient time management we should achieve this goal.

## 2.8 Software Diagram:

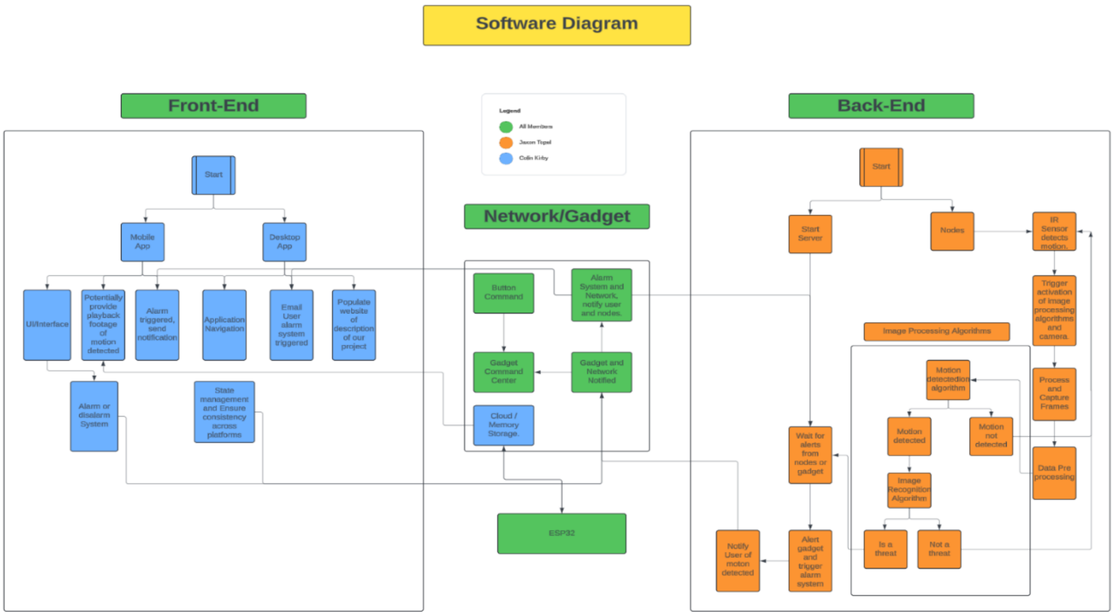



Figure 2.8: Software diagram created using Lucid Flowchart

### 2.8.1 Original Software Components Overview:

Our project involves creating a sophisticated software environment. Jaxon will take the lead on back-end development, focusing on image processing, software architecture, and developing the computer network to send information between nodes and gadgets. Kirby will oversee the development of both mobile and desktop applications for the front-end, focusing on enhancing the control our mobile applications will have on the functionality of our system. Both team members will collaborate closely as part of the software team to build a robust computer network system, ensuring seamless data communication between nodes and users.

Below is an overview of our initial software development plans. Please note that file names, concepts, and purposes are subject to change as the project evolves. For the most up to date architecture of our software, please see Chapter 7 – Software Design.

*Main.py:*

From the very start of our software (main.py) we are importing from many of the files that our database will contain. We will create the network/server that our nodes and our gadget are going to use to communicate with each other. After this we will instantiate the 3 nodes for our system, instantiate the gadget inside our network, and then add all 3 nodes to our network as well. After this, we are going to have the software on each esp32 node running at all times, therefore, we have a continuous loop that will monitor the network, look for alerts from the gadget (details discussed below), and then will monitor the environment of each node (also discussed in more detail below). From this central point of the software, we will accomplish the goals of a secure, abstract, and scalable system that will allow us to develop easily throughout the next two semesters.

### *Gadget.py:*

This file will represent the gadget inside our security system and provide all functionalities desired from it. Inside we will have the constructor that will assign the correct network to what our gadget will live in. Another function will be for the gadget to check for alerts. Inside we will call a function from network that will receive alerts inside of an array. Then we will go through the alerts and trigger the alarm for any valid alerts. Lastly, we will have the alarm trigger function. Ideas are still being discussed as to how we wish to trigger the alarm, but as of right now we plan to have some sort of loud alarm that comes from the node triggered that will allow the user to easily identify the location of the threat while also potentially fending off the threat.

### *Network.py:*

This is the file where all nodes will communicate from. We will have an initial constructor that will set the nodes and alerts as an array of values, initially empty. From there we will have an add gadget function (Still debating if this is going to be necessary), an *'add_node'* function, a function that will send alerts, and lastly a function that will receive alerts.

### *Image_processing.py*

This file will be where we use image processing algorithms to determine if the motion detected is a threat. Called from node.py, we will be using OpenCV and potentially many other libraries to implement many algorithms for image processing such as image recognition, threat level detection, and potentially more advanced algorithms that will give more detail of the threat with more accuracy. This is where the pride and joy of our software will live and most of the work done here will be new for both Jaxon and Kirby. Research is already being done for methods of interacting with our esp32 and obtaining the images and getting them into a consistent form for feature extraction and pre-processing our images. The general framework we have already consists of our initialization function which will recognize our esp32. A function that will get the images from the esp32, it has not yet been finalized as to how we wish to obtain the images, either directly from the hardware or from a web server that our esp32 will live stream to as the esp32 supports that feature. After this we will have a '*process_image'* function (the function called from node.py), here we will load the image, apply some image pre-processing techniques, and then determine if the image contains a threat or not for the image recognition algorithm. One method discovered by our software team is the potential use of a powerful library and tool called yoloV5. YoloV5 is a deep learning-based object detection model that is widely used due to its speed and accuracy for detecting images in real time systems. Further research is needed, however, I have provided this, so you have a general idea of how we wish to implement this model.

### *Node.py:*

This file serves as the representation of our nodes in our security system. Instantiated 3 times in main.py, our constructor when called will assign the node id, location, network, and then instantiate an image processor inside each node. Eventually once we get access to the hardware, we will have to assign an esp32 to each node. After the constructor, we have a function that will monitor the environment. Inside of here we will use our IR sensor to monitor for movement outside the secure area. When movement is detected, we would call the image processor algorithm to identify if the movement detected was a threat. If the movement detected was a threat, then we will send an alert to the network. Lastly, the next function is the one used above that sends an alert to the network.

*Ir_sensor.py:*

This file will be used to detect movement of the IR sensor. No code architecture has been laid out for this file; however, I anticipate that we will need at least 2 motion detection algorithms. One for monitoring the environment outside of the secure system, and another for ensuring nothing goes in or out of the perimeter of the secure area without us knowing.

*Note:*

Remember that more files may be created, and the code's architecture may change as needed to ensure our project's success.

## 2.8.2 Revised Software Components Overview:

Since software development began, many changes have happened all across the board and a complete restructuring of our software architecture had to take place as further research was done to ensure that we could meet the requirements of our project. Section 2.8.1 was kept in this document to serve as a showcase of progress that has been made since the original D&C submission.

Our new software architecture is simpler, consisting of backend code split into 2 main sections, ESP32-S3 Code (Gadget code) and ESP32-CAM Code (Node code). See below for respective sections. Keep in mind that as development of software continues in order to satisfy the objectives of the project, code restructuring may happen and will be noted in further documentation.

### 2.8.2.1 ESP32-S3 Code:

The code that will live on the gadget will be split into two main files. Main.ino and ObjectDetection.ino. Main.ino serves many different functionalities such as connecting the gadget to the router, monitoring the server for any images sent by a node, and interacting with the hardware that lives on the gadget, and the respective desired functionality for the hardware to perform. Inside of a function that is called once an image has been sent from a node to the server, the gadget receives the image, stores it respectively in PSRAM, and then calls a function inside ObjectDetection.ino. The function called ObjectDetection.ino simply returns a Boolean that will indicate whether a person has been detected inside of the image sent from a node. Inside ObjectDetection.ino, we will be using TensorFlow lite, which is described in section 3.2.3.2. ObjectDetection.ino interacts with many libraries to perform the image processing libraries and is the part of software that requires the most computational power.

### 2.8.2.2 ESP32-CAM Code:

Inside the ESP32-CAM, we will have 1 main.ino file running constantly. This code's main purpose is to interact with the hardware on the nodes we have in our system. Main.ino monitors the environment constantly in a low power mode state, implemented by a "sleep" function that turns off hardware that consumes the most power, ensuring that we can keep our system on as long as possible. Inside of this code, we connect each node to the router via http, monitor the environment by reading inside our loop function for a voltage change with our passiveIR sensor. If the passive IR sensor detects motion, we call a function that turns the camera on, captures an image, and sends it over the server that will immediately be taken care of by main.ino on the gadget. Other functions we have in this code interact with the active IR sensor, constantly monitoring this as well, and in the case that the sensor has detected motion, we will send an alert over the network and our gadget will sound an alarm.

## 2.9 Prototype Illustration:

The following illustration was created by us to show how the process of our system works. Our design idea consists of a system that lives inside of a computer network, we also will have infrared sensors as an additional measure of security, lastly along with a central gadget that will serve as the governing point of operations for the user.

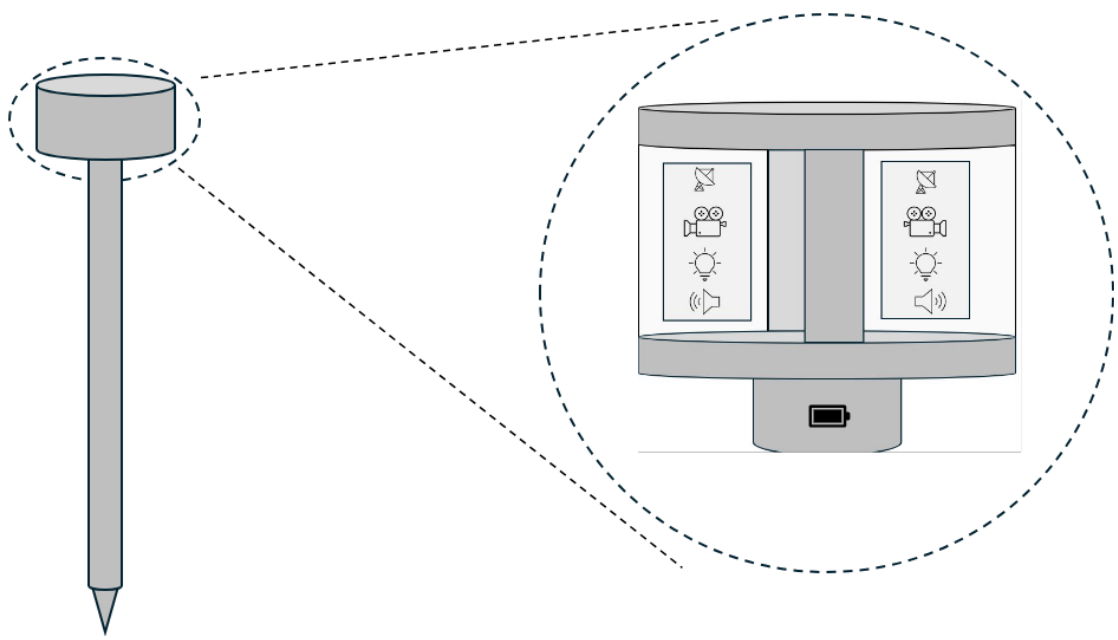


Figure 2.9.1: Prototype Illustration of a Single Node

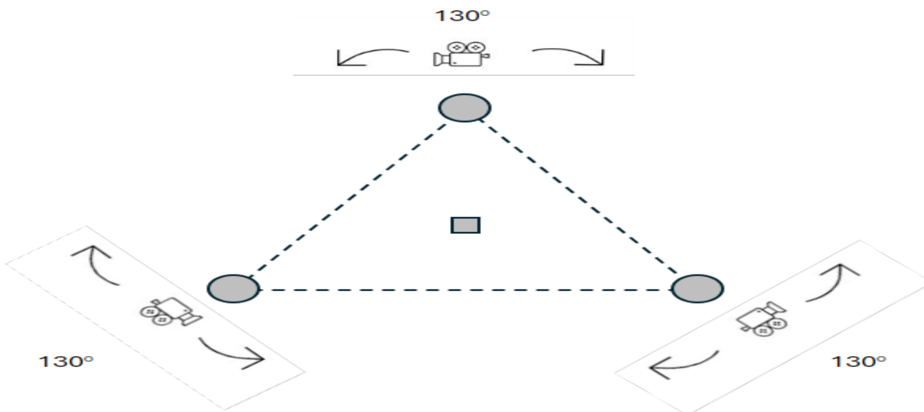



Figure 2.9.2: Overall System Prototype Including 3 Nodes and a Center Gadget

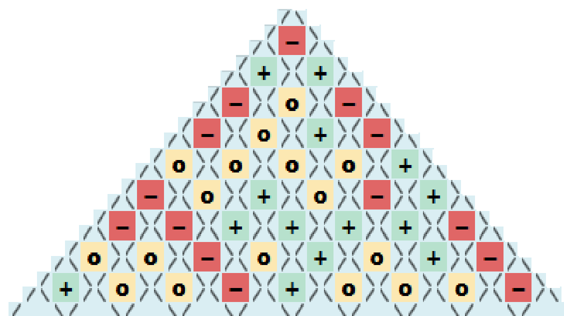
# 2.10 House of Quality:

The House of Quality shows the correlations and relationships of desirable features of our project. Following the legend will guide you to a better understanding. This figure shall guide our project to make a more educated decision when comparing tradeoffs.

| Correlations | |
|---|---|
| Positive | + |
| Negative | – |
| No Correlation | o |

| Relationships | |
|---|---|
| Strong Positive | ↑↑ |
| Weak Positive | ↑ |
| No Relation | ⊗ |
| Weak Negative | ↓ |
| Strong Negative | ↓↓ |

| Direction of Improvement | |
|---|---|
| Maximize | ▲ |
| Minimize | ▼ |



| Column # | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Functional Requirements / Customer Requirements (Explicit and Implicit) | Direction of Desirability | Size | Weight | Response Time | Communication Range | Battery Efficiency | Fault Tolerance | Signal Integrity | Weather Resistance | Latency | Cost |
| Direction of Desirability | | ▼ | ▼ | ▼ | ▲ | ▲ | ▲ | ▲ | ▲ | ▼ | ▼ |
| Affordability | ▲ | ↓↓ | ↓↓ | ↓ | ↓ | ↓↓ | ↓ | ↓ | ↓↓ | ↓↓ | ↑↑ |
| Accuracy | ▲ | ⊗ | ⊗ | ↑↑ | ⊗ | ↑↑ | ↑↑ | ↑↑ | ⊗ | ↑↑ | ↓↓ |
| Battery Life | ▲ | ↑ | ↑ | ⊗ | ⊗ | ↑↑ | ⊗ | ↑ | ↑ | ⊗ | ↓↓ |
| Environmental Adaptability | ▲ | ↑ | ↑ | ⊗ | ↑↑ | ↑↑ | ↑↑ | ↑↑ | ↑↑ | ⊗ | ↓ |
| Portability | ▲ | ↓↓ | ↓↓ | ↓ | ↑↑ | ↑↑ | ⊗ | ↑↑ | ↑↑ | ↑ | ↓ |
| User Experience | ▲ | ↓ | ↓ | ↑↑ | ↑↑ | ↑↑ | ↑ | ↑ | ↑↑ | ↓↓ | ↓↓ |
| Range | ▲ | ⊗ | ⊗ | ↓↓ | ↑↑ | ⊗ | ↑ | ↑↑ | ↑↑ | ↑ | ↓↓ |
| Set Up Difficulty | ▼ | ↓ | ↓ | ↑↑ | ↑ | ↑ | ↑↑ | ↑↑ | ⊗ | ↑ | ↓↓ |
| Reliability | ▲ | ↑ | ↑ | ↑↑ | ↑↑ | ↑↑ | ↑ | ↑ | ↑↑ | ↑↑ | ↓↓ |
| Connection Time | ▼ | ⊗ | ⊗ | ↑↑ | ↑↑ | ⊗ | ↑ | ↑↑ | ⊗ | ↑↑ | ↑↑ |
| Product Targets | | ~ 6 in^3 | ~5lbs. | > 1 s | < 20 ft | 1 - 2 Months Per Charge | ≤ 5% fail | ≥ 95% | IP65 | ≤ 100 ms | ~$300 |

Figure 2.10: House of quality

# Chapter 3 Research and Investigation

## 3.1 Research Overview:

Research was performed for our project in 3 main areas: Hardware selection and comparison, Machine Learning Algorithms and software architecture, and then front-end software. Each member oversaw their own respective sections (Dylan and Phillip covered hardware, Jaxon covered ML and software architecture, and Kirby covered Front-end integration). Each member contributed roughly 15 pages for this chapter and performed extensive research into their area, ensuring that all technology and parts were able to meet the restrictions of the assignment while meeting our main goals.

## 3.2 Technology Comparison and Selection:

### 3.2.1 Introduction:

When selecting software technologies and parts there are countless options available on the internet that allow developers to not so easily find exactly what they need to fit their needs. Selecting software that will run and be compatible with all hardware restrictions requires us to look for solutions that will maximize performance, minimize power consumption, and allow for successful integration with all hardware. Deciding which technologies to use inside of our software system design consisted of choosing between different programming languages, image processing libraries, communication protocols, Development environments, and data storage solutions.

### 3.2.2 Programming Languages:

| Technology | Performance | Control | Complexity | Memory usage |
|---|---|---|---|---|
| C/C++ | High efficiency. Ideal for low level hw interactions. | Provides almost complete control over resources, | Highest complexity out of all languages seen. | Requires manual memory management |
| Python | Slower execution compared to C++/C | Less control over hardware compared to C++ | Simple syntax, easy to learn language. | Higher memory consumption. |
| Arduino C++ | Slightly less efficient than C/C++. | More control over your program than python, less than C/C++. | Simplified C++ syntax, easier for beginners to pick up. | Less memory consumption than python, more than C++. |
| JavaScript | Slower execution compared to others. | Good for error handling, less control than C++, no hw interaction. | High level language, easy to debug. | Higher memory consumption. |

Table 3.2.2: Programming Language Comparison Table

#### 3.2.2.1 C++:

[5] "C++ was designed with Systems programming and embedded, resource-constrained software and large system in mind, with performance, efficiency, and flexibility of use as its design highlights."

Given this quote above from Wikipedia, you can now understand why using the language is so desirable for a project such as ours that requires our software to be very particular with memory management and the architecture of how our code will run. C++ gives us that advantage that other languages don't offer. Another huge reason we will be using C++ is due to the direct interaction of our software to GPIO pins that will be used to monitor our Active IR, Passive IR sensors, and more.

#### 3.2.2.2 Python:

[6] "Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly procedural), object-oriented and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library."

Python is the easiest programming language any developer can use. From doing simple algebraic equations all the way to writing deep neural networks, python provides an extensive choice of libraries that make developers lives insanely easy. Although we will not be throwing python code onto the ESP32 for many reasons, we will be using python and its extensive resources for testing out the code we have. For example, once I began development with the ESP32-CAM board that goes onto our node, I wrote a simple python script that would capture the image and display it to my computer to ensure that the hardware was working.

#### 3.2.2.3 Arduino C++:

[7] "The microcontrollers can be programmed using the C and C++ programming languages (Embedded C), using a standard API which is also known as the **Arduino Programming Language**, inspired by the Processing language and used with a modified version of the Processing IDE."

Arduino C++ is almost the same as C++. The primary difference lies in the abstractions and libraries provided by Arduino. Such Abstractions might simplify common tasks such as what our goal of object detection is. Libraries can provide functionality through providing easy to interact with functions that allow developers to avoid manipulating hardware. There are thousands of little, tiny advantages that Arduino C++ software provides and can make the lives of embedded system developers easier.

#### 3.2.2.4 JavaScript:

[8] "JavaScript is a high-level, often just-in-time compiled language the has dynamic typing, prototype-based object-orientation, and first-class functions". Expanding off the quote from Wikipedia, according to the same source [5], 99% of websites use JavaScript on the client side and webpage behavior.

JavaScript is an easy-to-use language that allows front-end developers to execute their client code and display as they wish. JavaScript's most popular use is web development, specifically creating dynamic and interactive front-end interfaces, as most interfaces you interact with today are built on JavaScript. While JavaScript is primarily used for the front-end side of development, there are still major applications for JavaScript with backend applications through environments such as with Node.js, which allows developers to use JavaScript for server-side programming as well. It allows us to use JavaScript throughout the stack, which reduces the complexity of development by reducing context switching between languages. However, because JavaScript's backend applications are not as suitable for our project's current backend applications with embedded programming it may not be ideal for embedded programming.

#### 3.2.2.5 Overview:

Out of these programming languages, requirements for our selection consisted of a programming language that requires an easy-to-use platform, while also maximizing performance, and minimizing memory usage. We will be using both python and Arduino C++. Python was chosen for its substantial amount of image processing tools available online, allowing for easier development with our image processing algorithms on our main gadget board where power consumption and memory usage is less of a worry due to the hardware selections made. We will not be putting any python code on the board; however, we will be using python for testing, especially in our early development stages as we have already been using this language to ensure our camera functionality is working by displaying images easily to the computer screen using built in functions. We will also be using python to implement our image processing ideas before we spend the tedious time working in C++ code to ensure our functionality will work as planned.

We will be using Arduino C++ for our embedded code selection due to the simplicity of the language combined with the control and efficiency it provides. C++ has many open libraries available for use with our project and will allow for the best implementation on both our Gadget and Node ESP32 boards. We will be using this language mainly due to its highly efficient coding capabilities, offering developers direct access to our board, specifically registers and cameras, allowing us to interact directly with the hardware. Another advantage of using this language is that we will have complete control of the memory management inside of our system, allowing us to free memory for flash or ram as deemed necessary. We will also be taking advantage of the object-oriented programming features this language offers, ensuring we will be abstracting as much as possible to implement easily readable code that will make debugging much easier. Overall working with C++ is going to give us a severe advantage of the control we will have over our system, allowing us to save memory, power, and time while our system is running.

### 3.2.3 Image Processing Libraries:

| Technology | Performance | Power consumption | Memory usage | Programming Language Integration |
|---|---|---|---|---|
| OpenCV | Good for image processing, not optimized for deep learning. | Moderate to high. | High memory usage. | Python, C++ |
| TensorFlow Lite | Optimized for edge devices. | Low to moderate. | Moderate, can be optimized for low memory usage. | Python, C++ |
| YoloV5 | High performance for real time object detection | Moderate to high. | High memory usage. | Python. C++ |
| ESP-WHO | Optimized for ESP32, good for face detection, needs optimization for person detection. | Designed for low power applications. | Low to moderate memory usage, optimized for esp32's memory. | C/C++ |

Table 3.2.3: Image Processing Libraries Comparison Table

#### 3.2.3.1 OpenCV:

[9] "OpenCV (Open-Source Computer Vision Library) is a library of programming functions mainly for real-time computer vision."

OpenCV was originally created by intel in 1999, and this was my first idea of an approach to our object detection goal. Implementation had begun and then was paused as more research was being done; hence we are now using TensorFlow Lite. The main reason we did not pursue an implementation with OpenCV was because we originally started our code architecture using python but faced difficulties finding a way to upload python code with the way we wanted our system to work. OpenCV is one of, if not the most powerful modern computer vision libraries today. It offers help for smaller machine learning applications such as linear regression, all the way to novel areas at the forefront of technology like deep neural networks and reinforcement learning.

#### 3.2.3.2 TensorFlow Lite:

[10] "TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks."

TensorFlow Lite is the machine learning framework that we have chosen to implement our object detection algorithm with. There are many advantages for us using this library to accomplish our goals, however the main one is due to TensorFlow Lites open-source ESP-32 implementations that will allow us to easily develop algorithms to complete our project. The other libraries mentioned, although they can be developed on the ESP-32, would either require too much computational power or not accomplish our desired task.

#### 3.2.3.3 YoloV5:

YoloV5 is a product made by Ultralytics, a company founded by graduates at Washington University. This product is one of many releases that fall under the common name Yolo (you only live once). YoloV5 was compared specifically, not because it is the best out of all the models or the most accurate, but because of the extensive open-source implementations using this version of the Yolo model. In the end, this product was not selected for implementation because it would require too much power and memory for our ESP-32 board that is quite restricted in both of those areas.

#### 3.2.3.4 ESP-WHO:

ESP-WHO is an Internet of Things (IOT) application framework, that offers extensive API’s available to developers for the area of Face recognition. It is possible to develop on the ESP-32 board, however, research was cut short for this product soon after realizing that its main applications would not meet the requirements of our project goal.

#### 3.2.3.5 Overview:

Selecting an image processing library is key to the success of our project. With the above options there are many tradeoffs that factored into our decision such as performance, Dev board compatibility, and the main metric considered was power consumption. Our ESP32 boards are not the best for high performing computations so something like YOLOv5 that would make my life a lot easier is not possible as we just do not have the power and even memory to store what we would need to. Options such as ESP-WHO and OpenCV were also considered given their features such as board compatibility, but did still not meet all the requirements we would need to perform object detection. Specifically, ESP-WHO while being made for our board, is not great for object detection and mainly used for image processing for tasks like face detection. That being said, we decided to pursue development as of now with TensorFlow lite and potentially using features of OpenCV, specifically CV2 for its image pre-processing features. This approach will allow us to meet memory requirements of our board and uses low computational power, all while allowing us to meet the main objective of the image processing algorithms which is to perform object detection identifying people and potentially more. Overall, our implementation of using both TensorFlow lite and OpenCV will allow us to accomplish our goals for the IP algorithm while meeting hardware constraints from our ESP32 board.

### 3.2.4 Communication Protocols:

| Technology | Performance | Power Consumption | Memory usage | Board integration |
|---|---|---|---|---|
| HTTP | Higher latency | Moderate, short-lived connections. | Low-Moderate, data dependent | Easy-Moderate, many available libraries |
| WebSocket | Low latency | Higher than http | Higher than http | Harder than http |

Table 3.2.4: Communication Protocols Comparison Table

#### 3.2.4.1 HTTP:

[11] "HTTP (Hypertext Transfer Protocol) is a application layer protocol in the Internet protocol suite model for distributed, collaborative, hypermedia information systems."

HTTP is a widely adopted communication protocol commonly that most are either aware of or have used in their everyday lives considering it is so prevalent in most app infrastructures used for transferring data over the internet. Its ubiquity and ease of use make it an ideal choice for systems that prioritize simplicity and broad compatibility. In SecureScape, HTTP is leveraged for communication between the nodes with the ESP32-CAM and the gadget with the ESP32-S3, which allows us to take advantage of its lightweight nature and built-in library support within the ESP32 framework. This allows developers to quickly establish communication channels with minimal setup.

A key advantage of HTTP is its secure counterpart, HTTPS, which encrypts data in transit and ensures user confidence in the system. Although HTTPS was not specifically implemented in the current design, its potential for future use aligns with SecureScape's scalability goals. Additionally, HTTP's low memory footprint makes it an attractive option for resource-constrained devices like the ESP32, ensuring that sufficient resources remain available for critical tasks such as image processing and motion detection. While HTTP introduces higher latency compared to WebSocket, this trade-off is acceptable given the system's current performance requirements and focus on ease of integration. [11]

#### 3.2.4.2 WebSocket:

[12] "WebSocket is a computer communications protocol, providing a simultaneous two-way communication channel over a single Transmission Control Protocol (TCP) connection."

For some extra background, WebSocket is a high-performance communication protocol designed for real-time, bidirectional data transfer over a single TCP connection. Its low latency and ability to maintain persistent connections make it well-suited for applications requiring continuous data streams or frequent updates. In the context of SecureScape, WebSocket could potentially enhance system responsiveness by reducing the time required for data transmission between nodes and the gadget, especially considering that it can be implemented between our ESP-32's and its lower latency in comparison to the HTTP makes it very appealing.

Despite its advantages, WebSocket was not selected for SecureScape due to its complexity and higher implementation overhead. Unlike HTTP, WebSocket support for the ESP32 is less mature, with fewer available libraries and APIs. Implementing WebSocket would require additional development time and resources, which the team determined could be better allocated to refining other aspects of the system. Furthermore, WebSocket's higher power and memory consumption were factors that conflicted with the project's emphasis on resource efficiency. While WebSocket remains a compelling option for future iterations of SecureScape, HTTP's simplicity and adequacy for the current requirements made it the more practical choice.

#### 3.2.4.3 Overview:

In our evaluation of communication protocols for SecureScape, HTTP and WebSocket emerged as the two primary options. HTTP was ultimately selected due to its ease of implementation, low resource consumption, and compatibility with existing ESP32 libraries. The protocol enables efficient data transfer between the nodes and the gadget, ensuring reliable communication without overburdening the system's power or memory capacity. By connecting all devices to the router through Wi-Fi, HTTP allows for straightforward setup and integration, leveraging the board's built-in Wi-Fi server capabilities.

While WebSocket offers superior performance with lower latency and persistent connections, its complexity and resource demands made it less suitable for our current implementation. For SecureScape's needs, HTTP strikes an optimal balance between performance, simplicity, and scalability, enabling the system to meet its objectives without unnecessary complexity. Future iterations of the system may explore WebSocket or similar technologies for further enhancing communication efficiency and responsiveness.

### 3.2.5 Development Environments:

| Technology | Ease of use | ESP32 Compatibility | Debugging capabilities |
|---|---|---|---|
| Arduino IDE | User-friendly, easy setup | Good, many libraries available | Serial print debugging |
| Visual Studio Code | Moderate, requires Platform Io setup | Good, official ESP extensions like platform IO | Advanced debugging, hardware debugging and breakpoints. |
| Eclipse IDE | More complex setup | Good, official ESP plugins | Advanced |

Table 3.2.5: Development Environments Comparison Table

#### 3.2.5.1 Overview:

Between the development environments mentioned in the table above, selecting the right tools for our workflow was a straightforward yet iterative process. Initially, we opted to use PlatformIO with Visual Studio Code for development. However, during the early stages of setup for the ESP32-CAM and ESP32-S3 boards, we encountered numerous issues, particularly with configuring the environment and monitoring output via the Serial Monitor. After troubleshooting these challenges to achieve a basic working solution, we reassessed our approach and determined that Arduino IDE would provide a simpler and more reliable development environment for our team.

The Arduino IDE was chosen as our primary tool for compiling and uploading code to ESP boards. Its integrated Serial Monitor proved invaluable for testing and debugging, enabling us to monitor board outputs effectively and ensure proper functionality. This decision was also informed by our team's prior experience with Arduino IDE, which made the transition seamless and minimized the learning curve for all developers.

At the same time, Visual Studio Code remains an essential tool in our workflow for version control and collaboration. Its intuitive interface and powerful integration with GitHub allow us to manage our codebase efficiently. By aligning Arduino IDE's working directory with the repository in Visual Studio Code, we simplified code sharing and collaborative development, compensating for the Arduino IDE's limited collaboration features. This dual-environment approach enables us to leverage the strengths of both tools: Arduino IDE for straightforward embedded development and testing, and Visual Studio Code for robust version control and team collaboration.

### 3.2.6 GitHub:

For our project, we are utilizing GitHub, a powerful platform designed for version control and collaborative development. GitHub provides a git-based environment that enables developers and engineers to efficiently share, store, and manage code in real time. Our team is leveraging GitHub to streamline collaboration, ensuring that all members work on the most up-to-date codebase while minimizing version conflicts. For more details, you can access our repository here: GitHub Link - SecureScape

### 3.2.7 Mobile App Frameworks:

When evaluating mobile app frameworks for SecureScape, several factors were considered, including performance, cross-platform compatibility, ease of use, UI/UX flexibility, and memory usage. The following sections provide an in-depth analysis of popular frameworks, highlighting their strengths and limitations to guide the decision-making process.

| Technology | Performance | Ease of Use | Cross-Platform Compatibility | UI/UX Flexibility | Memory Usage |
|---|---|---|---|---|---|
| Flutter | High, near-native performance | Moderate | Excellent (iOS, Android) | High, customizable UI | Moderate |
| React Native | High, but slower than Flutter | High, JavaScript-based | Excellent (iOS, Android) | Moderate | Moderate |
| Ionic | Moderate, uses web tech (HTML, CSS, JS) | High, web-based | Excellent (iOS, Android, PWA) | Moderate | High |
| Xamarin | High, near native | Moderate, requires C# | Excellent (iOS, Android, Windows) | High, but complex to implement | Moderate |

Table 3.2.7: Mobile App Frameworks Comparison Table

**3.2.7.1 Flutter:**

[13] “Flutter's control of its rendering pipeline simplifies multi-platform support as identical UI code can be used for all target platforms.”

Flutter is a standout framework due to its innovative approach to multi-platform development. By controlling its rendering pipeline, Flutter allows developers to use identical UI code across all target platforms, including iOS and Android. This eliminates the need for platform-specific adjustments and or any extra context switching, significantly reducing development time and complexity. For SecureScape, this capability aligns perfectly with the project’s requirement for a synchronized experience between mobile and web platforms, ensuring that the app looks and functions consistently across devices.

In addition to its performance, Flutter is well known for its high UI customizability. Its widget-based architecture allows developers to create intricate and visually engaging user interfaces tailored to the needs of specific applications. This is particularly important for SecureScape, where a polished and interactive UI enhances the user’s experience in managing security nodes and monitoring real-time alerts. Flutter’s rendering engine, Skia, ensures that these customizations perform seamlessly, delivering near-native performance regardless of the platform.

However, Flutter’s reliance on the Dart programming language may present a learning curve for developers unfamiliar with the language. While Dart is intuitive and well-documented, transitioning to a new language can require additional time and effort during the initial phases of development, especially with no familiarity with said language. Despite this, Flutter’s growing developer community and extensive documentation provide ample support, which makes it easier to overcome these challenges. The framework’s ability to streamline the development process while maintaining high performance and aesthetic flexibility makes it a top contender for projects requiring complex, multi-platform apps. [13]

**3.2.7.2 React Native:**

[14] “Reusable components are a key advantage of React Native. In addition to cost efficiency, reusing large amounts of code across platforms allows for shorter development time and a simplified development process overall.”

React Native’s primary strength lies its reusability of components across platforms, making it an efficient choice for cross-platform development. By allowing developers to write code once and deploy it to both iOS and Android, React Native reduces the time and resources required for application development. This feature is particularly appealing for teams with tight deadlines or limited budgets, as it enables quicker iterations and faster delivery.

One of React Native’s key advantages is its foundation in JavaScript, a language widely known and used in the development community, and considering our team has experience in JavaScript it becomes a very appealing option. Moreover, this familiarity makes React Native more accessible to developers who may already have experience with web development. As a result, the onboarding process is often faster, allowing for smoother collaboration and accelerated project timelines. This ease of use makes it an attractive option for projects where rapid development is a priority.

However, React Native’s performance can be limited by its reliance on a bridge to communicate between JavaScript and native code. This bridging process introduces some latency, which can affect the responsiveness of the application, particularly in cases where complex or dynamic UIs are required. For SecureScape, where real-time functionality and highly interactive components are critical, this performance trade-off may hinder its suitability. Despite its flexibility and cost efficiency, these limitations suggest React Native is better suited for simpler applications or those with less intensive performance requirements. [14]

**3.2.7.3 Ionic:**

[15] “It provides a set of pre-designed UI components and tools for building high-quality, interactive applications.”

Ionic excels in providing pre-designed UI components and tools that simplify the development of interactive applications. By leveraging popular web technologies such as HTML, CSS, and JavaScript, Ionic offers a familiar development environment that reduces the learning curve for teams already experienced in web development. This ease of use is particularly advantageous for rapid prototyping, which throughout our course is necessary considering the development timeframe available, it is advantageous as developers can quickly assemble functional interfaces with minimal effort.

The framework’s wide compatibility across platforms, including iOS, Android, and Progressive Web Apps (PWAs), further enhances its appeal. Teams looking to target multiple platforms without investing heavily in native development can achieve broad reach with a single codebase. Additionally, Ionic’s strong community and extensive library of plugins provide developers with a wealth of resources to extend the framework’s functionality.

However, as a web-based framework, Ionic’s performance does not match that of native or near-native solutions like Flutter and Xamarin. Applications built with Ionic may experience slower load times and reduced responsiveness, particularly in resource-intensive scenarios such as real-time updates or image processing. For SecureScape, where performance and responsiveness are critical, these limitations make Ionic less ideal despite its ease of use and rapid development capabilities. [15]

#### 3.2.7.4 Xamarin:

[16] "With a C#-shared codebase, developers can use Xamarin tools to write native Android, iOS, and Windows apps with native user interfaces and share code across multiple platforms, including Windows, macOS, and Linux."

Xamarin's use of a C#-shared codebase enables developers to create native Android, iOS, and Windows apps with high performance and native functionality. This shared architecture not only streamlines the development process by allowing code reuse but also ensures that applications maintain the look and feel of native apps on each platform. For projects requiring tight integration with platform-specific features, Xamarin provides a great solution.

In addition to its performance benefits, Xamarin's backing by Microsoft ensures continuous updates and a strong support ecosystem. Developers can leverage tools like Visual Studio and Azure to enhance the development experience, adding scalability and enterprise-level capabilities to the framework. This makes Xamarin a preferred choice for large-scale projects or those requiring advanced features.

Despite these advantages, Xamarin's reliance on C# can pose challenges for teams without prior experience in the language, especially in the app development realm. The framework's complexity in creating highly customized UIs may also increase development time, particularly for smaller teams or those working on simpler applications. For SecureScape, where speed and simplicity are priorities, these factors may outweigh Xamarin's strengths, making it less suitable for the project's specific needs. [16]

#### 3.2.7.5 Overview:

After proper evaluation of these frameworks, Flutter emerged as the best choice for SecureScape due to its ability to deliver high performance, near-native capabilities, and exceptional support for complex, customizable UI designs. Among the frameworks considered, Flutter offers the most balanced combination of speed, flexibility, and scalability, making it ideal for SecureScape's requirements, which include real-time image processing and consistent user experience across platforms.

React Native was initially considered for its ease of use and JavaScript-based foundation, which alights with our team's experience. However, upon deeper analysis, its reliance on a bridge for communication between JavaScript and native code presented potential performance limitations that could impact the responsiveness required for SecureScape's real-time functionality. Ionic and Xamarin, while strong contenders in their respective areas, were less suitable due to performance trade-offs and increased development complexity. Ultimately, Flutter's robust rendering pipeline, active developer community, and ability to deliver high-quality interfaces efficiently solidified its position as the optimal framework for the SecureScape mobile application.

### 3.2.8 Mobile App Hosting Comparison:

| Technology | Performance | Cost | Scalability | Ease of Use |
|---|---|---|---|---|
| AWS Lambda | High, serverless and event-driven | Pay-per-use (cost-efficient for small workloads) | Automatic, scales with demand | Moderate, requires knowledge of serverless architecture |
| Firebase | Moderate, real-time database optimized | Free for limited usage, pay-as-you-go | High, auto-scaling | High, easy integration with mobile apps and real-time databases |
| Heroku | Moderate, simplified deployment | Free tier, paid plans for higher usage | Good, but limited compared to serverless platforms | High, very easy to set up and deploy |
| DigitalOcean | High, requires manual server setup | Low-cost, starts at $5/month | High, manual scaling options | Moderate, more complex than Heroku and Firebase |

Table 3.2.8: Mobile App Hosting Comparison Table

#### 3.2.8.1 AWS Lambda:

[17] "AWS lambda are server-less compute functions are fully managed by the AWS where developers can run their code without worrying about servers. AWS lambda functions will allow you to run the code without provisioning or managing servers."

AWS Lambda provides a serverless architecture that allows developers to execute code without the need to manage or provision servers. This functionality is particularly beneficial for event-driven applications, where workloads can fluctuate significantly, as they scale automatically based on demand. For SecureScape, which requires responsive data processing for real-time alerts and image handling, this scalability ensures that the system performs reliably under varying loads.

The pay-per-use model further enhances AWS Lambda's appeal. By charging only for the compute time consumed, it offers cost efficiency for projects with intermittent workloads. This is ideal for SecureScape, where certain tasks, such as image uploads or alarm triggers, are executed sporadically rather than continuously, meaning that the usage will be charged based on use, rather than set fees. However, implementing AWS Lambda requires familiarity with serverless architecture, which could pose a learning curve for teams inexperienced with its ecosystem. Despite this, its flexibility, scalability, and cost-effectiveness make it a strong contender for hosting solutions. [17]

#### 3.2.8.2 Firebase:

[18] "It helps developers to build their apps faster and in a more secure way. No programming is required on the firebase side, which makes it easy to use its features more efficiently. It provides services to android, iOS, web, and unity. It provides cloud storage. It uses NoSQL for the database for the storage of data."

Firebase is a comprehensive backend-as-a-service platform that simplifies the development process by providing an integrated real-time database, cloud storage, and authentication services. Its ability to synchronize data seamlessly across Android, iOS, and web platforms ensures consistent user experience, which is critical for SecureScape's multi-platform functionality. Firebase's NoSQL database structure supports flexible data handling, making it well-suited for dynamic applications like SecureScape, where real-time data synchronization is essential.

One of Firebase's standout features is its ease of use, throughout set up and any necessary adjustments the UI for development is very easy to understand. Developers can integrate Firebase into their applications with minimal configuration, eliminating the need for extensive backend programming. This accelerates the development process, allowing teams to focus on refining the application's features rather than managing backend infrastructure. However, Firebase's performance may lag slightly behind serverless solutions like AWS Lambda in resource-intensive scenarios, which could affect its viability for tasks requiring significant processing power. Nonetheless, its scalability and integration capabilities make it a strong option for mobile app hosting. [18]

**3.2.8.3 Heroku:**

[19] "Heroku is a cloud platform as a service supporting several programming languages where a user can deploy, manage and scale their applications. It is widely used to deploy server-based web applications, APIs, discord bots, and more."

Heroku is a platform-as-a-service (PaaS) solution that simplifies app deployment and management. Its user-friendly interface allows developers to push code changes directly to production without worrying about complex infrastructure setup. This ease of use makes Heroku particularly appealing for smaller projects or prototypes where quick deployment is a priority. Furthermore, considering our group's experience with Heroku, it is a very viable option.

Despite its simplicity, Heroku has limitations in scalability and performance compared to serverless platforms like AWS Lambda. For SecureScape, which requires a highly scalable and flexible hosting solution to handle real-time data processing, these limitations may hinder its suitability. While Heroku provides an excellent environment for rapid development and testing, its higher costs at scale and restricted flexibility make it less ideal for the long-term hosting needs of SecureScape. [19]

**3.2.8.4 DigitalOcean:**

[20] "DigitalOcean provides developers, startups, and SMBs with cloud infrastructure-as-a-service platforms."

DigitalOcean offers a cloud infrastructure-as-a-service platform that provides developers with more control over server setup and configuration compared to other hosting solutions. This level of control allows teams to fine-tune their hosting environment to meet specific requirements, which can be beneficial for SecureScape's unique workload and processing demands.

However, the manual nature of DigitalOcean's setup process requires a higher level of expertise, making it less user-friendly than platforms like Firebase or Heroku. While its low cost and flexibility are appealing for smaller applications or businesses, the additional complexity may introduce challenges for teams looking to focus on application development rather than infrastructure management. For SecureScape, where ease of deployment and scalability are critical, DigitalOcean may not align as well with the project's goals as other serverless options. [20]

#### 3.2.8.5 Overview:

AWS Lambda emerged as the optimal hosting solution for SecureScape due to its serverless architecture, which provides automatic scaling and eliminates the need for server management. Its event-driven design aligns perfectly with our project's workload, enabling efficient handling of tasks such as real-time alerts and image uploads, which are the key features of our app. The pay-per-use model also offers significant cost savings, particularly for applications with variable workloads. While Firebase and other solutions provided strong integration capabilities and ease of use, AWS Lambda's unmatched scalability and flexibility ensure that SecureScape remains efficient and reliable, even as its user base and functionality expand.

### 3.2.9 Frontend Frameworks for User Interactions:

| Technology | Ease of Use | Performance | Learning Curve | Compatibility with HTML / CSS |
|---|---|---|---|---|
| jQuery | Easy | Moderate | Low | High |
| Alpine.js | Easy | High | Low | High |
| Stimulus.js | Easy | High | Low | High |

Table 3.2.9: Frontend Frameworks for User Interactions Comparison Table.

#### 3.2.9.1 Overview:

Prioritizing performance and ease of use, Alpine.js stood out, considering its lightweight nature, high performance, and our members' previous experience, we believe it is the best choice for us. Its integration with HTML and CSS, and a low learning curve offers the balance for creative interactive UIs for our project.

### 3.2.10 UI Component Libraries:

| Technology | Ease of Use | Customizability | Performance | Library Size |
|---|---|---|---|---|
| Material UI | High | High | High | Large |
| Ant Design | Moderate | Moderate | High | Large |
| Tailwind CSS | Moderate | High | High | Small/Lightweight |
| Bootstrap | Easy | Moderate | Moderate | Large |

Table 3.2.10: UI Component Libraries Comparison Table

#### 3.2.10.1 Overview:

In developing SecureScape, selecting the right UI component library was essential to ensuring a fast, responsive, and user-friendly interface. Tailwind CSS was chosen as our primary library for its high customizability, lightweight nature, and exceptional performance. Its utility-first approach enables rapid development with pre-defined classes that streamline the design process and ensure the interface remains consistent and efficient. Additionally, Tailwind’s integration with existing frameworks allows for the creation of responsive layouts optimized for various devices, ensuring SecureScape’s usability across mobile and web platforms.

Tailwind CSS excels in facilitating responsive design, a key requirement for SecureScape. For instance, its grid and flexbox utilities allow the Home Page to adapt seamlessly to different screen sizes, ensuring that features like alarms, live feeds, and notifications are accessible and functional across devices. Similarly, the responsive typography utilities provided by Tailwind, such as text-base, sm:text-lg, and md:text-xl, ensure that critical alerts and notifications are appropriately scaled, enhancing readability and maintaining the system’s usability regardless of the viewing platform.

By leveraging these features, Tailwind CSS provides SecureScape with a strong foundation for developing a responsive and efficient user interface. This focus on adaptability ensures that users can interact with the system effectively, whether on a desktop or mobile device, meeting the project’s goals of accessibility and real-time responsiveness.

## 3.2.11 Database for App Backend:

| Technology | Ease of Use | Performance | Scalability | Security Features |
|---|---|---|---|---|
| Firebase Firestore | High | High | High | Built-in auth features |
| MongoDB Atlas | Moderate | High | High | Needs external auth setup |
| PostgreSQL | Moderate | High | High | Secure but complex to set up |
| Supabase | High | High | High | Built-in user authentication |

Table 3.2.11: Database for App Backend Comparison Table

### 3.2.11.1 Firebase Firestore:

[21] “Firestore is a NoSQL document database built for automatic scaling, high performance, and ease of application development. While the Firestore interface has many of the same features as traditional databases, as a NoSQL database it differs from them in the way it describes relationships between data objects.”

Firestore offers a NoSQL document-based structure designed to simplify application development with automatic scaling and high performance. For SecureScape, its ability to synchronize real-time data across devices and platforms is crucial for maintaining consistent and responsive system behavior. Firestore's automatic scaling ensures that as more devices and nodes are added to the network, the database can handle increased workloads without manual intervention.

Its ease of use is another significant advantage. Developers can quickly integrate Firestore into applications without needing extensive backend expertise, enabling the team to focus on core functionality rather than infrastructure management. Additionally, Firestore’s built-in authentication features enhance security, allowing sensitive data like sensor logs and user alerts to be protected effectively. Despite its many benefits, Firestore's NoSQL nature may require developers to adjust traditional relational database methodologies, as it organizes data differently, which may pose issues for our team. Nonetheless, its scalability, performance, and intuitive setup make it an excellent choice for dynamic applications like SecureScape. [21]

**3.2.11.2 MongoDB Atlas:**

[22] “MongoDB is a source-available, cross-platform, document-oriented database program. Classified as a NoSQL database product, MongoDB utilizes JSON-like documents with optional schemas.”

MongoDB Atlas is a flexible and powerful NoSQL database solution, particularly well-suited for applications that require dynamic and customizable data storage. By utilizing a document-oriented structure, MongoDB allows developers to store and query data in JSON-like formats, making it ideal for projects that involve diverse and complex data relationships. For SecureScape, this flexibility could be advantageous when dealing with data from multiple security nodes, each potentially producing unique types of logs and alerts. Furthermore, this team’s experience with using MongoDB on other web app development projects makes it a very appealing option.

However, MongoDB’s lack of built-in authentication features requires developers to implement additional security measures, which may increase development complexity. While this extra step may offer more control over authentication processes, it also places a greater burden on the development team. MongoDB’s ability to scale effectively and its high performance under heavy workloads remain significant strengths, making it a strong candidate for projects requiring detailed data customization. [22]

**3.2.11.3 PostgreSQL:**

[23] “PostgreSQL is an advanced relational database system. PostgreSQL supports both relational (SQL) and non-relational (JSON) queries. PostgreSQL is free and open-source.”

PostgreSQL stands out as a robust and versatile relational database system that also supports non-relational data formats such as JSON. This hybrid functionality makes it an appealing choice for applications that need to work with both structured and semi-structured data. SecureScape could benefit from this feature, as it combines relational data models for node configurations and user details with semi-structured formats for alerts and logs.

PostgreSQL is widely regarded for its reliability, being free, open-source, and feature-rich. However, its complexity in setup and maintenance can pose challenges for teams without prior experience, our team does have minimal experience using PostgreSQL but implementing a database with alternating data types might create a new realm of issues that could hinder development. Implementing PostgreSQL often involves more time and effort compared to NoSQL alternatives like Firestore or MongoDB. Despite these challenges, its scalability and robust security measures make PostgreSQL an excellent option for applications with a need for complex queries and data relationships. [23]

#### 3.2.11.4 Supabase:

[24] "Every Supabase project comes with a full Postgres database, a free and open source database which is considered one of the world's most stable and advanced databases... Supabase Auth makes it easy to implement authentication and authorization in your app. We provide client SDKs and API endpoints to help you create and manage users.

Supabase offers a managed PostgreSQL backend with modern features such as built-in authentication and real-time capabilities. For SecureScape, these features simplify the development process by providing user authentication out of the box, enabling the team to focus on building other critical components of the system. The integration of a stable and advanced PostgreSQL database ensures reliable data handling and scalability.

Another advantage of Supabase is its ease of integration with modern development workflows. It provides developers with client SDKs and API endpoints that streamline tasks like user management and data queries. While Supabase offers many benefits, its dependency on PostgreSQL means developers still face some of the challenges associated with relational database setups, and as mentioned within the previous sub chapter because of PostgreSQL's notoriety in complexity it becomes a weaker option for our database. However, for projects needing a full-featured backend with real-time capabilities, Supabase presents a compelling solution. [24]

#### 3.2.11.5 Overview:

For our deployable security system, Firebase Firestore emerged as the most suitable database due to its automatic scaling, real-time synchronization capabilities, and ease of integration. Its NoSQL document model allows for flexible and efficient data handling, which is essential for managing real-time alerts, logs, and sensor data from multiple security nodes. Built-in authentication features and cloud storage further enhance its security and reliability, streamlining development and reducing infrastructure concerns. While alternatives like MongoDB Atlas and PostgreSQL offer significant flexibility and advanced functionality, Firestore's ability to handle dynamic workloads and its developer-friendly interface make it the optimal choice for SecureScape's backend needs.

### 3.2.12 State Management Libraries:

Effective state management is a cornerstone of modern application development, ensuring data consistency and efficient updates across components. For SecureScape, the choice of state management library impacts the overall performance, scalability, and maintainability of the application. The following sections evaluate four popular state management libraries—Redux, MobX, Provider, and Vuex—based on their features, ease of use, and integration with specific frameworks.

| Technology | Ease of Use | Performance | Scalability | Integration with Framework |
|---|---|---|---|---|
| Redux | Moderate | High | High | Excellent (React, Angular) |
| MobX | High | High | Moderate | Excellent (React, Vue.js) |
| Provider | High | High | High | Excellent (Flutter) |
| Vuex | Moderate | High | Moderate | Excellent (Vue.js) |

Table 3.2.12: State Management Libraries Comparison Table

**3.2.12.1 Redux:**

[46] "Redux is an open-source JavaScript library for managing application state. It is most commonly used with libraries such as React or Angular for building user interfaces."

Redux is one of the most widely adopted state management libraries, known for its reliability and compatibility with frameworks like React and Angular. At its core, Redux provides a predictable state container, ensuring that the application state is managed centrally and updated in a consistent manner. This predictability is particularly advantageous with our applications for SecureScape, where real-time updates to the interface are critical.

Despite its advantages, Redux has a moderate learning curve due to its reliance on concepts like reducers, actions, and middleware. While this can increase initial development effort, the trade-off is a scalable and high-performing state management solution. Redux's middleware ecosystem, including tools like redux-thunk and redux-saga, enables developers to manage asynchronous operations effectively, making it well-suited for complex workflows such as SecureScape's real-time alerts and logs. However, its wordiness with outputs can result in boilerplate code, which might slow down rapid development. [46]

**3.2.12.2 MobX:**

[47] "MobX is a signal based, battle-tested library that makes state management simple and scalable by transparently applying functional reactive programming. The philosophy behind MobX is simple: … Straightforward … Effortless optimal rendering… Architectural Freedom…"

MobX offers a reactive approach to state management, focusing on simplicity and scalability. Unlike Redux, which relies on a strict structure, MobX allows developers to manage state dynamically, automatically tracking changes and updating components as needed. This reactive nature would reduce the overhead of manual managing state, making it an attractive option for projects like SecureScape that need real-time synchronization of data.

MobX's straightforward API minimizes the amount of boilerplate code required, allowing for faster implementation and reducing development complexity, making it very appealing to us considering our lack of experience with State Management Libraries. Moreover, its ability to optimize rendering by updating only the components affected by state changes enhances performance, especially in UI-heavy applications. However, its scalability is considered moderate compared to Redux, as its flexibility can lead to less predictable state management in larger applications. Nevertheless, MobX is an excellent choice for smaller or medium-sized projects requiring responsive, dynamic interfaces.

**3.2.12.3 Provider:**

[48] "By using provider you get: simplified allocation/disposal of resources, lazy loading, a vastly reduced boilerplate over making a new class every time, increased scalability…"

Provider is a lightweight state management solution specifically tailored for Flutter applications. Its simplicity and direct integration with Flutter's widget tree make it a natural choice for managing state in SecureScape. By reducing boilerplate code, Provider streamlines the allocation and disposal of resources, ensuring that the application runs efficiently without unnecessary overhead. Considering our choice of Flutter for the Mobile App Framework, during our research we realized that this would be our most likely option as to prevent any unnecessary context switching.

One of Provider's standout features is its support for lazy loading, which optimizes resource usage by loading data only when required. This is particularly valuable for SecureScape's functionality, such as loading images or fetching logs from the database. Additionally, Provider's scalability allows it to handle growing application complexity while maintaining high performance. Its tight integration with Flutter ensures seamless implementation, making it a highly efficient and user-friendly solution for SecureScape's state management needs.

**3.2.12.4 Vuex:**

[49] "Vuex is a state management pattern + library for Vue.js applications. It serves as a centralized store for all the components in an application, with rules ensuring that the state can only be mutated in a predictable fashion."

Vuex is the standard state management library for Vue.js applications, providing a centralized store to manage application state predictably. This predictability is achieved through strict rules for state mutations, ensuring that updates occur in a controlled and traceable manner. For SecureScape, where maintaining consistent state across components is critical, Vuex's structured approach ensures reliability and data integrity.

However, Vuex requires a moderate level of effort to implement, as developers must adhere to its rigid patterns for managing state. While this ensures scalability and reduces the likelihood of errors, it may increase development time compared to more flexible solutions like MobX. Despite these challenges, Vuex's deep integration with Vue.js and its ability to manage complex state requirements make it a strong contender for applications built on the Vue framework and since ours will be developed with Flutter it seems redundant to consider it as an option.

**3.2.12.5 Overview:**

After evaluating the state management libraries, Provider was selected as the optimal choice for SecureScape due to its easy integration with Flutter, the mobile app framework chosen for the project. The provider's lightweight state management capabilities align perfectly with Flutter's widget-driven architecture, ensuring a natural fit for our application. Its ability to reduce boilerplate code simplifies development, enabling the team to focus more time on other aspects of our project.

Provider also offers features such as lazy loading, which optimizes resource usage by only loading data when needed—critical for maintaining the responsiveness of SecureScape's real-time functionalities like notifications and image retrieval. Additionally, its scalability ensures that SecureScape can accommodate future enhancements which is ideal considering we want each network of nodes to be able to pick up new nodes with ease for the user. Moreover, these future enhancements such as adding new features or integrating more nodes should be able to be implemented without compromising performance which further strengthens Provider's appeal. By leveraging Provider, SecureScape benefits from a highly efficient and cohesive development process, ensuring the application's state management is both robust and adaptable to evolving requirements.

# 3.3 Parts Comparison and Selection:

## 3.3.1 Camera:

In our project, we will need four to six cameras. They will need to be able to stream video and have a high enough resolution to perform image processing. These two requirements conflict with our other two needs, which are low cost and low power drain. Picking the correct technology and camera will be a balancing act of the two sides of our needs. The camera has a lot of constraints on it. The one we choose needs to be able to produce video, have a low power mode, be integrated with the system, and be low cost enough for multiple cameras to fit into the budget. We are looking at three technologies, the micro CCV cameras, basic off-the-shelf webcams, and MCU level cameras.

### 3.3.1.1 Technology Comparison Table:

| Technology | CCV | Webcam | MCU Level |
|---|---|---|---|
| Cost: > 20$ | ↓ | ↓↓ | ↑ |
| Resolution Optimal: 1080p | ↑ | ↑↑ | ↓ |
| Power Consumption | ↓ | ↓ | ↑ |
| Integration Ease | ↓ | ↓↓ | ↑↑ |
| Video | ↑ | ↑ | ↑ |

Table 3.3.1.1: Camera Technology Comparison Table

#### 3.3.1.2 Micro CCV:

| Part | Cuifati | BCOOSS | EYBEAR |
|---|---|---|---|
| Cost | $9.99 | $17.99 | $26.99 |
| Resolution Optimal: 1080p | 720p | 1080p | 720p |
| Viewing Angle | 170 degrees | 150 degrees | 70 degrees |
| Ingress Protection Rating | IP68 | - | IP67 |
| Dimensions | - | 1.69 x 1.69 x 0.98 inches | 3.5 x 2.32 x 1.69 inches |
| link | Cuifati | BCOOSS | EYBEAR |

Table 3.3.1.2: Micro CCV Comparison Table

The Micro CCV doesn't look to be a good solution. While a few of them have some good features like night vision and onboard memory, the core functionality is subpar. The average cost is high considering that we will be buying 6 of them. Along with that our PBC board would need multiple Video ports, along with receiving and managing 2 video feeds at once.

#### 3.3.1.3 Webcam:

| Part | STINYTECH | PEGATISAN | ARDUCAM |
|---|---|---|---|
| Cost | $49.99 | $40.59 | $38.99 |
| Resolution Optimal:1080p | 1080P | 1080P | 4K |
| Display Size | 2 inches | 2.8 inches | 4 inches |
| Integration Ease | ↑ | ↑↑ | ↑ |
| Viewing Angle | 60 degrees | 90-120 degrees | 60 degrees |
| Dimensions | 4.25 x 3.03 x 2.13 inches | 4.72 x 1.18 x 1.57 inches | 1.73 x 0.98 x 1.73 inches |
| link | STINYTECH | PEGATISAN | ARDUCAM |

Table 3.3.1.3: Webcam Comparison Table

Webcam is a better solution technically wise. With extra functions like Wi-Fi connectivity and onboard memory storage the webcam would make other parts of the project easier. The downside is they are larger than the other two technologies and the fact that they are a fully developed product they all have software and applications already installed. This would mean either removing the software and OS settings or integrating them. This could be impossible depending on what product we get. The best selection, however, would be the Pegistan camera because of its viewing angle. The wide lens is imperative for our system while the resolution quality is not as important.

### 3.3.1.4 MCU level camera:

| Part | ESP32-CAM | Raspberry Pi 4 B 3 B+ Camera Module | WayPonDEV CanMV-K230 |
|---|---|---|---|
| Cost per MCU | $7.67 | $43.47 | $25.99 |
| Video Resolution | 480p | 1080p | 3840 x 2160 |
| Power Consumption | 260 mA/h | 2.85 watts | 185mW-585mW |
| Integration Ease | ↑ | ↓ | ↓ |
| Board Dimension | 4.37 x 3.7 x 1.14 inches | 3.54 x 3.54 x 0.39 inches | 3.35 x 2.2 x 0.3 inches |
| link | ESP32-CAM | Raspberry Pi | WayPonDEV |

Table 3.3.1.4: MCU Level Camera Comparison Table

The MCU level is clearly the best choice for our application. Any cost that is incurred is outweighed by the ease of integration. Each choice would not only be the camera but would also be the backbone of the rest of the product. Each of these choices can natively be integrated with other applications and parts. The only downside is video quality, but this point will be negligible for our system. The ESP32-CAM specifically will be used because of its familiarity and low cost.

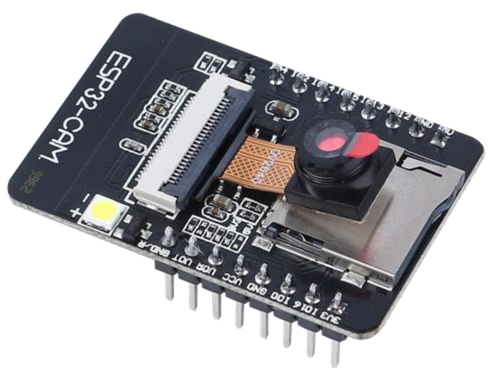



Figure 3.3.1: Camera Choice

## 3.3.2 Router:

WiFi routers are devices that allow multiple devices to connect to the internet wirelessly. Normally they work by receiving data from a modem, then broadcasting it as a wireless signal. But we will be using one as the backbone of our Wireless LAN. They connect devices like phones, laptops, and smart gadgets to this signal and access the internet. Most modern routers support multiple frequencies, allowing better performance and range depending on the environment. But due to the ESP32 only having the ability for 2.4 Ghz band, we will be staying on this band.

The 2.4 GHz band is a widely used frequency for Wi-Fi connections. It has a longer range than higher frequency bands, like 5 GHz, but tends to be slower. This is both good and bad for us. One hand the area covered will give us flexibility where we deploy our nodes but trying to push video through this band could be a problem. 2.4 Ghz can pass through walls and other obstacles more easily. Making it good for larger areas. However, many devices such as Bluetooth and microwaves also operate on 2.4 GHz, this could be problematic in high population areas but in the woods, it should be negatable problem. Despite these limitations, it remains a reliable option for basic internet needs and larger coverage areas.

Travel routers are smaller and more portable than full-sized routers. They are convenient for people who need to connect multiple devices to the internet while traveling. They offer easy setup in hotel rooms/public spaces. However, they often have fewer features, like lower signal strength and fewer Ethernet ports. Also, they might not perform as well as full-sized routers for heavy internet use. The Travel type should be fine for a localized wireless LAN.

**3.3.2.1 Technology Comparison Table:**

| Part | GL.iNet GL-SFT1200 | TP-Link AC750 | GL.iNet GL-AR300M16-Ext Portable |
|---|---|---|---|
| Cost | $34.90 | $33.99 | $36.90 |
| CPU | SF19A28, Dual-Core @1GHz | IPQ4018, Quad Core@717Mhz | QCA9531, @650MHz SoC |
| Product Dimensions | 4.65"L x 3.35"W x 1.18"H | 2.91"L x 0.87"W x 2.64"H | 2.28 x 2.28 x 0.98 inches |
| EAP Support | No | Yes | Yes |
| Memory | DDR3 128MB | DDR3L 128MB | DDR2 128MB |
| Storage | SPI NAND Flash 128MB | - | FLASH 16MB |
| Speed | Max. 300Mbps | Max. 400Mbps | 300Mbps |
| Link | GL-SFT1200 | TP-Link | AR300M16 |

Table 3.3.2.1: Router Technology Comparison Table


Figure 3.3.2: Router Choice

### 3.3.3 Memory:

With us needing to be offline we will need to hold data and processes locally. The users of the Wireless LAN will need to access the app and website offline so it will need to be hosted locally. Then on top of that the image processing will need to be done locally too. With all these functions needing to be hosted locally we need a lot of memory.

With these constraints there are a few solutions. First there are exterior SSD this will give us a huge amount of data storage but is quite large and needs a large amount of power to operate. Then there are SD cards, these are very small and low powered. The third being a flash drive, this one is the cheapest per GB and removable but is easily corrupted.

#### 3.3.3.1 Technology Comparison Table:

| Technology | External SSD | Micro SD | Flash Drive |
|---|---|---|---|
| Cost | ↓ | ↑ | ↑ |
| Capacity | ↑ | ↑ | ↑ |
| Power Consumption | ↓ | ↑ | ↑ |
| Retrieval | ↑↑ | - | - |
| Product Dimensions | ↓ | ↑↑ | ↑ |
| Weight | ↓ | ↑↑ | ↑ |
| Longevity | ↑ | ↓ | ↓ |
| Encoding | ↑ | ↓ | - |

Table 3.3.3.1: Memory Technology Comparison Table

#### 3.3.3.2 SSD:

SSDs (Solid State Drives) are storage devices that use flash memory rather than spinning disks like traditional hard drives. This design eliminates moving parts, which means SSDs are faster, more durable, and consume less power. They have quick boot times and silent operation. Though historically more expensive than HDDs, the price of SSDs has been dropping. For our project these were a good starting point to get a feel for the type of memory we needed.

| Part | Vulcan Z | Crucial BX500 | SanDisk SSD PLUS |
|---|---|---|---|
| Cost | $17.99 | $62.80 | $29.99 |
| Capacity | 240 GB | 1 TB | 240GB |
| Retrieval (read speed) | 520 Megabytes Per Second | 540 Megabytes Per Second | 1 Megabytes Per Second |
| Product Dimensions | 3.94 x 2.76 x 0.28 inches | 3.95" x 2.75" x 0.27" inches | 3.96"L x 2.75"W x 0.28"Th |
| Weight | 45g | 2.8 g | 31.75 |
| Encoding (write speed) | 450 | 500 | 530 |
| Link | Vulcan | Crucial | SanDisk |

Table 3.3.3.2: SSD Comparison Table

#### 3.3.3.3 Micro SD:

A microSD card is a small, removable storage device used in mobile phones, cameras, and portable gadgets. It's a smaller version of the SD card allowing it to fit into tight spaces while providing decent storage options. With no moving parts it's durable and energy efficient. Speeds vary by class and type. With higher-end models offering faster data transfers for tasks like 4K video. MicroSD cards come in capacities ranging from a few gigabytes to 1TB. Widely used, but reliant on compatible slots or adapters for broader device use. These seem like the optimal use for our product.

| Part | SanDisk 128GB Ultra | SanDisk 256GB Extreme | TEAMGROUP GO Card |
|---|---|---|---|
| Cost | $14.20 | $25.54 | $16.99 |
| Capacity | 128 GB | 256 GB | 256 GB |
| Retrieval | 140 Megabytes Per Second | 190 Megabytes Per Second | 100 Megabytes Per Second |
| Dimensions | 0.04"D x 0.59"W x 0.43"H | 0.4"L x 0.5"W | 0.04"D x 0.59"W x 0.43"H |
| Encoding | 500 | 130 | 50 |
| Link | SanDisk 128 | SanDisk 256 | TEAMGROUP |

Table 3.3.3.3: MicroSD Comparison Table

**3.3.3.4 Flash drive:**

Flash drives (USBs) or thumb drive are a small, portable storage device that uses flash memory. They connect via a USB port and requires no external power source. These drives are popular due to their ease of use, durability, and ability to store large amounts of data. They are good for quickly moving files between computers or backing up important documents. They are slower than the other options but their ability to be easily removable could add functionality in testing and operations.

| Part | SanDisk 128GB Ultra USB 3.0 Flash Drive | Samsung Type-C™ USB Flash Drive, 256GB | PNY 512GB Turbo Attaché 3 USB 3.2 Flash Drive |
|---|---|---|---|
| Cost | $12.49 | $26.99 | $34.99 |
| capacity | 128 GB | 256 GB | 512 GB |
| Retrieval | 130 Megabytes Per Second | 400 Megabytes Per Second | 100 Megabytes Per Second |
| Dimensions | 0.32"D x 1.33"W x 0.63"H | 0.32"D x 1.33"W x 0.63"H | 0.32"D x 1.33"W x 0.63"H |
| Encoding | 40 | 300 | 100 |
| Link | SanDisk | Samsung | PNY |

Table 3.3.3.4: Flash drive Comparison Table

Figure 3.3.3: Memory Choice

## 3.3.4 DC to DC voltage regulators:

The nodes and gadget will be powered through batteries. Each device will have peripherals that must be powered by a specific voltage to function. We will be considering many different voltage levels, which will be or add up to above the desired voltage. We will introduce a voltage regulating element to solve the issue of having varying voltage inputs to efficiently convert to the required voltage.

### 3.3.4.1 Buck Converters:

Buck converters are step-down voltage regulators for when the output voltage is lower than the input voltage. They operate by modulating the on-off time of the MOSFET and supplying power to the inductor. A buck converter typically consists of an inductor, a switching FET (Field-Effect Transistor) or diode, a capacitor, and an error amplifier with a switching control circuit. The switching waveform is either generated by pulse-width modulation (PWM) or pulse-frequency modulation (PFM). It is then filtered through capacitors or inductors outside the chip to create the output voltage. Noise is usually a concern due to the inductors.

### 3.3.4.2 LDO:

Low dropout voltage regulators (LDO) are linear regulators that turn excess power into heat. They generate a stable output voltage when the difference between input and output voltages remains small. When the difference is at its lowest value is when it is called the dropout voltage. An LDO consists of a voltage reference, error amplifier, feedback voltage divider, series pass element, and a bipolar or CMOS transistor. The designs of LDOs are simple and produce no noise because there is no switching taking place. The loss of power through heat generation makes LDOs perform the worst in efficiency. The increase of the difference between input and output voltages only exacerbates the efficiency problem. Heat generation within our node or gadget must be carefully controlled due to the proximity of other electrical devices.

### 3.3.4.3 Power Management IC's:

Power Management ICs (PMIC) are a compact system-in-a-package solution to voltage conversion, voltage regulation, and battery management. A PMIC can manage multiple power sources and supply multiple loads. Most can mitigate overcurrent and undervoltage situations. Configuration is possible, allowing the user to control various aspects like output voltage, sleep and operating modes, and switching frequency.

#### 3.3.4.4 Technology Comparison Table:

| Technology | PMIC | Buck Converter | LDO |
|---|---|---|---|
| Cost | ↓ | ↑ | ↑ |
| Ease of implementation | ↑↑ | ↓ | ↑ |
| Power efficiency[1] | - | ↑ | ↓ |
| Heat generation[1] | - | ↑ | ↓ |
| Noise[1] | - | ↓ | ↑ |
| Size | ↓ | ↑ | ↑ |
| Versatility | ↑ | ↓ | ↓ |

Table 3.3.4.4: DC to DC Voltage Regulators Comparison Table

1: Power efficiency, heat generation, and noise are dependent on the PMIC's components

PMICs will be used because of their versatility and ease of implementation. There are variants to solve many different functions and can be easily scaled up and down to fit design requirements. Because we want to step down our input voltage to multiple levels, it is easier to achieve this with PMICs. Specifically, we will be looking at PMICs with multiple step-down buck converters because the LDOs are not as efficient and produce heat as a byproduct.

### 3.3.4.5 PMIC Parts:

| Part | TPS65090 | TPS65023 | TPS65270 | TPS65265 |
|---|---|---|---|---|
| Voltage In Range (V) | 6 -17 | 2.5 - 6 | 4.5 - 16 | 4.5 - 17 |
| Voltage Out Range (V) | 1 - 5 | 0.6 - 6 | 0.8 - 15 | 0.6 - 15 |
| Max Current Out (A) | 5 | 1.7 | 3 | 5 |
| Quiescent Current (µA) | 30 - 110 | 85 | 10 | 11.5 |
| Step Down converters | 3 | 3 | 2 | 3 |
| LDOs | 2 | 3 | 0 | 0 |
| Cost | $10.85 / 1 | $3.29 / 1 | $1.96 / 1 | $2.45 / 1 |
| link | TPS65090 | TPS65023 | TPS65270 | TPS65265 |

Table 3.3.4.5: PMIC Parts Comparison Table

The TPS65265 best fits our nodes because it meets the requirements of powering a microprocessor, and many other peripherals at their rated voltage. This PMIC only has buck converters because we want to limit the heat generated in enclosed spaces.

The gadget will need 5 and 3.3 volts to power its router and microprocessor. Stepping down from 7.4 volts is best done through the TPS 65270. It offers two buck converters, which is exactly what the gadget needs, and other features such as overcurrent protection and low-power pulse skipping mode for better light load efficiency. Like the nodes, we will power other peripherals like the LEDs by stepping down using resistors.

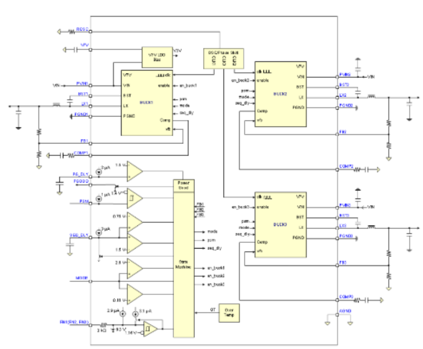


Figure 3.3.4: DC to DC Voltage Regulator Choice

### 3.3.5 Battery:

Our project's power management is one of the most important aspects of the entire project. There are seemingly endless amounts of paths we can take for powering our system, so the research in this section will go very in depth. Our main goal is to reach a max output voltage of at least 5V to reach the minimum voltage required by our parts so far. The most direct path to fulfil this requirement is to find a specific 5V battery that directly connects to our PCB that will direct the power elsewhere. Other types of batteries will need to be regulated by external factors to obtain a desired value.

Our system has multiple endpoints with a central control gadget, therefore, there will be two types of batteries chosen. Our node will need to be able to discharge a considerable amount of milliamperes at a moment's notice. The gadget on the other hand, will be consistently draining power from its active peripherals. Both devices will be powered by rechargeable batteries, so the chemistry composition will likely be similar.

Adding more batteries in parallel is an easy solution to powering our devices for longer. The electric potential will remain unchanged while adding more amp-hour capacity. The tradeoff for increasing the capacity will be an increase in charging time. The current delivered in a parallel system is the sum of each individual battery which decreases the energy used per cell that is added.

Battery voltages are determined by their chemical reactions and the concentration of the components. Nominal battery voltage is found at the equilibrium condition of a battery. Our system will be using rechargeable batteries which will limit the choice of materials. Common types include Lithium-ion, Lithium-ion polymer, Lithium-ion phosphate, and Nickel-metal hydride.

#### 3.3.5.1 Nickel-metal Hydride:

The latter includes multiple drawbacks when in comparison with the other lithium-based batteries. The most important being their performance degradation from overcharging, and a high amount of self-discharge. There is an LSD NiMH version which features a “Low Self-Discharge" at the expense of a small amount of capacity. The constant current draw is handled well by LSD batteries, but they may struggle when a spike in current occurs from activating the node.

#### 3.3.5.2 Lithium-ion Phosphate:

Lithium-ion phosphate is similar to the other lithium-based batteries but also suffers from high amounts of self-discharge. The cathode material is a non-flammable phosphate, so the lifespan is generally longer and safer. Their rugged nature is well suited to the harsh outdoors in both the heat and cold. This type of battery is also best for the environment due to the materials it is composed of.

#### 3.3.5.3 Lithium-ion Polymer:

Lithium-ion polymer batteries use a form of highly conductive polymer in dry gel to store more energy in a smaller size. They are mainly used in laptops and smart phones because of their ability to conform to specific dimensions. A solid electrolyte in a pouch cell is safer because it is stronger and less prone to leakage.  Although they aren’t completely safe because they are flammable if not used properly.

### 3.3.5.4 Lithium-ion:

Standard Lithium-ion batteries are the most common high-capacity rechargeable batteries in many devices including laptops, cell phones, and cameras. The cathode is made from lithium oxide, and the anode is made from carbon. Using this type of battery requires a way to regulate the energy flow to prevent overheating or even combustion. These batteries are also flammable which make them more sensitive in extreme conditions and perform poorly in low temperatures.

### 3.3.5.5 Technology Comparison Table:

| Technology | Lithium-ion | Lithium-ion Polymer | Lithium-ion Phosphate | Nickel-Metal Hydride |
|---|---|---|---|---|
| Cost | ↑↑ | ↑↑ | ↑ | ↓ |
| Max Voltage | ↑↑ | ↑ | ↑↑ | ↓ |
| Energy Capacity | ↑ | ↑ | ↑ | ↓ |
| Self-Discharge[1] | ↑ | ↑ | ↑ | ↓ |
| Charge rate | ↑↑ | ↓ | ↑ | ↓ |
| Lifespan | ↑ | ↓ | ↑↑ | ↓ |
| Safety | ↓ | ↓ | ↑↑ | ↑↑ |
| Environmental | ↑ | ↓ | ↑↑ | ↑ |

Table 3.3.5.5: Battery Comparison Table

1: The self-discharge for NiMH is usually high, but it becomes low in its LSD version.

Our node will face the brunt of any weather conditions that occur. Our batteries must be ruggedized enough to withstand more than just temperate climates. The node has a passive and active mode, so the current must be able to spike at a moment's notice. The battery that best addresses the previously mentioned concerns would be the Lithium Phosphate batteries for the nodes. However, our self-implemented budget constraints render us to choose the next best option which is lithium-ion batteries.

Weather conditions will also affect the performance of the gadget batteries, but to a lesser degree because the gadget is meant to be within the safe zone created by nodes along with the user. There is a larger current draw at a more consistent pace in the gadget than in the node. Lithium-ion batteries are the best fit for the gadget because of their battery capacity to fit the needs of its high current drawing peripherals.

### 3.3.5.6 Lithium-ion Parts:

| Voltage | 1.5V | 3.7V | 9V |
|---|---|---|---|
| Cost / Amount | $31.49 / 4 | $18.98 / 4 | $34 / 8 |
| mAh per battery | 10000 | 9900 | 1300 |
| Batteries required[1] | 4 | 2 | 8 |
| Cycle Life | 1500 | 1000 | 1200 |
| Dimensions per item | 32*60 mm | 67 * 18 * 18 mm | 7.64 x 4.06 x 0.75 inches |
| link | 1.5V Li | 3.7V | 9V Li |

Table 3.3.5.6 Lithium-ion Comparison Table

1: To meet 5 volts and about 10,000 mAh

**3.3.5.7 Lithium-ion Polymer Parts:**

| Voltage | 3.7V | 7.4V | 11.1V |
|---|---|---|---|
| Cost / Amount | $18.49 / 1 | $30 / 1 | $55 / 1 |
| mAh per battery | 10000 | 10000 | 10500 |
| Batteries required[1] | 2 | 1 | 1 |
| Cycle Life | 500 | 300-500 | 300-500 |
| Dimensions per item | 5.16 x 3.27 x 1.26 inches | 3.86 x 1.26 x 3.46 inches | 6.1 x 2 x 1.7 inches |
| Link | 3.7 LiPo | 7.4V LiPo | 11.1V LiPo |

Table 3.3.5.7 Lithium-ion Polymer Comparison Table

1: To meet 5 volts and about 10,000 mAh

**3.3.5.8 Lithium-ion Phosphate Parts:**

| Voltage | 3.2V | 6V | 12.8V |
|---|---|---|---|
| Cost / Amount | $39 / 4 | $35 / 1 | $39.99 |
| mAh per battery | 12800 | 12000 | 10000 |
| Batteries required[1] | 2 | 1 | 1 |
| Cycle Life | 2000 | 2000 | 2000 |
| Dimensions per item | 32 * 70 mm | 100 * 29.8 * 89.5 mm | 5.94 x 2.56 x 3.7 inches |
| Link | 3.2V LiFePO4 | 6V LiFePO4 | 12V LiFePO4 |

Table 3.3.5.8: Lithium-ion Phosphate Comparison Table

1: To meet 5 volts and about 10,000 mAh

**3.3.5.9 Nickel-Metal Hydride Parts:**

| Voltage | 1.2V | 3.4V | 6V |
|---|---|---|---|
| Cost / Amount | $23.33 / 4 | $20 / 2 | $30 / 2 |
| mAh per battery | 10000 | 3600 | 2500 |
| Batteries required[1] | 4 | 6 | 4 |
| Cycle Life | 1000 | 300 | 500 |
| Dimensions per item | 4.5 x 3.2 x 3 inches | 7.13 x 3.07 x 2.32 inches | 3.3 x 2.04 x .67 inches |
| Link | 1.2V NiMH | 3.4V NiMH | 6V NiMH |

Table 3.3.5.9: Nickel-Metal Hydride Comparison Table

1: To meet 5 volts and about 10,000 mAh

Our gadget will use the 3.7V lithium-ion batteries shown in table 3.3.5.6. The main considerations were the high capacitance and low cost. Lithium-ion batteries are very abundant making them much more accessible and cheaper for the average consumer. Its capacitance rivals the best in the market, while having the lowest costs.

The best batteries for weather storms and inclement weather are the lithium phosphate options. They are better for the environment and have the best lifespan and durability, however the costs of powering three separate nodes make them less than ideal. The next best option that will be used is the same batteries in the gadget.


Figure 3.3.5: Battery Choice

## 3.3.6 MCU:

The MCU in the gadget must have a high data throughput to process the imagery coming from six cameras. To achieve this goal there needs to be a powerful processor and enough RAM to perform the intensive threat detection code. Another benefit to having more processing power is the data transfer rate to our storage device. The microprocessor must be able to quickly free its ram to move on to the task.

### 3.3.6.1 ESP32-S3:

The ESP32-S3 is a powerful microcontroller with enhanced AI and machine learning capabilities. On top of that it is relatively cheap for the functionality it provides. It features a dual-core Xtensa LX7 processor that can run at up to 240 MHz. The chip has 512 KB of internal SRAM and supports external flash and PSRAM for more memory. The on-board Rams gives us the capability to do most of the processing in each node locally. It also comes with 2.4 GHz Wi-Fi and Bluetooth 5.0, allowing for wireless communication in various IoT applications, perfect for our wireless LAN network. The ESP32-S3 supports hardware acceleration for vector instructions, making it ideal for tasks like neural network inference. It has plenty of GPIO pins, ADCs. And a variety of peripheral interfaces like SPI, I2C, and UART. Additionally, it includes secure boot and flash encryption for improved security. All this combined with its low power footprint and ability to go into low power modes make the ESP32-S one of our top picks.

### 3.3.6.2 STM32W:

The STM32W series microcontrollers offer a well-balanced combination of performance and power efficiency, particularly suited for wireless applications. With an Arm Cortex-M33 core running at 100 MHz, it supports multiple wireless protocols. It has a particularly low CoreMark score of 407, but it’s lower processing power also lowers its energy consumption, boasting a minimum power consumption of just 1.25 µA in standby mode while retaining 64 KB of RAM active. There is 1 MB of integrated flash and 128 KB of RAM, and up to 35 GPIO pins. While lacking in a few areas, this MCU provides a better low-power profile making it ideal for battery-operated devices.

#### 3.3.6.3 PIC32MZXW1:

The PIC32MZXW1 has an M-Class core running at 200 MHz, making it more suitable for applications that require speed and reliability. The memory is broken down into 2 MB of embedded flash, 640 KB of RAM, which includes 512 KB allocated for data and 128 KB for Wi-Fi buffering. Its CoreMark score of 710 is great for complex calculations like our image processing. It includes advanced wireless features like a codeless WLAN development framework and has its power-saving modes consume only 2.7 mA in wireless sleep mode (WSM). With up to 37 GPIO pins, the PIC32MZXW1 is a versatile choice capable of handling our projects many peripherals.

#### 3.3.6.4 CC3220:

The CC3220 from Texas Instruments is an MCU designed with wireless connectivity at its core, perfect for our applications which prioritize fast and reliable Wi-Fi communication. It features a dual core Arm Cortex-M4 running at 80 MHz which is slower than every other option by a significant margin. However, it is tailored specifically for Wi-Fi applications which are enhanced through SimpleLink technology, providing autonomous quick connections. The throughput ranges from 13 Mbps to 72 Mbps over TCP and UDP. Memory of the CC3220 includes 256 KB of RAM and optional 1 MB of executable flash. which overall makes this MCU optimized for efficient execution of wireless applications. While this MCU meets the minimum requirements for our project, it is extremely specialized and sacrifices too much processing power for wireless connectivity.

#### 3.3.6.5 Technology Comparison Table:

| Technology | ESP32-S3XX | STM32WXX | PIC32MZXW1 | CC3220X |
|---|---|---|---|---|
| Cost | ↓ | ↑↑ | ↑↑ | ↑ |
| Processor | ↑↑ | ↓ | ↑ | ↓↓ |
| Memory | ↑↑ | ↓ | ↑ | ↓ |
| Power Consumption | ↑ | ↑↑ | ↓ | ↑ |
| Architecture Familiarity | ↑↑ | ↓ | ↓ | ↓ |
| Pin Count | ↑ | ↑↑ | ↑↑ | ↓ |

Table 3.3.6.5: MCU Comparison Table

Our gadget requires a very high throughput of data because of our image processing capabilities. Therefore, our processing power and memory are very highly valued. The product that would best fit this role is the ESP32-S3 series.

**3.3.6.6 ESP32-S3 Parts:**

| Part | S3R8 | S3FN8 | S3R16V |
|---|---|---|---|
| Flash (MB) | 0 | 8 | 0 |
| PSRAM (MB) | 8 | 0 | 16 |
| Supply Voltage Min (V) | 3 | 3 | 1.8 |
| Cost | $2.28 | $2.28 | $2.49 |
| link | S3R8 | S3FN8 | S3R16V |

Table 3.3.6.6: ESP32-S3 Comparison Table

There is minimal variance when selecting a specific version of the ESP32-S3 SoC. When the nomenclature ends in V, there is only a 1.8V external SPI flash. The letter F and R denote the usage of flash and PSRAM respectively. Pin mapping is also slightly different due to Quad versus Octal SPI modes. The deciding factor came down to the memory, and the best choice available for our gadget is the S3R16V.


Figure 3.3.6: MCU Choice

## 3.3.7 LED:

Light Emitting Diodes (LED) are nonlinear semiconductor-based components that can’t be connected directly to a power source. LEDs are like other diodes in that the current will flow in one direction. Because of this, polarity and the components orientation must be carefully considered for the unit to work. LEDs compared to traditional light sources have better power efficiency where less power is lost as heat generation. Another big difference is the response time of an LED is instant, while older lightbulbs tend to fade on and off. Color variance is a lot more common in an LED because they produce one wavelength while a normal light bulb would need a color filter like dyed glass. Our project will need a small luminating device, and LEDs are our only realistic option. Miniature bulbs are quite expensive and fragile, which won't be useful for our ruggedized system.

Different colors require different wavelengths to be produced. The wavelength of an LED is determined by the specific semiconductor materials used. These wavelengths have their own typical forward voltage. It is expected that ultraviolet light is more voltage intensive than infrared light emission because the device must commit more energy to increase the frequency which shortens the wavelength. The same reasoning is applied to the visible spectrum where white and blue lights which have a greater frequency will require more voltage than a red light. There is an Ingress Protection (IP) rating system that applies to many LED light fixtures. It is recommended for outdoor LEDs to use a minimum IP65 rating for complete protection against dust and water jets from any direction. In this rating system it is important to know that the last digit represents the degree of water protection and that higher numbers equate to a greater degree of protection.

#### 3.3.7.1 Through Hole LEDs:

Through-hole (THT) LEDs function by connecting physical leads on the opposite side of a PCB by passing through a premade hole. The LED consists of a chip protected in an epoxy resin case and two long leads. Holes in the PCB are needed, which could constrain the board layout. They require soldering at the end of the diode after the pins go through holes of the PCB. This creates a secure bond capable of handling more stressful situations. Replacing components is much easier, allowing for adjustments in the design process. Main applications of THT LEDs are for when durability and reliability is paramount.

#### 3.3.7.2 Surface Mount LEDs:

Surface mounted (SMD) diodes are soldered directly to the surface of a PCB which decreases the size of the component. Because of their size they are generally more difficult to solder in place. Atop the solder which connects to the PCB board is the SMD frame. Inside the frame is a chip with gold wiring encapsulated by a phosphor substance. The frame does not allow light to pass through and acts like a cup around the chip which limits the light emitting angle. They are the most suitable for environments with limited space. They boast low power consumption, and thus have less heat generation and a longer lifespan than other LEDs.

#### 3.3.7.3 Chip-on-Board LEDs:

Chip on Board (COB) LEDs have multiple LED chips directly mounted on a single substrate or board. The composition includes the chip on the PCB board encapsulated by a soft substance capable of withstanding deformations. There isn’t any frame to prevent light like in the SMD LEDs, which makes the light emitting angle much greater. The many chips on a small area produce a powerful and bright light. The best use case is for consistent and high luminosity and light uniformity. The close arrangement of LED chips creates an even distribution of light. COB LEDs are commonly found to be sold as long strips with very few other uses. Considering our projects gadget and nodes, there isn’t a strong necessity to have a uniform and bright light. For both types of devices in our project we will use LEDs primarily to denote on/off statuses. The only use case I can assume this type of LED providing for in our project is to light the surroundings of the nodes. The strips of COB LEDs are designed to bend to multiple surfaces allowing for versatile installation options. Something to consider is the typical voltage requirement is much higher than the other options. They are mostly found to be at 5 volts at the minimum, and commonly found to be higher too.

### 3.3.7.4 Technology Comparison Table:

| Technology | THT | SMD | COB |
|---|---|---|---|
| Cost | ↑ | ↓ | ↑↑ |
| Size | ↑ | ↑↑ | ↓ |
| Ease of Usage | ↑↑ | ↓ | ↑ |
| Energy Efficiency | ↓ | ↑ | ↑↑ |
| Brightness | ↓ | ↑ | ↑↑ |
| Lifespan | ↓ | ↑ | ↑↑ |
| Durability | ↑↑ | ↓ | ↑ |

Table 3.3.7.4: LED Comparison Table

Lighting in our gadget will indicate whether the device and its peripherals are being powered. THT LEDs are preferred due to their ease of implementation and replacement. While still being mounted on a PCB, they extend further to enhance light distribution. It is also easier to design the housing of the devices around an LED that sticks out more.

To illuminate the surroundings of our nodes, COB LEDs are the optimal choice due to their superior brightness. They can be easily implemented as peripherals without direct attachment to the PCB. THT LEDs within the node will be used for the same purposes as and reasons mentioned above. COB LEDs also excel in their waterproofing availability, while it is harder to find this feature on the other variants.

### 3.3.7.5 Chip-on-Board LED Parts:

| Part | KXZM | Samsion | CHNMALITAI |
|---|---|---|---|
| Voltage (V) | 5 | 5 | 5 |
| Wattage per meter (W/m) | 5 | 6 | 5 |
| Length (m) | 2 | 1 | 1 |
| LEDs | 640 | 320 | 320 |
| Cost / Per | $11 | $7 | $10 |
| IP rating (waterproof) | 65 | 67 | 65 |
| CRI index | 93 | 85 | 90 |
| link | KXZM | Samsion | CHNMALTITAI |

Table 3.3.7.5: Chip-on-Board LED Comparison Table

Our nodes will perform best with Samsion brand COB LEDs. The key consideration was their waterproof rating. Samsion offers the most cost-effective option without compromising performance. Although the CRI index is the lowest, this does not affect quality since they will be used outdoors where it is less critical.

### 3.3.7.6 Through Hole LED Parts:

| Part | CHANZON | Novelty Place | WOTOFULIN |
|---|---|---|---|
| Cost | $5 | $6 | $7.29 |
| Quantity | 100 | 100 | 100 |
| Voltage (V) | 2 – 2.2 | 2 – 2.2 | 2 – 2.4 |
| Current (mA) | 20 | 20 | 20 |
| Brightness (mcd) | 600-800 | 3000-4000 | 8000-12000 |
| Light Beam Angle (Deg) | 20 | 30 | 20-30 |
| link | Chanzon | Novelty Place | WOTOFULIN |

Table 3.3.7.6: Through-Hole LED Comparison Table

Chanzon LEDs are the best choice for minimal outdoor lighting. They offer the lowest costs and brightness, making them ideal for nighttime use due to reduced light pollution. The beam angle is not that important due to the fact we will be wrapping the strip around the node which will cover all angles.

### 3.3.7.7 Surface Mount LED Parts:

| Part | UXCELL | AEDIKO | CHANZON |
|---|---|---|---|
| Voltage (V) | 2-3 | 3-3.2 | 3-3.2 |
| Current (mA) | 15-20 | 20 | 20 |
| Style | 0603 | 0805 | 1206 |
| Brightness (mcd) | 300 | 210 | 320 |
| Cost / Per | $7.69 | $4.99 | $6.99 |
| Dimensions | 0.03 x 0.03 x 0.06 inches | 0.05 x 0.05 x 0.08 inches | 0.06 x 0.06 x 0.12 inches |
| Color Variety | Blue | White | White |
| Color Temperature (K) | - | 9000 | 6000-9000 |
| link | UXCELL | AEDIKO | CHANZON |

Table 3.3.7.7: Surface Mount LED Comparison Table

If we were to choose to use surface mount LEDS we wouldn't care about waterproofing as the LEDS would remain inside of the gadget with a protective see through material. The best choice from the table above is the AEDIKO LEDs because of their small size and lower cost. The UXCELL would be a better option if we wanted to use blue LEDs, but the white variant was not available. The difficulty difference in installation versus the other types make the SMD LEDs not a preferrable choice.


Figure 3.3.7: LED Choice

## 3.3.8 Alarm:

Audio hardware is an extremely dense topic, and we will investigate a few parts that will be beneficial to implement. Potentiometers are very useful in audio equipment. The benefits include frequency attenuation, and volume control. The taper in a potentiometer refers to how the resistance varies as the analog controlling device is rotated or as the wiper slides. A linear taper varies the resistance in an even manner, while the logarithmic taper slowly varies resistance on the low end and rapidly toward the high end. Audio tapers are logarithmic because our ears are sensitive to changes at lower volumes.

### 3.3.8.1 Speaker:

Speakers are the most common device used in audio, along with a supporting amplifier to increase the volume and improve sound quality. They are generally louder and have more capability to produce a variety of sounds. Sound is created by sending current through coils of wire to create an electromagnetic field. Audio signals travel through the voice coil. The coils repel and attract with a permanent magnet, which displaces the air from its alternating motion and that produces the sound that we hear.

### 3.3.8.2 Piezo Buzzer:

Named after the piezoelectric materials that comprise them, piezo buzzers provide auditory cues to the end user. Their materials offer a unique ability to convert energy between mechanical and electrical forces. The piezoceramic disk converts electric current into sound producing mechanical movement through vibration.

### 3.3.8.3 Magnetic Buzzer:

Magnetic Buzzers differ from piezo buzzers in their components and process of converting current to sound. At the center is a ferromagnetic disk which is a flexible magnetic disk. The flexibility allows for vibration to produce sound. The current is run through a wire coil which creates an electromagnetic field. When the field is activated, the disk bends towards the field, and returns to a normal state when there is no current. The buzzer creates sound when rapidly alternating the current between on and off.

### 3.3.8.4 Technology Comparison Table:

| Technology | Speaker | Piezo Buzzer | Magnetic Buzzer |
|---|---|---|---|
| Cost | ↓ | ↑ | ↑↑ |
| Operating Voltage | ↓ | ↑ | ↑↑ |
| Power Consumption | ↓ | ↑↑ | ↑ |
| Sound Pressure Level (dB) | ↑↑ | ↑ | ↓ |
| Frequency Response | ↑↑ | ↑ | ↓ |
| Size | ↓ | ↑ | ↑↑ |

Table 3.3.8.4: Alarm Comparison Table

For our needs we do not need all the functions of a speaker, but we still want to produce a loud sound. Piezo buzzers are the second loudest option while staying simple in design. We won't need to add an amplifier for either of the buzzers, and the power consumption is the lowest for the Piezo buzzer.

### 3.3.8.5 Speaker:

| Technology | WEICHUANG | BNYZWOT | Jameco |
|---|---|---|---|
| Cost / Per | $5.49 / 3 | $8.39 / 10 | $7.90 / 2 |
| Range of Voltage (V) | 3 - 24 | 3 - 24 | 4 - 16 |
| Rated Voltage (V) | 12 | 12 | 12 |
| Rated Current (mA) | 15 | 12 | 25 |
| Sound Output (dB) | 100 | 85 | 95 |
| Frequency Response (Hz) | 3300 ± 500 | 3300 ± 500 | 2900 |
| link | WEICHUANG | BNYZWOT | Jameco |

Table 3.3.8.5: Speaker Comparison Table

The WEICHUANG Piezo buzzer is the best option due to its highest decibel output, which is crucial for our needs. It is also the most cost-effective option and comes in the exact quantity required for our project. The rated current may not be the best option for conserving power, but the alarm will be active for a small portion of the battery's life.

Figure 3.3.8: Buzzer Choice

# Chapter 4 Standards and Design Constraints

## 4.1 Introduction:

In the development of our project, adhering to established standards and properly identifying design constraints is essential to ensure the reliability and functionality of any system. This chapter outlines the key standards and design constraints that influenced the development of our deployable security system. These standards not only guide the technical aspects of our project but ensure they operate effectively within the physical, environmental, and ethical boundaries of its intended use.

Considering our project's use as a security system designed for use in outdoor environments, it requires a more careful consideration of both resilience and usability. This helps us prioritize certain goals for our project such as providing efficiency and reliability while being able to withstand the elements. Throughout our design process this far we have been mindful of the standards related to resilience, user-centric design, and environmental sustainability.

This first chapter will explore relevant standards our project must meet, such as resilience against tampering and maintaining performance in varying weather conditions. Then moving onto design constraints which explain the obstacles that have impacted the system's development, such as material selection, power efficiency, memory management, and environmental factors. Analysis of these standards and constraints have informed our decisions on how to optimize the system for long-term functionality and ease of use. We aim to deliver a product that not only meets the needs of our users but also aligns with the best practices in security, design, and environmental responsibility.

## 4.2 Relevant Standards:

Some standards that our project is going to have to meet are resilience towards attacks on our system and ensure we have recovery mechanisms in such a case. Another standard that our system wishes to meet is having a user-centric design. Through meeting these standards, we wish to provide a reliable system that our customers will feel confident in, while also providing ease of use and abstracting any intellectual requirement from operating the system on the user side.

When designing our system, standards were not the first thing that popped into our minds, so it was something that we really had to put some thought into as far as how we wished to accomplish our goals. Reliability is one of, if not the most important aspects of a security system. If you are not confident in your security system to accomplish its sole purpose, then it is useless. We plan to gain trust of our customers by providing reliable and trustworthy product through our security methods such as anti-tampering, providing durable physical product to our users, and through thoroughly testing our product to identify any potential security breaches our system may have. Anti-tampering plays a pivotal role in our system's resilience. We plan to accomplish this feature through both hardware and software efforts. For example, if per say somehow someone walks up to one of our nodes without the passive IR sensor and ESP32-CAM detecting a person and alerting our system. Their next goal will more than likely be either to destroy one of our nodes or get inside of our secured area. Measures we have against getting inside of the area are a different topic, but for a short countermeasure we will be having an active IR sensor that will continuously monitor the perimeter. Measures we have against node tampering consist of software running inside of our network that will monitor the "Heartbeat" of each node inside of our network. We will be setting up alerts and notifications that will be sent out to the gadget and our mobile application, that when the system detects any unauthorized changes or if a component goes offline unexpectedly, our alarm will be triggered classifying this as a security breach, hence alerting the user through notifications and our physical alarm on each node. The software that will perform our heartbeat functionality will live inside of main.ino on each node. The goal will be to monitor, just like how we monitor the passive IR sensor continuously, and if the heartbeat is ever not detected then we will consider that node tampered with. We then will send a message over HTTP to the gadget. Once the gadget has received this message it will then Alarm the other nodes and the user via our alarm system. The alarm system will be set up as follows, whenever a threat has been detected, by either identifying a person from our image processing algorithms, the active IR sensor being tripped, or by a node being tampered with. The Gadget will send messages to all "alive" nodes and from there the nodes will sound an alarm contained in the housing of the nodes.

Another feature of our system resilience is what our physical hardware will be made of. This is a feature that does not play a role in the efficiency of detecting people or motion outside of our secure area, however it will allow us peace of mind with the fact that if our system is put in place in the real world, it will not crumble under pressure if there is a little wind or rain and completely fail. We plan for the physical pole that will hold up our "Nodes" to be roughly 5-8ft above the ground and they will be made from pvc pipes. Pvc pipes are resistant to corrosion and rust, which is perfect for outdoor usage in the rain. Their hollow interior increases spatial scalability because they will be insulated and protect the electronics we may want to implement inside. LEDs and other peripherals can have a ingress protection rating (IP) which will allow us to meet a variety of reliability standards such as being waterproof, durable, and able to withstand wind as we plan to implement ground spikes or to carve the bottom of the pvc pipe to ensure our pole does not get lifted out of the ground unintentionally unless we have a hurricane rolling into town. As far as our ESP32 housing goes, we plan to make that water resistant as well. We will be building custom housing for our ESP32-CAM nodes that will fit perfectly to our system requirements. Once we have fully integrated and developed all aspects of our security system, to ensure we can guarantee our customers' reliability under circumstances, we plan to thoroughly test our product with multiple test plans. For example, one of our tests will consist of a waterproof test where we will provide a rainlike environment to ensure the system is capable of enduring moderate weather as this product is geared towards the outdoors and should not have to be picked up every time it starts sprinkling outside. We will go over more testing in chapter 9, however rest assured that our system will be thoroughly tested to meet reliability under circumstances that our product could encounter once deployed.

Our second standard is based on our product having a user-centric design. For any product sold to consumers, it is common practice for there to be little to no hard effort to deploy a system that you have bought, hence why we are making this a standard for our project. We plan to accomplish this in a couple of ways. First, we wish to be able to set up and deploy our system in the field relatively quickly and easily. Simply, turn on the gadget, connect everything to the router by flipping a switch on the gadget, and then place your nodes in the correct formation to where the active sensors are at least pointing at each respective pole. Another feature we will be adding to our system design that ensures we keep this a user-friendly product is we will be developing an interactive mobile app. This new feature will allow users to not be physically present at the secured perimeter and will set up a communication network between the router and your phone. Through this app you will receive notifications if your secured area alarm has been triggered, and even can request for your cameras to take a snapshot of the perimeter and you will see 6 different captured images of the perimeter for the area on your phone. Another feature we have integrated into the design of our system that allows for a user-centric design is through the functionality of the gadgets. We will have multiple switches on the gadget that will allow users to interact with the system such as turning the power on, a switch to flip that will capture and send a message to the nodes and an image will be captured and sent to the mobile app, and then a switch that will turn the alarms on if a user has noticed any activity outside the area that hasn't already tripped by the alarms, and the last switch on the gadget will turn the alarms off if a threat has been handled and the user no longer wishes for the alarm system to be triggered. Keeping the user in mind while designing and developing a product is key to success with anything you create, and we believe that with these features we are developing we have met the requirements to meet the standard for our product by having a user-centric design.

[44] One of the key institutions we have benchmarked ourselves against was IEEE. The Institute of Electrical and Electronics Engineers (IEEE) is a global professional organization dedicated to advancing technology across multiple fields. Founded in 1963, it has grown to become one of the largest organizations of its kind. Its mission focuses on fostering innovation and education, including the development of technical standards. These standards ensure that technology is safe, reliable, and compatible across different platforms and industries worldwide.

IEEE is renowned for its role in creating standards that are followed across various sectors, such as telecommunications, information technology, and power systems. These standards are crucial because they provide guidelines for manufacturers and engineers, enabling the design of products that are interoperable with other systems. Additionally, they ensure the safety and efficiency of electrical systems, helping prevent accidents and equipment failure.

One focus of IEEE’s standards is low-voltage electrical systems, which typically operate under 1000 volts and are commonly used in residential, commercial, and industrial settings. The IEEE standards for these systems promote safe, energy-efficient designs while ensuring compliance with regulations. By providing installation guidelines for equipment like wiring, circuit breakers, and protective devices, they help mitigate risks such as electrical fires or shocks.

The IEEE 3189 standard is one example of their contributions, concentrating on energy-efficient solutions for computing devices. It encourages the design of electronics that minimize energy consumption, a growing concern as digital technology expands. This is particularly significant given the rising global demand for energy as more devices enter daily use.

Another prominent standard is the IEEE 802 series, which governs local area networks (LAN) and metropolitan area networks (MAN), including technologies like Wi-Fi and Ethernet. IEEE 802 serves as the foundation for many modern networking systems, ensuring that devices from different manufacturers can communicate seamlessly. This enhances compatibility and performance across networks, benefiting consumers and businesses alike.

The IEEE 1012 standard addresses software verification and validation. It offers a structured approach to ensuring that software is developed and maintained according to best practices. This standard outlines processes for planning, executing, and documenting software tests, which are critical in industries such as aerospace, medical devices, and automotive systems, where software reliability directly impacts safety and performance.

[44] Ingress Protection (IP) standards are another critical area of classification, used to assess the level of protection a device offers against solids and liquids. These ratings are vital for ensuring that electrical devices are safe in different environments. The ratings are represented by two numbers, such as IP96, where the first number indicates protection against solids like dust, and the second number shows resistance to liquids like water. The higher the number, the stronger the protection.

The IP standards emerged from the need for a consistent way to measure and communicate device durability under various conditions. Before the IP system, manufacturers often used their own protection ratings, leading to confusion and difficulty in product comparison. In 1976, the International Electrotechnical Commission (IEC) introduced the IP standard through the IEC 60529 classification system to provide clear and universal guidelines for manufacturers to follow.

The purpose of the IP system is to help both consumers and professionals understand how well a device withstands dust, moisture, and other environmental factors. For instance, a device rated IP68 offers full protection from dust (indicated by the "6") and can be submerged in water for prolonged periods (indicated by the "8"). These standards ensure that devices intended for harsh environments, like outdoor cameras or industrial machinery, continue to function safely and reliably.

IP standards are crucial in many industries, from consumer electronics to construction and healthcare. They enable manufacturers to design products with appropriate protection for their intended environments. For example, smartphones often carry ratings like IP67 or IP68, signifying they are dust-tight and resistant to water immersion. In industrial applications, IP ratings help ensure that equipment can endure exposure to dust and water, promoting worker safety and reducing the likelihood of equipment failure.

## 4.3 Design Constraints:

Designing an outdoor electronic device comes with multiple challenges. One of the biggest constraints is weatherproofing. Since the device will be outside and needs to be protected from rain and humidity. This means using materials that are resistant to water. Such as plastics and silicone-based seals. Plastic cases with rubber/silicone gaskets will help keep water out and prevent damage to the internal electronics. Also, the materials should be able to withstand extreme temperatures, without warping or breaking down.

Preventing water damage isn't just about protecting the outside. The inside of the device must also be carefully designed. Any opening of wires or ventilation can be a potential point for water to get in. So, these areas need extra protection. We have considered adding coatings or sealants to internal components to provide more water resistance but that is outside the scope of a prototype. And with moisture comes corrosion. This is another problem, so choosing corrosion-resistant materials is important. To avoid internal moisture, we are also looking at desiccants and other moisture reducing materials. Some parts in the design have specific waterproof rating called ingress protection or IP for short. It is a protection measurement rating system against solids like dust and dirt as well as liquids. Other peripherals also have IP ratings like buzzers; however, they aren't necessary with a proper housing design. It is more beneficial to invest in LEDs with an IP rating because they are more subject to the outside environment. They need to be more present and illuminate outside the safety of our housing units.

The interior of our housing units for the nodes will be accounting for stormy weather by allowing the rain to runoff away from our electrical parts. The goal is to never allow water inside of the housing and we will achieve this by creating an elevated center and the edges will extend further out. The runoff will fall off the edges without entering the device's confines. The gadget's housing will also be designed to work in the rain by sealing the exterior. Along the precipice of the exterior will be through-hole LEDs that are protected by a plastic casing, and the connections to the housing will be filled with a waterproof sealant. The charging ports on both the nodes and gadget will be protected as well. The housing design will include access points to charge the devices and will close off the access points for when in an operational mode.

Safety is another big factor in outdoor designs. The product needs to be electrically safe, especially if it's exposed to moisture. It must be properly grounded to prevent electrical shocks or short circuits. Considerations have been made to add grounding strips to the spike. But with low voltage and amperage this is a marginal concern. Additionally, the device must be secured so it doesn't pose a hazard if it falls. The product should also meet local electrical safety standards, and care must be taken with wiring.

The ESP32 is a powerful microcontroller with a lot of functionality. But its Wi-Fi capabilities are limited to the 2.4 GHz band. This creates certain design constraints. One key issue is interference because many other devices, like Bluetooth and appliances that operate on the 2.4 GHz frequency. This can cause signal congestion, leading to slower data transmission and unreliable connections, especially in environments with many competing devices.

Another constraint in the 2.4 GHz is speed. The band has lower data transfer rates compared to higher frequencies like 5 GHz. This can be a problem if your network needs to handle a lot of data quickly, such as streaming video. With our product having video streaming capability, this will be a hurdle for us. But the 2.4 GHz band has better range and can penetrate walls more effectively.

Additionally, the ESP32's use of the 2.4 GHz band affects network scalability. Since it's a shared frequency, the more devices you add, the more likely the network performance will degrade. We have to carefully plan how many devices connect to the network to avoid bottlenecks. These design constraints require trade-offs between range, speed, and the number of devices to maintain a stable network. This is one of our largest constraints

One of the primary design constraints for our deployable security system is the efficient management of memory allocation for storing and processing images captured by the ESP32-CAM nodes. The ESP32-CAM has limited onboard memory, which presents a challenge when dealing with high-resolution images or video streams. Given that the system needs to store multiple images for analysis, including those from motion detection events, we must carefully manage how memory is allocated and used.

The limited onboard RAM of the ESP32-CAM restricts the size and number of images that can be stored locally at any given time. This constraint requires us to use memory-efficient image formats, such as JPEG compression, to reduce the file size while maintaining enough quality for threat detection. Additionally, the image processing tasks performed on the ESP32-S3, such as identifying potential threats using machine learning algorithms, require sufficient memory to run effectively. Balancing the storage of raw images and processing capabilities is crucial to avoid performance bottlenecks.

Another constraint arises from the need to store images long-term for review and analysis. Since the system is designed to operate in remote environments, we have to account for the limited storage available on the ESP32-CAM itself. One potential solution is to periodically offload images to external storage or cloud services, but this introduces additional power and network constraints, especially if operating in areas with limited or no internet connectivity. Therefore, efficient memory management strategies are essential to ensure the system can store, process, and transmit images without running out of memory or compromising system performance.

### 4.3.1 Time Constraints:

It is essential that our system reliably performs its duty to ensure the safety and security of the environment. In a real-time system like our own, a desirable response is degraded by the delay of its arrival. Depending on the magnitude of the delay, it could be just as hazardous as an incorrect response. The node works in a low-power mode when idle and must be capable of running at full capacity at a moment's notice.

Performance is an important non-functional requirement in our real-time system. In contrast with other real-time systems, response time is not necessarily important to protect the system but to inform the user of their surroundings. Another aspect of a performance requirement is the systems throughput. More specifically, it is crucial when our system is handling imagery from capturing at the node to processing on our gadget and finally alerting the user of what was detected.

Threat detection in our image processing is our greatest timing concern. There are potentially six cameras sending imagery to be processed at the same time over the same network.

Factors outside of our system can also affect the response time. A major selling point of our system is that it can be deployed just about anywhere the environment allows. Both extreme hot and cold temperatures will affect the performance and by proxy the response time. Obstructions near the devices may cause interference when transmitting data over the Wi-Fi network.

Our solution to this constraint is combatted by our software architecture. The architecture has been designed to be able to wake up the entire MCU within a split second, capture an image, send that to the gadget, and process the image on the gadget quickly. We are accomplishing this through multiple ways. The first way we are doing this is by the selection of language for our software. We have chosen C++ for numerous reasons, however, the main being that with C++ we can have the most control of what our system does as possible. For example, with C++ we can take advantage of manipulating memory directly, giving us access to how much memory we use and ensure that no time is wasted in that process of allocating/deallocating too much or too little. Another way we will be saving time is in our image processing algorithm, which is the most time-consuming program our system runs. Most programs that are in python, although a lot easier to implement, take up quite a bit of resources and time that our system simply does not have. Through the use of C++ and simply the libraries such as TensorFlow Lite that we will be using, we will be using far less time and resources to accomplish person detection.

### 4.3.2 Part Constraints:

Our product must be rugged and mobile, requiring parts to be lightweight, low power, and durable enough for constant use and abuse. With the main use case being outside we needed to prepare for rain, dirt and other hazards to electrical systems. Given time and a proper R&D cycle we would aim for an IP67 rating to guarantee the safety of our equipment. But for this prototype we will not be aiming for any IP rating and will not be testing using water or dust.

A key parameter we considered was the use of self-contained systems. For example, we chose the ESP32-CAM as the camera system. It is low power, lightweight, and only requires physical integration for power. Software integration is handled internally or over our local wireless LAN to the gadget and the locally hosted website.

Power consumption is another priority. We aim for at least 24 hours of operation between charges, which requires over 10,000mAh for the nodes and 15,000mAh for the gadget. Lithium-ion packs are the best solution due to their flexibility in size and shape, fitting well into our design. They also operate within the voltage range of common phone chargers (6-12V), allowing us to use micro-USB for easy charging.

The main challenge will be ruggedizing the system. Ruggedizing the PCBs and ESP32-CAMs is beyond the scope of this project. For consumer-grade products, we would use higher-grade PCBs and water-resistant casings, but this is too costly for a prototype. To mitigate potential issues, we're using low-profile components on the PCB to reduce shake and moisture damage. The casing will be rigid to protect the battery pack from impacts.

### 4.3.3 Safety Constraints:

With any product a good Indepth safety review is a must. This allows designers to look at their products from a different angle and design safety features as well as catch problems before development is in effect. Saving time and money but more importantly making sure their product doesn't harm anyone.

Our product will have a few pieces of equipment that could be harmful. The main concerns are the Infrared sensors, the battery packs, and the alarm system. We will be using OSHA's Technical Manual for our safety standards.

Infrared emitters can be harmful to the eyes. This is compounded by the fact that infrared is invisible to the human eye. For our first idea for our active IR trip wire, we planned on using a class I laser as the emitter to gain distance. Chapter 6, section III of OSHA's regulations covers lasers and their hazards and states that "Class I: cannot emit laser radiation at known hazard levels (typically continuous wave: cw 0.4 µW at visible wavelengths). Users of Class I laser products are generally exempt from radiation hazard controls during operation and maintenance (but not necessarily during service)." While this is a good thing, we still need to be careful with power generation and heat generation. For our prototype we have opted for an IR LED instead of the class I laser due to its lower energy/heat cost as well as a wide beam width compared to the laser giving us more room to work with. In future iterations of our product a laser could be implemented to give us a significant range increase.

Our batteries onboard have a need for significant power compacity. This has pushed us to use lithium as our battery of choice. Lithium provides several safety concerns. Lithium Batteries are not impact resistant and when damaged can experience a thermal discharge resulting in fire and even detonation of the battery pack. There are also temperature concerns with Lithium; battery packs can be damaged with temperatures over 130 F and below 32 F. Both are addressed inside our casing. We have opted for a nonremovable battery pack to mitigate these problems. With the PVC pipe protecting the pack and light insulation protecting the temperature, the battery pack shouldn't experience any environmental dangers. On future iterations, upgrading the battery to a more stable solution will be to meet the demands of harsher climates.

For the alarm system we need to be able to scare off any perceived threat to the area, but we also need to stay within the correct limits. OSHA standard 1910.95 is the Occupational noise exposure. We used this as guidance for our noise level. As we designed for 80 dB max level this is lower than the lowest level, they state placing us in a very safe range as far as noise is concerned.

### 4.3.4 Environmental Constraints:

The outdoors introduces many new considerations when creating an electrical system. Our goal is to have full functionality of every part of our system in temperate climates. The housing unit for our node must be able to withstand a moderate amount of rain at the very least. Considering our target demographic, we must provide a product that can be easily transported and deployed in multiple terrain types. The PVC slant at the bottom of the nodes will not penetrate harder substances like rock or concrete. This prohibits practical usage in most urban and indoor environments. Part of our project's functionality requires sensors to properly line up, thus the nodes must be planted in the ground or held by external devices.

Factors such as temperature are very important to consider when working with electrical systems. Thorough research for every single part was completed to find that the optimal temperature range is between -10 and 50 degrees Celsius. The operating range is limited due to the batteries. When operating outside of the range, it is expected to have reduced charge capacity, and a general sense of degradation in the performance of the whole system. It is advised by REDARC Electronics to charge lithium-ion batteries in an ambient temperature over 0 degrees Celsius. Below freezing has adverse effects like reducing the lifespan and performance of the battery.

Environmental conservation is a major consideration in our project's functionality. Our project will have minimal disruption to the wildlife when it is deployed. There will be a slight amount of displacement in the dirt when placing the nodes in the ground. Upon completion and dismantling of our system, that displacement can be immediately filled manually or over time naturally. According to the National Wildlife Federation, "The moment the ecosystem begins adjusting to one stress, another appears. [4]" Our product may cause stress amongst the wildlife which could affect the ecosystem of that area. This is an unavoidable issue because we must prioritize our user's safety. The deployment of our system should therefore be discouraged from being used for long periods of time. The disruptions caused by our system include a loud alarming noise, and flashing LEDs to alert the user and keep wildlife away. Our gadget was designed in a way to create no disturbance to the environment. Our system from its deployment to its application, and after tearing down, will leave no waste or pollutants in the area. When used appropriately, our system will create little to no disturbance to its surroundings.

### 4.3.5 Sustainability Constraints:

A major concern for our system is its ability to operate effectively over time, particularly in suboptimal outdoor conditions. Sustainability focuses on how well our system can function over long periods while minimizing resource consumption. Given that the primary usage of the deployable security system will be outdoors, our design process aimed to create a device that not only serves users efficiently but also offers a long operational lifespan.

Durability plays a crucial role in sustainability. A highly durable system reduces the need for frequent replacements and minimizes waste. The physical components of our system, such as the frame, will be made from ABS filament, known for its toughness and resistance to environmental stress. This material choice ensures that our system can withstand outdoor conditions such as rain, wind, and temperature fluctuations. Additionally, the ESP32-CAM nodes in our system are designed to operate with low power consumption, which extends battery life and significantly reduces the frequency of battery replacements.

In terms of energy consumption, our product is designed for continuous operation in outdoor environments. We selected lithium-ion batteries for their high energy density, which strikes a balance between power requirements and sustainability. These batteries are lightweight, portable, and offer longer charge cycles than other technologies, reducing waste and the environmental impact associated with battery disposal.

To address hardware issues and ensure easy maintenance, our system features a modular design. This allows non-technical users to perform basic repairs without the need for specialized tools or expertise. If one of the nodes malfunctions, the issue can be easily identified and resolved. This repair-friendly approach reduces the likelihood of users discarding the entire system due to a single component failure.

In summary, sustainability is a vital consideration in the design of our Deployable Security System. By prioritizing material durability, efficient power management, and modularity for easy repair, we ensure that our system remains functional, reliable, and sustainable over time.

### 4.3.6 Ethical Constraints:

Our product's main purpose is security and the safety of the user. Our product will give the user a higher situational awareness of the surrounding area and give them a heads-up on perceived threats. With this surveillance of the surrounding area, we must not cross the line into intruding on others' personal privacy. To keep in line with ethical concerns we will be using our state laws as our ethical guideline.

Our design is meant to be used in camping and outdoor activities. Many campgrounds are on either state or federal land and are considered public areas, even businesses are considered public areas in terms of privacy. Florida Statutes Title XLVI. Crimes § 810.145. Video voyeurism (5). states as follows: “(c) Video surveillance device that is installed in such a manner that the presence of the device is clearly and immediately obvious;”. Our product is clearly marked and visible as it is made to be a deterrent. We will not be implementing any Stealth features to disuses our product.

Along with design we need to consider if the end user missuses our product. In the same statute: “(d) Dissemination, distribution, or transfer of images subject to this section by a provider of an electronic communication service as defined in 18 U.S.C. s. 2510(15), or a provider of a remote computing service as defined in 18 U.S.C. s. 2711(2). For purposes of this section, the exceptions to the definition of “electronic communication” set forth in 18 U.S.C. s. 2510(12)(a), (b), (c), and (d) do not apply, but are included within the definition of the term.” This subsection defines us as the service provider and aren't held accountable for misuse of our product or services.

We also have operation states that let the end user determine the uses of our equipment. With the ability to use just the IR triggers with no Camera and the quiet mode that doesn't use the alarms, end users can be conscientious of the surrounding area and of others privacy and peace.

### 4.3.7 Software Constraints:

Software constraints are a very common difficulty when developing any system, therefore, this is a crucial section to include. Some of the issues for software addressed above are memory management and time, but there are plenty more to be discussed for the scope of this project. One of the other major software constraints is a resource constraint. When running our image processing algorithms, a serious issue we are going to encounter is the amount of computational power required by our algorithm vs what our board is going to offer. Extensive research has been performed for this topic and we have concluded that TensorFlow Lite is going to be a library low-level enough for us to meet the objectives of our goal in a timely manner while not crashing our MCU. Another software constraint must be developed for real-time processing. Meaning all algorithms must be geared and architected carefully with a low latency design in mind. Plans to achieve such a task have been accounted for with research into the best data structures that will result in the lowest run times. Typically, with a real time system the data structures should aim to be as close to O(1) run time as possible.

# Chapter 5 - Comparison of ChatGPT with other similar Platforms

Chat GPT, developed by OpenAI, is a free virtual assistant / chatbot that came online in November of 2022 and quickly gain popularity. There are many products on the market like ChatGPT like Claude and Google Gemini. But ChatGPT is the most popular, this is in part due to it being free and being easily accessible as well as having versatility across multiple disciplines. It also is very good at formatting information to be easily digestible as possible. With many added features like image generation capabilities and conversation memory features, users feel very connected to the platform. The only negative many users experience is occasional latency when accessing real-time information. But overall, the platform is highly competent at helping with a wide range of tasks.

Google's Bard excels particularly in current events analysis and general inquiries, leveraging its direct integration with Google Search and the world wide web. The platform offers seamless compatibility with Google Workspace applications, providing significant value for users invested in the Google ecosystem. However, its responses tend to be less sophisticated than ChatGPT's, and its programming capabilities are somewhat limited. Googles Bard seems to be focused on a wider user base rather than deep In-depth analysis.

Microsoft's Bing Chat, powered by GPT-4 technology, combines advanced language processing with Bing's search functionality. While it should be more functional than GPT alone, Bing chat feels limited in its capabilities. Its integration with Microsoft Office applications makes it particularly suitable for professional environments, though it typically provides more concise responses compared to its competitors.

Claude, Anthropic's contribution to the field, distinguishes itself through its emphasis on ethical considerations and exceptional clarity in communication. Its capacity to process extensive text inputs makes it particularly valuable for document analysis and summarization tasks. It shows a good level of nuanced understanding of complex topics. While it may not offer comprehensive coverage of technical subjects such as coding or as many interactive features as its counterparts, it excels in its core functionalities.

Each platform offers distinct advantages that cater to specific use cases and user preferences, making the optimal choice dependent on individual requirements and objectives.

| Feature | ChatGPT | Google Bard | Bing Chat | Claude |
|---|---|---|---|---|
| **Real-Time Info** | Limited (Browsing) | Yes | Yes | Limited |
| **Code Proficiency** | High | Moderate | Moderate | Moderate |
| **Tool Integration** | DALL-E, Code Tools | Google Services | Office Suite | None |
| **Content Length** | Medium to Long | Medium | Short | Long Context Support |
| **Best Use Case** | Multi-functional | Real-Time Insights | Office Integration | Document Analysis |

Table 5.1: AI Comparison Table

We used Chat GPT and other models extensively throughout this project and report. The benefits to the more tedious and mundane parts of the report like consistent tone, consistent tense, wording, and grammar are a tremendous boon to our team. These functions were used throughout the report. Most of our processes with these tools were regenerative in nature. This means we have information, and we are using these tools as a standardizer and checker rather than using them as a generative tool that produces new information.

We also have used them to gain different perspectives than our own. Listed below were a few ideas we had and used ChatGPT to give ideas and feedback that we might not have thought of.

We have also used ChatGPT for technical help too. We have used it to verify the thought processes of wiring between components and verify if there are any logical faults we have missed. It has also been used to help explain technical language in documentation, quickly explaining foreign nomenclature and verbiage. It has also been used to verify code and syntax of said code.

While we can't document every time, we used these tools as they were used within the development process as a regenerative tool, we still want to give them the credit they are due for being useful tools.

## 5.1 Case study 1:

**Prompt:** “What functionality should a deployable security system have, noting it will need to be on battery power.”

**Outcome:** A deployable, battery-powered security system needs to balance functionality with power efficiency. Here are key features it should have:

1. Power Management: The system should have low-power operation, optional solar charging, and battery monitoring for power efficiency.

2. Sensors for Detection: It should include motion detection (PIR or microwave radar), cameras (with night vision), and environmental sensors for tamper detection.

3. Communication & Alerting: Wireless connectivity via Wi-Fi, LoRa, or cellular for remote alerts, local alarms, and push notifications. It should offer cloud storage for footage.

4. Video Recording & Storage: Onboard storage (e.g., SD card) with triggered recording to save power, along with a low-power live video feed option.

5. User Interface: Mobile app control, web interface for remote access, and a physical keypad for local control.

6. Tamper Resistance: The system needs a rugged, weatherproof enclosure, tamper alarms, and backup communication to resist physical interference.

7. Expandability & Customization: It should allow additional sensors or cameras, and include geofencing for auto arming/disarming based on GPS.

8. Autonomy & Smart Features: AI integration for object detection, and a self-learning algorithm to reduce false alarms based on typical movement patterns.

By focusing on power-efficient components, modularity, and robust communication options, the system can balance effectiveness with long-term deployment on battery power.

**Takeaway:**

In this case Chat GPT gives us good information on our direction with our product. Most of the features it suggested have already either been implemented into the design or looked at and deemed outside the scope of the project. One of these features being Night vision. While useful the technology adds a lot of cost and complexity that is too large for a prototype like this.

## 5.2 Case study 2:

**Prompt:** "Please provide the software architecture and design to approach such a project like a deployable security system, please include both code written on the gadget (embedded and python) and code written on the node(s) both embedded and python. Thank you!"

**Outcome:** To create a deployable security system involving gadgets and nodes, we can break down the software architecture and design into two main components: **Node** and **Gadget**. Each component will interact with each other via a network, and both will have specific responsibilities.

**System Overview:**

- **Nodes**: These ESP32-CAM devices will act as security monitors, capturing images or video and sending them to the central gadget. The nodes will operate over a network using secure communication protocols.
- **Gadget**: This acts as the main hub that processes the data sent from the nodes. The gadget will run Python code to process images and trigger any necessary actions like alarms or alerts based on the motion detected.

**1. Node Design (ESP32-CAM)**

**Node Overview:**

The nodes are ESP32-CAM devices placed at various security points. They capture images or video and send data to the gadget when triggered.

**Key Tasks:**

- **Image/Video Capture**: Using the ESP32-CAM camera.
- **Motion Detection**: Using an infrared (IR) sensor or onboard algorithm to detect motion.
- **Secure Communication**: Sending images securely to the gadget.
- **Periodic Heartbeat**: Sending regular updates to ensure the node is functional.

**Node Code (Embedded - C++):**

- **Libraries**: WiFi, WebServer, esp32cam, SSL, IR sensor.
- **Key Functions**:
  - **WiFi Setup**: Connect to the local network using WiFi.
  - **Image Capture**: Capture images at a specified resolution and send them to the gadget.
  - **Motion Detection**: Detect motion using the IR sensor and trigger image capture.
  - **Secure HTTP**: Use SSL to ensure secure communication when sending the captured data to the gadget.
  - **Health Checks**: Send regular pings or heartbeats to ensure the node is active.

**2. Gadget Design (Raspberry Pi or PC)**

**Gadget Overview:**

The gadget serves as the central hub, receiving and processing images or videos from nodes, analyzing the data (motion detection), and providing output actions (alerts, alarms, etc.).

**Key Tasks:**

- **Image Reception**: Receiving images sent by the nodes.
- **Motion Detection**: Using a Python-based algorithm (OpenCV) to analyze images for any motion.
- **Alerts/Actions**: Trigger alarms or send notifications if motion is detected.
- **GUI**: A simple dashboard to view images and control the system.

**Gadget Code (Python):**

- **Libraries**: requests, OpenCV, Flask (optional for GUI), Socket, SSL.
- **Key Functions**:
  - **Receive Image**: A Python web server that listens for incoming images from nodes.
  - **Motion Detection**: Use OpenCV to process the received image.

- **Trigger Alarms**: Based on motion detection, trigger alarms or alerts.

**3. Communication & Security**

**Data Flow:**

1. **Node** captures an image using the ESP32-CAM when motion is detected.
2. The image is sent over HTTPS to the **Gadget**.
3. The **Gadget** receives the image, processes it for motion detection, and triggers alerts if needed.

**Security:**

- **SSL/TLS**: Secure the communication channel between the node and gadget.
- **Token Authentication**: Optionally, each node can use token-based authentication for additional security when sending data.

**4. Image Processing Design**

**Algorithm:**

- **Background Subtraction**: Compare the current image frame with a reference background to detect motion.
- **Thresholding**: Convert the image into binary for easy processing.
- **Contours Detection**: Use OpenCV to identify and highlight objects that have moved.

**Summary**

- **Nodes**: ESP32-CAM devices that capture images and send them securely to the gadget.
- **Gadget**: A central processor running Python, which processes images from the nodes and performs actions (motion detection, alerts).
- **Communication**: Secured via SSL, possibly with token-based authentication for added security.
- **Image Processing**: Uses OpenCV for detecting motion or analyzing images.

This approach balances hardware and software across both the embedded (Node) and application (Gadget) layers, ensuring secure and efficient operation.

**Takeaway:**

This response was very beneficial in the early stages of software development. It provided a very detailed guide of how to approach designing such a complex system and lead to validation of current ideas while also inspiring new ones. Validated ideas consisted of focusing on developing the image processing algorithms on the gadget and having the nodes send image captures over a secure communication message. New ideas inspired consisted of a heartbeat to ensure node functionality throughout the running program. This idea will let us be confident that our system is always working.

## 5.3 Case study 3:

**Prompt:** "How do I upload more than one file to an esp32 board when running code?"

**Outcome: Using Arduino IDE**

The Arduino IDE can also be used, but managing multiple files is a bit different.

**Steps:**

4. **Open Arduino IDE**: Make sure you have installed the ESP32 board package in the Arduino IDE.
5. **Create Tabs**:
   - To manage multiple files, create new tabs within the Arduino IDE.
   - Each tab will represent a file. You can create a .cpp file or a .h file by clicking the arrow button on the right side of the Arduino IDE and selecting **New Tab**.
6. **Add Your Files**:
   - Add your .cpp and .h files as needed.
   - The Arduino IDE will automatically compile all the files when you upload your project.
7. **Upload**:
   - Click the **Upload** button in the Arduino IDE to upload all the files to the ESP32.

**Takeaway:**

This response was very detailed and helpful as this was one of the main questions I had as to how we will implement software onto our board. From my experiences in Embedded Systems, I was knowledgeable in how to upload one code file to a board, however, I was not sure how to implement uploading multiple code files to a board, specifically to the ESP32 through our development platform Arduino IDE.

## 5.4 Case Study 4:

**Prompt:** "I'm creating a deployable security system that will be powered by lithium batteries. What other considerations should I have in terms of power?"

**Outcome:** Battery capacity and power requirements, power management circuits, solar integration, low power modes, Thermal considerations and battery longevity.

"When designing a deployable security system powered by lithium batteries, there are several critical power-related considerations you should take into account:

- **Battery Capacity & Power**: Calculate total power consumption and select batteries with enough capacity (mAh). Use energy-efficient components and optimize the ESP32-CAM's duty cycle.
- **Power Management**: Use buck/boost converters for stable voltage, battery protection circuits, and power monitoring to prevent overcharge/discharge and monitor usage.
- **Solar Integration**: Size the solar panel to match charging needs and use a solar charge controller to regulate input to the batteries.

- **Low Power Modes**: Utilize low-power or deep-sleep modes on components, and power down unused parts to extend battery life.
- **Thermal Considerations**: Ensure heat dissipation to avoid battery overheating during charging or heavy use.
- **Battery Longevity**: Plan for battery degradation over charge cycles and avoid charging in extreme temperatures.
- **Weatherproofing**: Use a weatherproof enclosure for outdoor deployments to protect the batteries and electronics.

Addressing these considerations will help you build a reliable and efficient power system for your deployable security system!" (OpenAI, 2024)

**Takeaway:**

This response is interesting because it remembers the past chat interactions about choosing to work with ESP32-CAMs without mentioning it in the prompt. ChatGPT correctly identifies many important aspects of powering our system. The most helpful feedback given was to consider how to dissipate heat to avoid damaging our electrical system. Because of this response, we will make efforts to create a ventilation system. Independent research within our group has addressed every other point made by ChatGPT.

## 5.5 Case Study 5:

**Prompt:** "I'm creating a deployable security system that uses cameras and sensors to detect threats, I want to integrate a mobile app for alerts and control. We are currently using an ESP32-CAM and ESP32-S3 for communication. What would be the best way to approach this?"

**Outcome:**

"To integrate a mobile app for alerts and control into your deployable security system using ESP32-CAM and ESP32-S3, here's a simple breakdown:

**Backend Setup (Server for Alerts & Control):**

Set up a lightweight backend (e.g., Node.js, Flask, or Firebase) to handle communication between the ESP32 devices and the mobile app.

The ESP32s will send data (like motion detection or camera images) to this server using HTTP requests or WebSockets.

The backend can trigger push notifications to the mobile app using services like Firebase Cloud Messaging (FCM).

**ESP32 Firmware (Data Transmission):**

Program the ESP32-CAM to capture images and detect motion. When a motion is detected or at regular intervals, send the data to the backend.

The ESP32-S3 can manage additional sensor data (e.g., temperature or IR sensors) and send updates to the server.

Use Wi-Fi for communication between the ESP32 devices and the server, sending data via HTTP POST requests or MQTT.

**Mobile App (Alerts & Control):**

Build a simple mobile app using Flutter or React Native for cross-platform compatibility.

Integrate the app with Firebase Cloud Messaging (FCM) or another notification service for real-time alerts when motion or threats are detected.

The app should also allow the user to control settings, view the camera feed (by pulling from the backend), and manage the system remotely (turning on/off sensors, adjusting settings, etc.).

**Real-Time Video Streaming (Optional):**

Set up real-time video streaming from the ESP32-CAM. This can be done by creating a web server on the ESP32-CAM that streams video, and then accessing the stream from the app through the backend.

This approach will allow for real-time alerts and system control via the mobile app, ensuring your deployable security system is functional and user-friendly."

, (OpenAI, 2024)

**Takeaway:**

The response provided a clear and actionable plan for integrating a mobile app with our deployable security system. It addressed essential components such as backend setup, data transmission, and mobile app development. The recommendation to use Flutter is particularly valuable, as extended research shows it simplifies the creation of clean, intuitive UI components. This guidance provides a solid foundation for establishing communication between our physical devices and the mobile app, enhancing the system's overall usability. We will incorporate these recommendations throughout our design and development process to ensure seamless integration and an improved user experience.

# Chapter 6 – Hardware Design

## 6.1 Hardware Design Introduction:

The hardware designs will be split into two different sections for better clarity. Our gadget is up first because it will be acting as a centralized hub for our entire system. Every possible interaction within our system is through the gadget because of the router placed inside of it. Our nodes are next and will show the many peripherals and how they are connected to function properly. Lastly, we will present how our software integrates into the design.

We used multiple different resources to help with the design of the schematics, we used open-source designs to seed ours. We used esp32-s3-base by jonmill [50] for the base of our power regulation, ethernet base was LAN8720 Ethernet by GarryChen[51], and Espressif PCB document for ESP32-S3 [52]. All easyeda sources used are public domain and open sourced.

## 6.2 Gadget Design Introduction:

The gadget designing process is centralized around the ESP32-S3 microprocessor and will be directing the many peripherals to keep the unit acting as a whole. The many additions each have their own supporting passive devices like resistors to help distribute power, making sure that the additions are given their desired voltage and amperage.

### 6.2.1 Gadget Overview:

This schematic represents a complete circuit design for our gadget design, powered by the ESP32-S3 microcontroller. The system is divided into sections, maintaining power regulation, communication interfaces, and peripheral integrations in their respective places. Voltage regulation in our design uses an AMS1117-3.3 regulator to get stable 3.3V output from 5V input power. The system is designed run smoothly with several subsystems: decoupling capacitors, RTC power, voltage protection, and so on.

At the heart of the design is an ESP32-S3 interfaced to external peripherals: flash memory, Ethernet for wired networking, and a chip antenna for wireless. These functions are complemented with a USB-C port for power and data transfer and peripheral connectors for LEDs and switches. The design incorporates robust filtering, clock generation via crystal oscillators, and modularity for ease of integration. This schematic showcases our versatile embedded system that balances power management, connectivity, and peripheral interaction effectively.

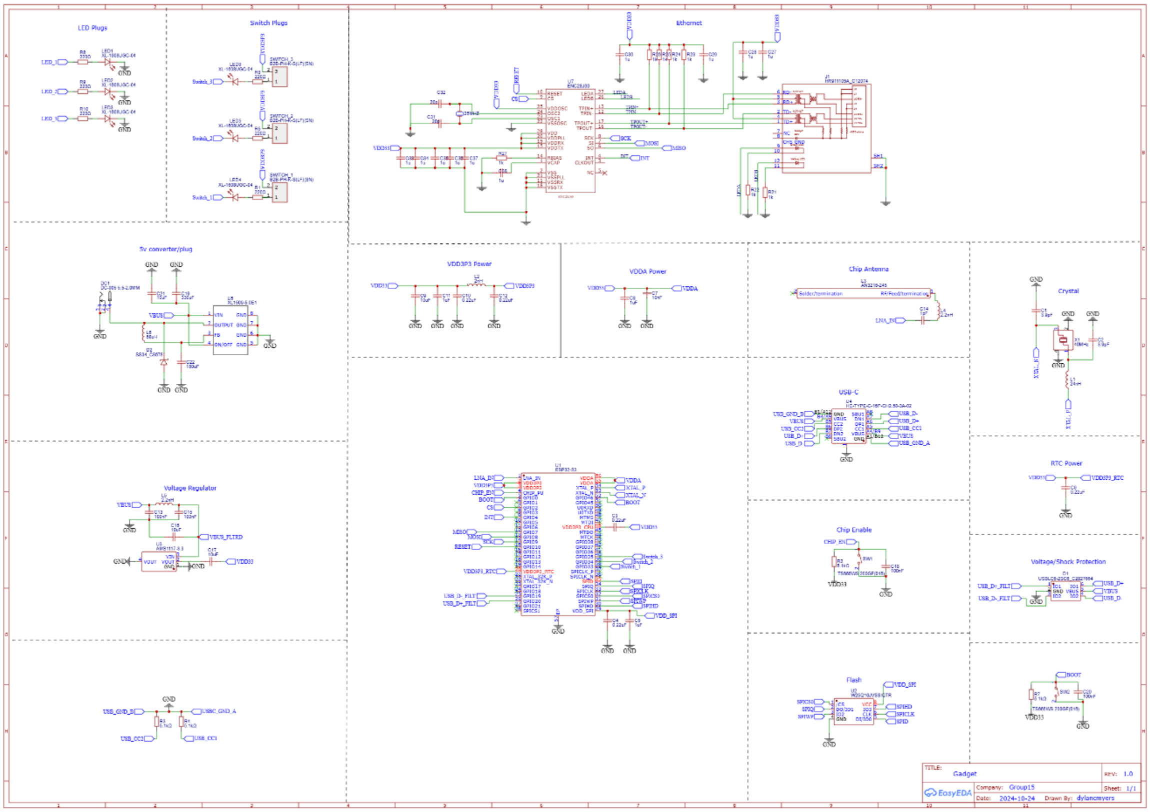


Figure 6.2.1: Gadget Schematic Overview

## 6.2.2 Power Distribution:

### 6.2.2.1 Voltage Regulation:

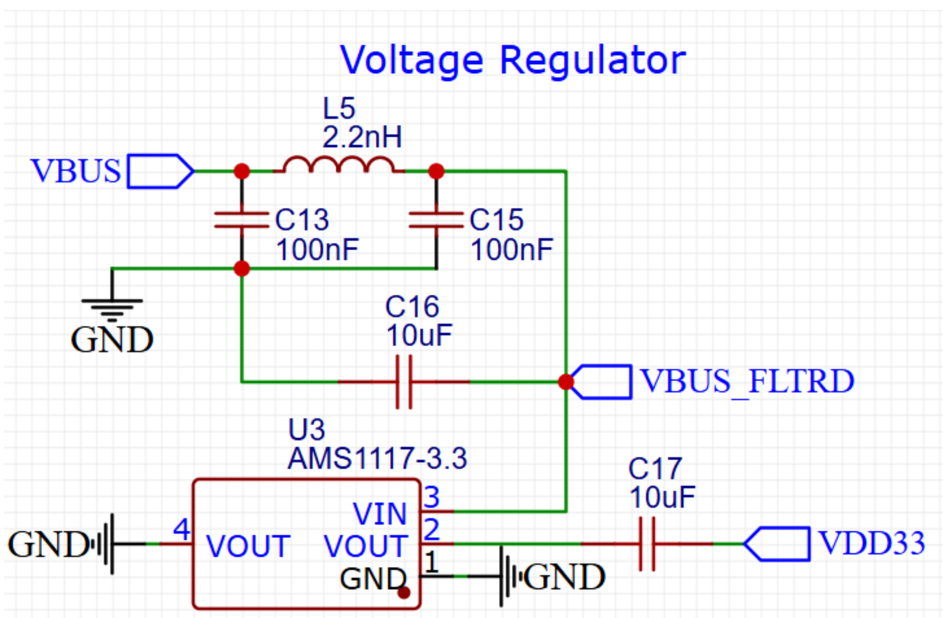



Figure 6.2.2.1: Voltage Regulator

This section shows the voltage regulation circuit that steps down an input voltage, usually 5V sourced from, say, USB VBUS to a stable 3.3V output VDD33. At the heart of it is the linear voltage regulator AMS1117-3.3. The input voltage passes through a noise-filtering network: a 2.2nH inductor L5 and two 100nF capacitors C13 and C15. This stage of filtering, VBUS_FLTRD, reduces high-frequency noise before it feeds into the voltage regulator.

The filtered voltage is regulated by the AMS1117-3.3 to a steady output of 3.3V. To ensure stability in its operation and to damp out voltage ripples, the stabilizing capacitors are placed at the regulator input and output stages, respectively (C16 and C17, each 10µF). The regulated output voltage VDD33 can be used for feeding low voltage electronics such as microcontrollers and sensors. It gives a clean and stable power supply with proper noise suppression and output stabilization for sensitive devices.

**6.2.2.2 VDD3P3 Power:**

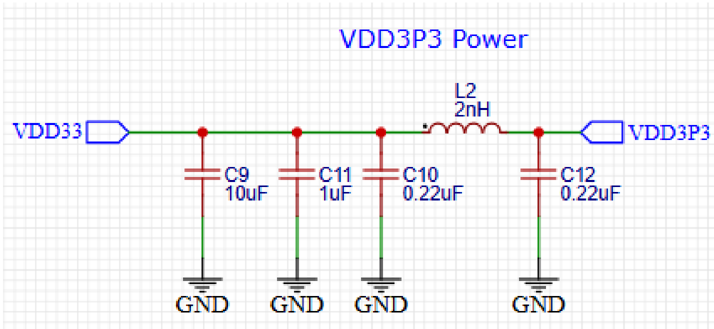



Figure 6.2.2.2: VDD3P3 Power

The VDD3P3 power is set up to capture any high frequency and unstable power. From the AMS1117-3.3 at VDD33. The capacitors are used to capture and smooth power. These are commonly referred to as decoupling capacitors. The inductor throttles amperage to smoothen out the power. Due to VDD3P3 powering many of the sensitive functions of the ESP32-S3 it is important to have smooth power.

**6.2.2.3 VDDA Power:**

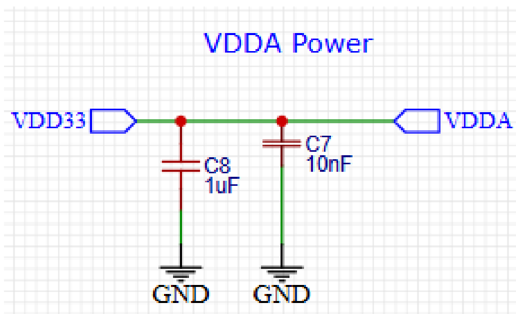



Figure 6.2.2.3: VDDA Power:

VDDA is the main power of the analog circuits of the ESP32-S3 at 3.3V, given power from the AMS1117-3.3 at VDD33. This also has decoupling capacitors to ensure smooth power supply.

### 6.2.2.4 RTC Power:

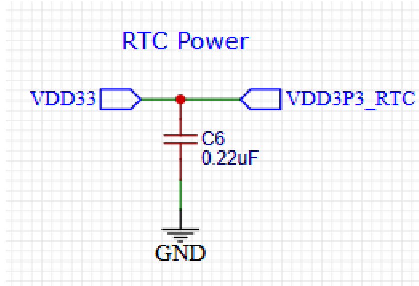



Figure 6.2.2.4: RTC Power

RTC (Real-Time Clock) power also is supplied by the AMS1117-3.3 at VDD33. RTC is important to a lot of the internal time keeping functions in the ESP32-S3. This also has a decoupling capacitor for smooth power supply.

### 6.2.2.5 Voltage Shock Protection:

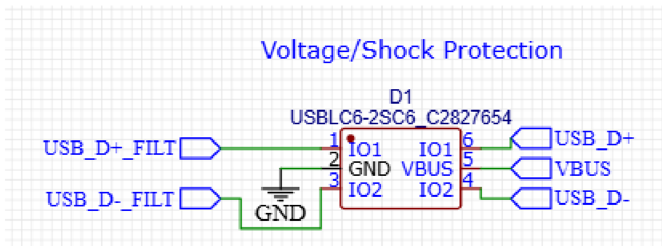



Figure 6.2.2.5: Voltage Shock Protection

The protection of voltage/shock is highly critical in saving USB communication and power lines from ESD, voltage surge, and other transient events. The USBL6-2SC6 diode array plays a keystone role in protecting both USB D+/D- data and VBUS power lines by clamping excess voltage to ground, ensuring stability and longevity for the system.

The design ensures that the pre-filtered USB signals-D+, D-, and VBUS power-are fed through a protection diode to pass on to the rest of the circuit. Such a setup is important in most embedded systems using USB-C or other USB interfaces, since frequent connections and disconnections could expose a system to ESD or transient voltage spikes. This protection cleans up the circuit from damage to sensitive components and works quite reliably.

## 6.2.3 Boot:

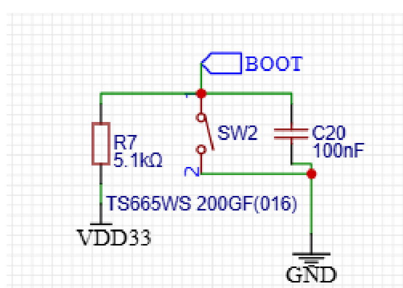


Figure 6.2.3: Boot

This set up allows us to put the ESP32-S3 into Boot mode or Flash mode. This lets us add new code onto the ESP32-S3. Added in resistor and capacitor to remove debounce and unwanted amperage.

## 6.2.4 Chip Enable:

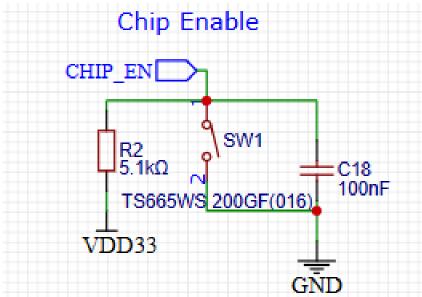



Figure 6.2.4: Chip Enable

The Chip Enable (EN) pin on the ESP32-S3 acts as a power control or reset pin. EN pin is pulled high, the ESP32 is powered on and operational. When turned to low the chip resets. This functionality is essential for power management and resetting.

## 6.2.5 Common Ground:

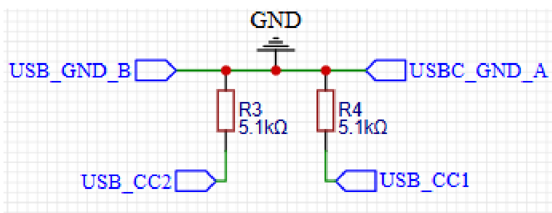



Figure 6.2.5: Common Ground

This section connects the USB channels and their grounds to create a common grounding. Common ground gives stability to the system and prevents feedback and voltage spikes.

## 6.2.6 External Flash:

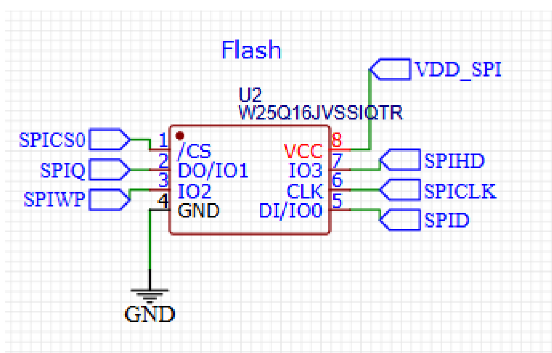



Figure 6.2.6: External Flash

This is the SPI Flash memory, external storage for the ESP32-S3 microcontroller using a W25Q16JV Flash chip. The Flash memory uses a standard SPI interface with Chip Select, or CS; Clock, or CLK; and data lines for MISO (DO/IO1) and MOSI (DI/IO0). The other two pins are for Write Protect, or WP#, and Hold, or HOLD#, respectively, allowing various controls over write operations and state maintenance upon communications pause.

The integration of external Flash is a very critical point for applications with higher non-volatile storage requirements; for instance, data logging, or high resource-demanding programs. Using the ESP32-S3 SPI interface, the code can perform very high-speed reads and writes on the Flash memory. This design choice enhances flexibility and memory capacity in the ESP32-S3 system for more complex embedded applications.

### 6.2.7 External Crystal:

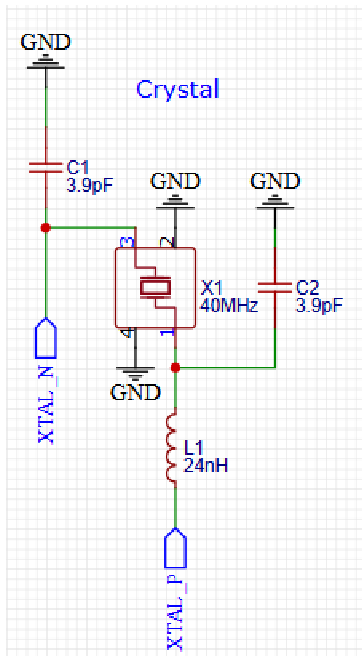



Figure 6.2.7: External Clock

This crystal oscillator circuit provides a 40 MHz clock to the microcontroller ESP32-S3. This drives all timing, processing, and communication functions within the device. The circuit consists of a 40 MHz crystal X1, load capacitors C1 and C2, and a tuning inductor L1. The crystal is connected to the XTAL_N and XTAL_P oscillator pins of the microcontroller in the standard Pierce oscillator configuration.

Load capacitors of 3.9 pF each help provide the right loading on the crystal to stabilize its oscillation frequency. The 24 nH inductor in series with the XTAL_P line is probably for fine-tuning to get the best condition of oscillation and for noise reduction. A stable source of a clock is principal in the operation of the ESP32-S3, making sure everything runs smoothly from internal processing to peripheral communications. The design uses a dedicated crystal oscillator circuitry for its accurate and stable clock.

### 6.2.8 Chip Antenna:

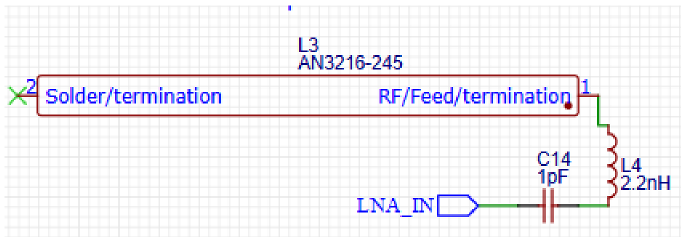



Figure 6.2.8: Chip Antenna

This RF circuitry forms part of the antenna interface of the ESP32-S3, destined for radio signal feeding into the LNA input of the microcontroller. The 1 pF capacitor, C14, and a 2.2-nH inductor, L4, form an impedance matching network-very important for ensuring maximum signal transfer and minimum loss at the LNA input.

This design effectively provides the least possible impedance mismatch between the antenna feed line and the LNA input for effective reception of RF signals. It is highly important in maintaining signal integrity and optimizing performance within the ESP32-S3 radio module, with the main emphasis on WiFi or Bluetooth communication. Proper matching of antenna and RF input increases sensitivity and reduces distortion for better overall wireless performance.

### 6.2.9 USB – C Connector:

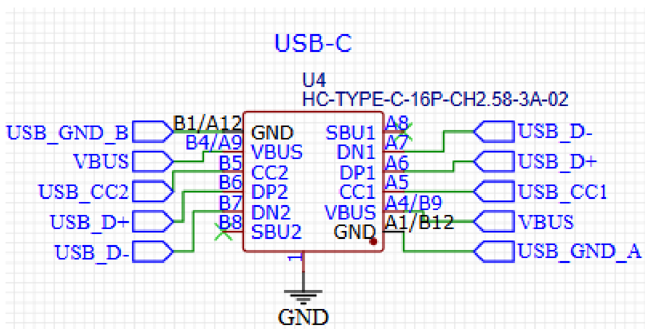



Figure 6.2.9: USB-C Connector

This USB-C connector interface can support data communication and power deliverables. Included are the VBUS and GND connections for power input, differential data lines (D+/D-), configuration channels (CC1 and CC2), and optional sideband use (SBU1 and SBU2). The USB data lines support USB 2.0 high-speed data rates, and the configuration channel pins provide a means for the system to determine the USB-C plug orientation and initiate power delivery negotiations.

The CC pins present are crucial for the functionality of USB-C since they detect the orientation of the cable and initiate the communication to make use of power delivery to enable standard charging and higher current modes of up to 3A. Sideband pinsSBU1, SBU2-offer more flexibility for potential future use in alternate modes of application, such as in DisplayPort. It supports a universal USB-C interface that assists in efficiently handling modern power and data demands, hence making the equipment optimal for data transfer, charging, and extended functionalities applications.

### 6.2.10 Ethernet Port:

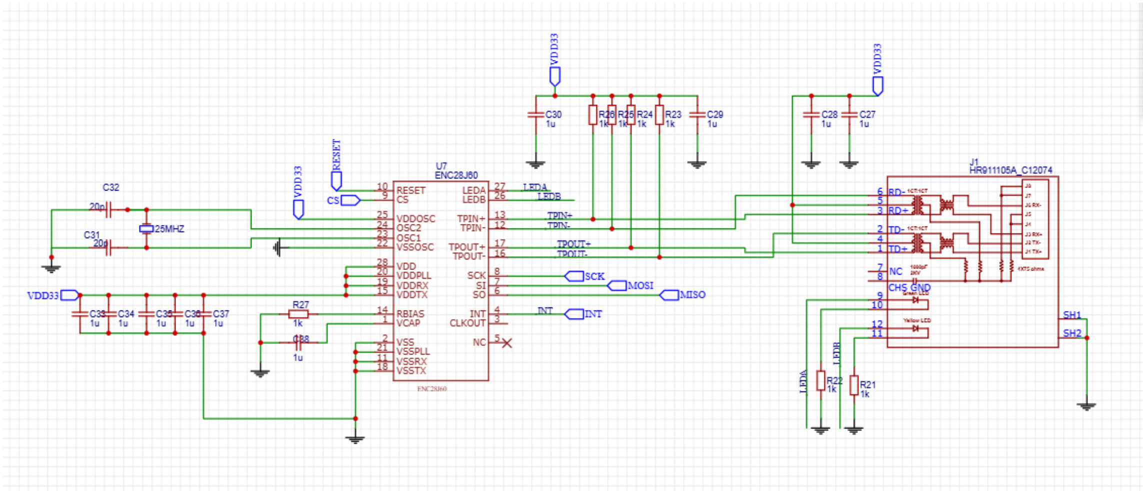



Figure 6.2.10: Ethernet Port:

This schematic shows the integration of an ENC28J60 Ethernet controller connected with the ESP32-S3, enabling wired Ethernet connectivity. The ENC28J60 interfaces with the ESP32-S3 via an SPI bus, using dedicated lines for data transfer, clock, and chip select. An interrupt line is also provided for event signaling. A 25 MHz crystal oscillator provides the precise timing needed for Ethernet operations, while decoupling capacitors and voltage regulation ensure stable power for the controller.

The Ethernet connection is established through an RJ45 connector with integrated magnetics, which provides electrical isolation and impedance matching for the Ethernet signals. This design allows the ESP32-S3 to offload Ethernet communication tasks to the ENC28J60, enabling robust and efficient wired network functionality. This type of setup is ideal for IoT devices or embedded systems requiring high-speed and reliable network connectivity in environments where wireless communication might be impractical.

## 6.2.11 Switch Breakout:

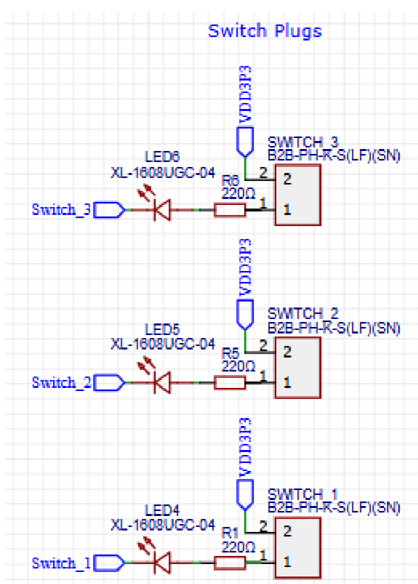


Figure 6.2.11: Switch Breakout

This circuit provides a simple plugin for our switches, each paired with a green LED indicator. The switches are connected to the ESP32-S3 via labeled plugs, allowing them to act as inputs for user commands. The LEDs, positioned next to each switch, provide visual feedback, such as indicating whether a switch has been switched or a particular function is active.

The design ensures modularity by using 2-pin headers for each switch, making it easy to connect and disconnect during prototyping or maintenance. The inclusion of current-limiting resistors protects the LEDs while ensuring sufficient brightness. This circuit is made for our applications requiring manual user input, such as mode selection, system reset, or other interactive functions. The integration with the ESP32-S3 provides flexibility, as the microcontroller can monitor the switches.

## 6.2.12 LED Breakout:

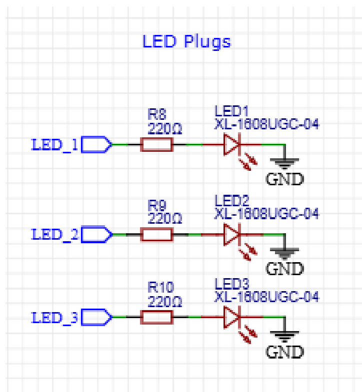



Figure 6.2.12: LED Breakout

This LED plug interface provides a simple, modular way to add visual feedback to an embedded system like the ESP32-S3. The LEDs are protected against overcurrent conditions using 220 ohm current limiting resistors, which regulate the current flow and ensure safe efficient operation.

This design is good for providing indications of various system states or events such as power status, activity indication, or debugging. The individual plugs involved in the modular approach will help the LEDs to be easily replaced, relocated, or connected with different microcontroller pins, enhancing the flexibility of the system during prototyping and deployment.

## 6.2.13 5 Volt Power Jack:

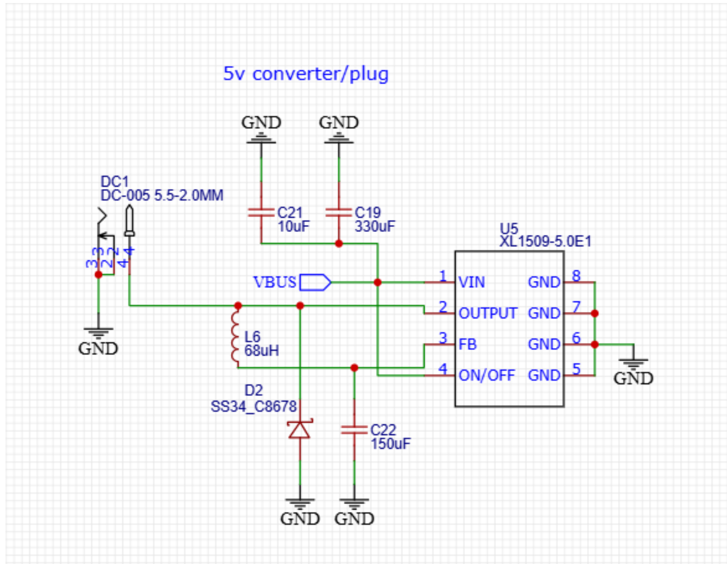



Figure 6.2.13: 5 Volt Power Jack

Following is a schematic of the 5V step-down DC-DC converter circuit, which is integral to the power management of the Wireless router. The buck converter IC XL1509-5.0 efficiently converts a higher input voltage-in supplied from an external DC power supply-to a regulated 5V output.

Key features such as input decoupling capacitors, a 68µH inductor, and a Schottky diode have been integrated to ensure the efficiency of the converter and provide stable output. The DC barrel jack offers flexibility in choosing external supplies for the power input. This design is designed as a robust and efficient power source for embedded systems, especially in areas where low power dissipation and stable operation become critical such as ours.

## 6.3 Node Design:

### 6.3.1 Node Design introduction:

The node is centered around the ESP32-CAM and represents a modular, adaptable basis to help support several functions. We have taken design direction to ensure that reliability, and ease of use Easy integration of peripherals were key points within our node to meet the requirements of our project.

This section describes the main components comprising our node design: power management, USB-C, UART communication, and interfaces for peripheral devices. Each of these elements has been developed to meet our design goals.

## 6.3.2 Node Overview:

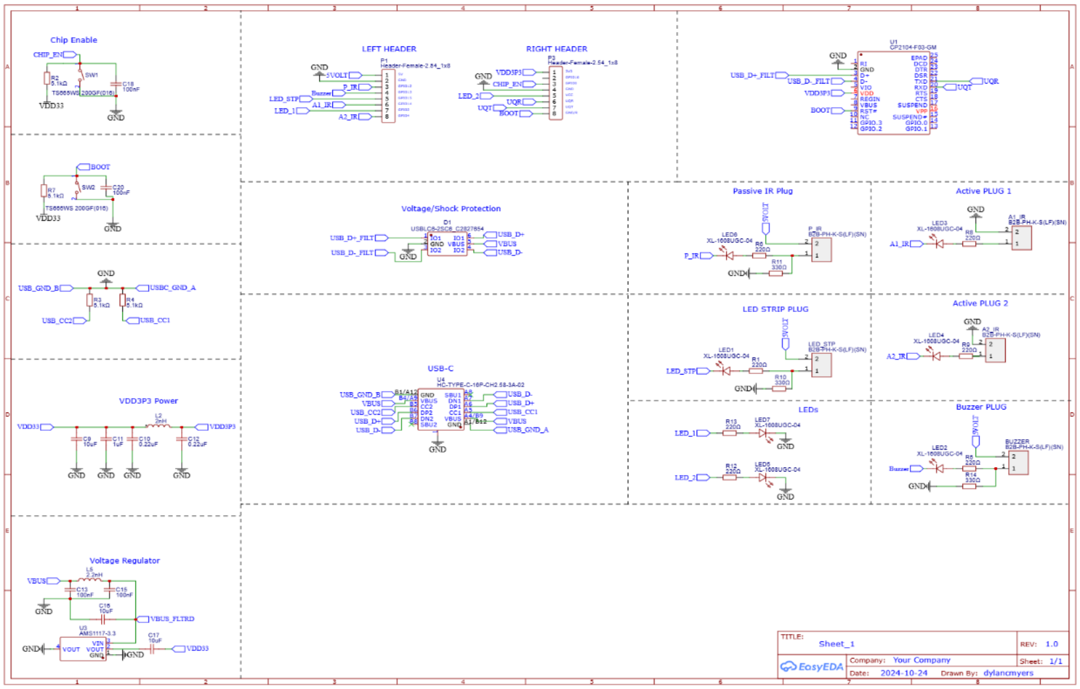


Figure 6.3.2: Node Overview

This schematic represents a modular and flexible interface design tailored for the ESP32-CAM. This module makes it central, basically, for peripheral devices, while it also serves as an interface for programming- firmware uploads. Our design features robust power management with a regulated 3.3V supply and includes a USB-C interface for both power and data transfer. Signal protection increases the reliability and robustness of the system in harsh environments susceptible to voltage spikes or ESD.

Our design focuses on peripheral integration, having dedicated plugs for LEDs, motion sensors, infrared sensors, and buzzers. The ESP32-CAM can easily connect to most devices, making the development board perfect for IoT, surveillance, or smart system projects. Inclusion of headers for external connections and the support for programming make this design versatile, expandable, and user-friendly for our project.

## 6.3.2 Power Distribution:

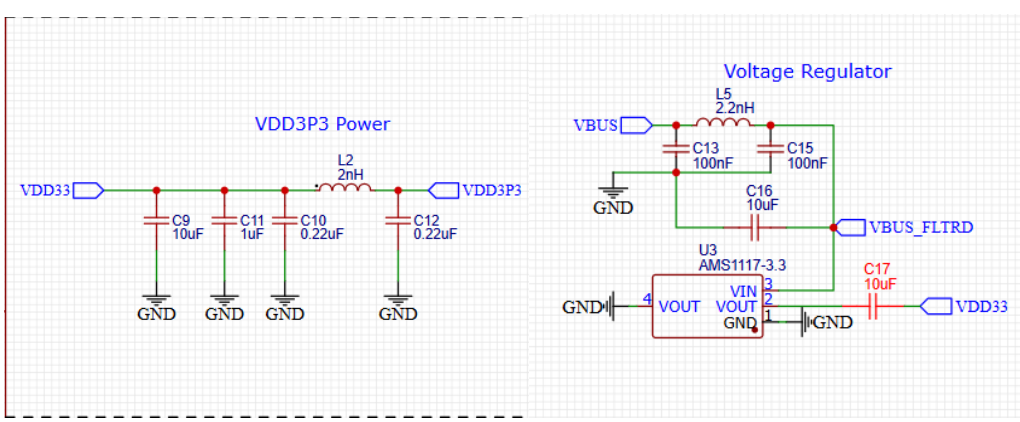


Figure 6.3.2: Power Distribution

These are the same setups as the power setup of the gadget. Once again, we use the AMS1117 for steady 3.3v and decoupling capacitors for clean power and removal of high frequencies.

## 6.3.3 USB-C Connector:

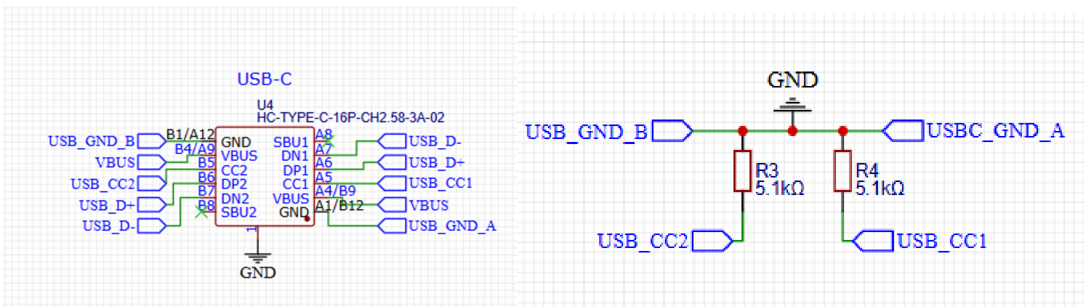



Figure 6.3.3: USB-C Connector

This is the same set up as the gadget. The USB will be used as both a power source and the data supply for installing code. This is also with appropriate grounding to chasse.

## 6.3.4 UART to Serial Converter:

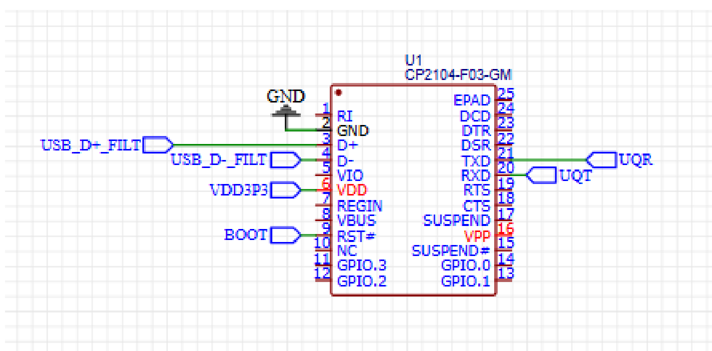



Figure 6.3.4: UART to Serial Converter

This circuit incorporates the CP2104 USB-to-UART bridge into the ESP32-CAM PCB, which will help to program or communicate with the microcontroller using USB. The CP2104 translates USB signals from a host, such as a computer, into UART signals that the ESP32-CAM can recognize. Filtered USB data lines ensure quality in the signal, while power is supplied by the regulated 3.3V rail for stable operation. The UART TXD and RXD lines are directly connected to the ESP32-CAM to ensure reliable communication.

Additional functionality is provided by BOOT and RST pins, which allow the CP2104 to trigger bootloader and reset modes on the ESP32-CAM for seamless programming. GPIO pins and SUSPEND functionality and flexibility for future expansion or low-power applications. This design ensures that the ESP32-CAM can be easily programmed and debugged via USB, making it user-friendly and suitable for development and deployment in our application.

## 6.3.5 Camera Mounting:

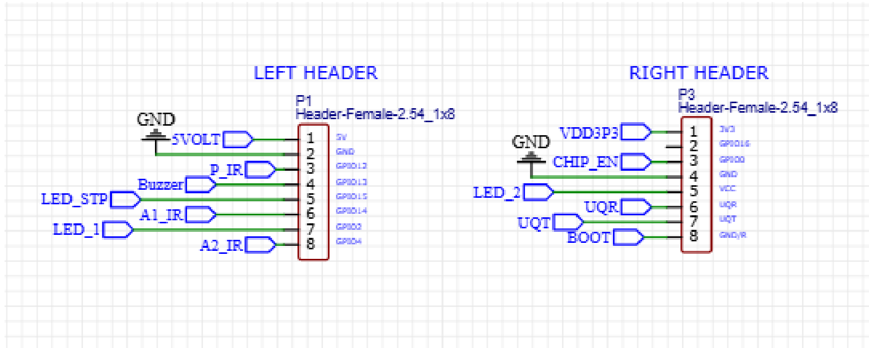



Figure 6.3.5: Camera Mounting

This is the schematic that offers a neat interface for connecting and functionality control of the ESP32-CAM with two headers. The Left Header (P1) is adapted to input/output devices like LEDs, buzzers, and infrared sensors. It provides direct access to GPIO pins and a stable 3.3V power source, which simplifies adding the peripherals into the system.

Right Header (P3) is used for control and auxiliary interfaces, providing connections to power the ESP32-CAM, enable the chip, trigger bootloader mode, and attach additional components such as status LEDs. In this way, modules can easily be added or prototyped for the system of the ESP32-CAM without adding a mess and allowing structural hardware to stay neat and organized.

## 6.3.6 Peripheral Connectors:

### 6.3.6.1 Passive IR:

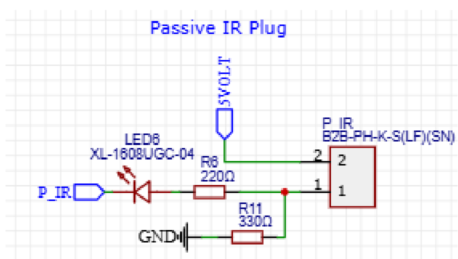



Figure 6.3.6.1: Passive IR

This is a PIR sensor plug interface designed to detect motion and feed the output signal to the ESP32-CAM PCB. The PIR sensor is connected through a modular 2-pin plug in a simple manner during attachment and replacement, either for installation or during prototyping. It includes a pull-down resistor on the signal line in order not to go into a floating state but provide a stable output.

An integrated LED provides immediate visual feedback on the detection of motion or active powering of the PIR sensor. As designed, the PIR sensor becomes user-friendly and reliable in motion detection for our applications. Equipped with power and stable signal output, this interface interactively works with ESP32-CAM for efficient motion-triggered functionality.

### 6.3.6.2 LED Strip:

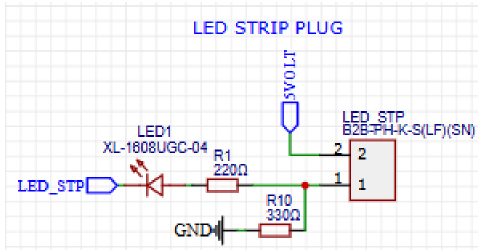



Figure 6.3.6.2: LED Strip

This LED strip plug interface connects the addressable LED strips for control through the ESP32-CAM PCB. A modular 2-pin plug simplifies integrating LED strips into our design, offering a neat and straightforward connection point for power and signal. A pull-down resistor stabilizes the signal line, ensuring proper operation and avoiding floating states during the time the controlling signal is inactive.

Moreover, a status LED on board provides immediate visual feedback on the state of connection for the LED strip or the activity of the control signal. It guarantees reliability and ease of operation, improving the functionality of the system using the ESP32-CAM.

**6.3.6.3 Active IR:**

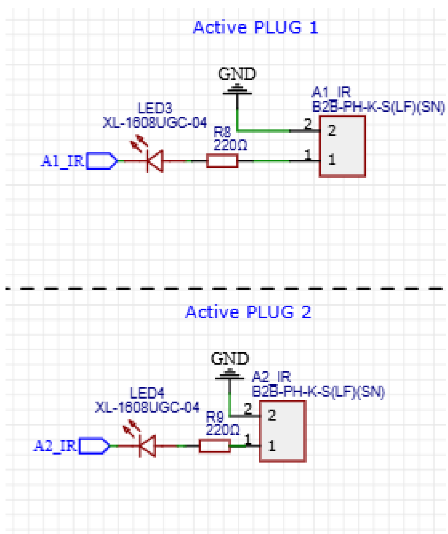



Figure 6.3.6.3: Active IR

This schematic describes two modular interfaces to connect active IR sensors to the ESP32-CAM PCB. Each interface includes a 2-pin plug, a status LED, and related current-limiting resistors for both functionality and visual feedback. The module's layout will allow for easy attachment and detachment of IR sensors for quick prototyping and testing.

Status LEDs are included, ensuring visual feedback of the operation of each IR sensor, thereby enhancing usability and simplifying troubleshooting. The listed interfaces work well for object detection, presence sensing, or motion tracking applications like ours. With such plugs integrated into the ESP32-CAM PCB, our design achieves a compact, user-friendly platform to interface multiple active IR sensors.

**6.3.6.4 Buzzer:**

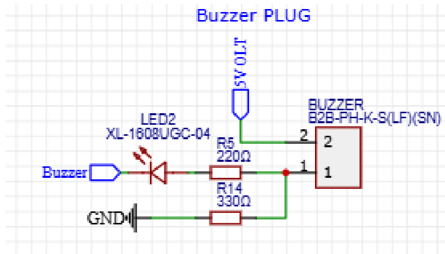



Figure 6.3.6.4: Buzzer Schematic

This is the interface for the Buzzer Plug. It will enable the integration of a buzzer module into the ESP32-CAM PCB. A 2-pin connector allows modular connection, making installation and replacement easier. A pull-down resistor provides stability within this circuit, and status is indicated by an LED, which visually shows when the buzzer is active.

This will be highly useful for our audible notifications, like IoT projects involving notifications, alarms, and feedback systems. The interface couples power and signal stability with visual feedback for reliable operation and an enriched user experience. This makes it a versatile component integrated with the ESP32-CAM.

## 6.3.7 Reset Buttons:

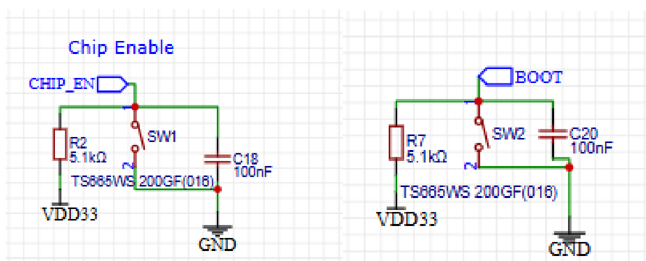



Figure 6.3.7: Reset Buttons

These CHIP_EN and BOOT circuits are responsible for some of the ESP32-CAM operation modes. CHIP_EN turns on/off the ESP32-CAM power supply and is a kind of master control for the operation of the module, while BOOT supports the entry into bootloader mode, updating of firmware, and system recovery.

Both circuits utilize pull-up resistors to stabilize logic levels and decoupling capacitors to eliminate noise, hence guaranteeing reliable switching. The functions are controlled straightforwardly by the user through manual switches, adding to the robustness and simplicity of operation. These further extend the flexibility and functionality of the ESP32-CAM in several ways, especially when programming, debugging, or deploying in embedded systems.

## 6.3.8 Testing LEDs:

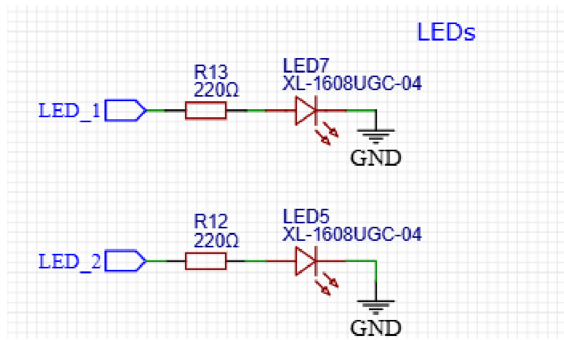



Figure 6.3.8: Testing LEDS

ESP32-CAM test indicators use LED indication circuit. It includes two LEDs with current-limiting resistors, ensuring that the LEDs are operating safely and effectively. These LEDs connect via GPIO and enable the ESP32-CAM to drive them for multiple purposes: to indicate the ON/OFF state of the system, for debugging, or to show system activity in general.

The use of standard 220Ω resistors ensures appropriate current levels are supplied to the LEDs for longevity. This simple structure is quite important in a number of embedded systems, whereby it assists both users and developers in monitoring states of systems or debugging. By mounting these LEDs onto the PCB, the ESP32-CAM platform acquires a reliable and user-friendly way of indicating statuses.

# Chapter 7 – Software Design

## 7.1 Software Design Introduction:

The software design chapter consists of detailed flowcharts, components/data structures, descriptions for our machine learning algorithm, class diagrams, and designs for the system and the communication protocols used. The primary function of our software inside the scope of our system is to provide effective interaction between our two main hardware components (Node & Gadget), interaction between our hardware and the camera/sensors, perform necessary data processing, and evaluate images captured from our ESP32-CAM (Node) using object detection algorithms through TensorFlow Lite. Shown in the subsections below, the system architecture is organized to manage several functionalities such as: Device and network initialization, data acquisition & storage, image processing using TensorFlow Lite, data transfer & communication, and interaction/processing sensor data through embedded code.

## 7.2 Software Design Flowchart:

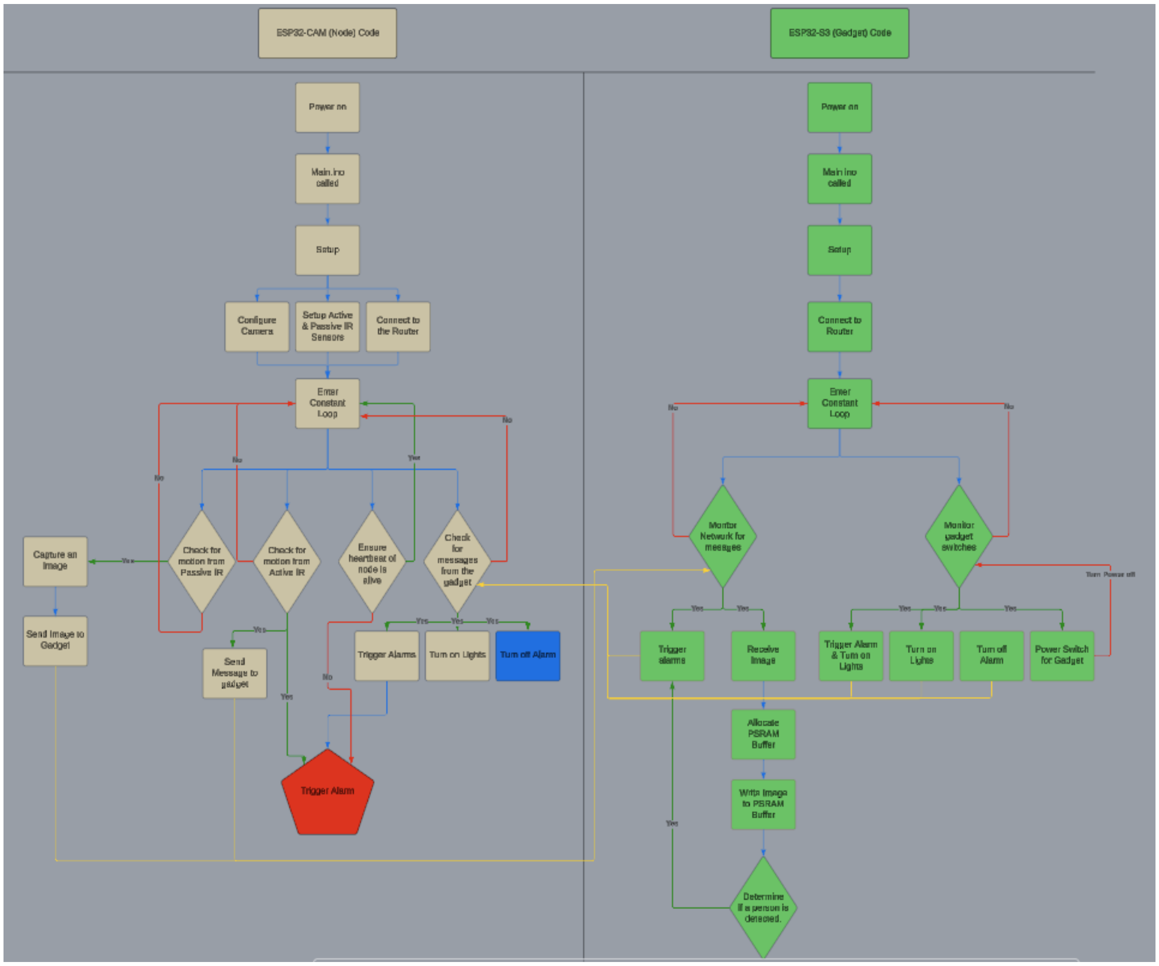


Figure 7.2: Software Design Flowchart created using Lucid Chart Software

## 7.2.1 ESP32-CAM Software Diagram Flowchart Explanation:

The ESP32-CAM Software Diagram can be seen on the left side of Figure 7.2. Both the ESP32-CAM and the ESP32-S3 were included in the same diagram to show how interconnected they were. Throughout the architecture of the software there are multiple instances where message protocols will need to be sent from one to the other. The Diagram for the ESP32 starts with the power being turned on, once this happens, main.ino is run. The first thing that runs in main.ino is setup() is called. Inside of our setup() function we do a couple things, first we configure the camera, next we set up the Active & Passive IR sensors to their correct configuration, and lastly, we connect the ESP32-CAM board to the router that is located on the gadget. After this, we enter inside of a constant loop() function that will run while the power is on. Our constant loop function is designed so we have a few functionalities we wish to have in our system that will always be checked for. First being if motion has been detected from the Passive IR sensor. In the case it has, we will capture an image and send the image to the gadget (will cover what happens from there in 7.2.2). Another functionality in our loop() is we are constantly checking for motion in the Active IR sensor. If motion has been detected, we send a message to the gadget and trigger the alarm. The third functionality we have is the heartbeat of our system, we will constantly monitor this heartbeat to ensure that our system has not been tampered with, if so, we will trigger the alarm. Last functionality inside of our loop(), is we will constantly monitor the network for messages from the gadget. We will be checking for a message for turning off the alarm, triggering the alarm, or turning off the lights.

### 7.2.2 ESP32-S3 Software Diagram Flowchart Explanation:

The ESP32-S3 Software Diagram can be seen on the right side of Figure 7.2. The diagram begins with the power being turned on, in our case by a switch on the gadget. Once the power is on, similarly as to the ESP32-CAM, main.ino will run and the setup() function will be called. In our setup function, we will connect/ ensure connection from the gadget to the router. We also will be configuring the necessary hardware for the gadget such as reading voltage changes for the switches. After this, again like our node code, we will enter into a constant loop that will monitor two things: Check network for messages and monitor switches from the gadget. There are two cases for receiving a message on the gadget. First being to trigger the alarm, meaning we have a threat, and we will all nodes' alarms to be triggered. The second case is when we receive an image. Upon receiving an image, we allocate the necessary memory for the image and pass our image into our model that we trained to perform object detection on. In the case we have detected a person (threat), we will send a message to all the nodes to trigger the alarm. The other functionality in our constant loop is checking for voltage changes from the gadget switches. There are 4 switches: Trigger alarm & turn on lights, turn on lights, turn off alarm, and power switch to turn off/on the gadget. If one of these switches has been flipped, their respective messages will be sent over to the nodes and perform the desired functionality.

## 7.3 Software Use Case Diagram:

The use Case Diagram can be seen below in Figure 7.3. The diagram is very simple as we have very few interactive features in our system. We designed this to ensure that our users will have safety without having to monitor the area themselves. The two interactive areas are the mobile app and the gadget. With the mobile app, we can request images from all the nodes, we can turn off the alarm and trigger the alarm. Finally, we have a feature where we can receive alarm notifications from the system when the alarm has been triggered. With the gadget, we can trigger the alarm, turn on the lights, turn off the alarm, and then we have a power switch for the gadget as well.

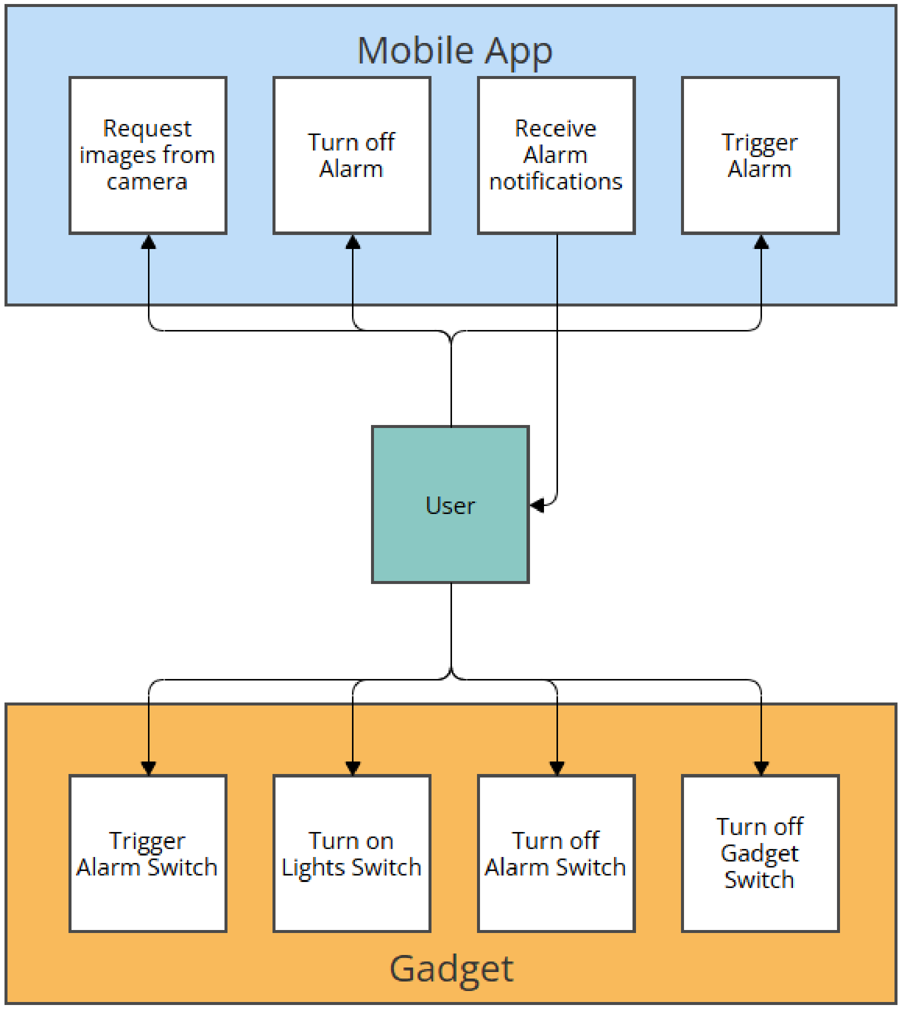



Figure 7.3: Use Case Diagram created using Lucid Chart Software

## 7.4 Software State Diagram:

The Software State Diagram can be seen below in Figure 7.4. Our software can be generally split up into 2 different sections for most diagrams, Node and Gadget code. However, the architecture is the exact same for the two sections as far as what state our code can live in. We have 3 states that our software can live in: Power off, setup phase, and monitoring the system. We start off with power being off. Once we have turned the power on, we will now be in the setup phase. The setup phase consists of connecting the respective system to the router, and then setting up all hardware peripherals. Once setup is complete, we move to monitoring the respective system. Monitoring will run a constant loop, hence the arrow towards itself. Inside of this state we will monitor the hardware peripherals and the network for each respective system. The only way to get the power off after this phase has begun is to turn the power off.

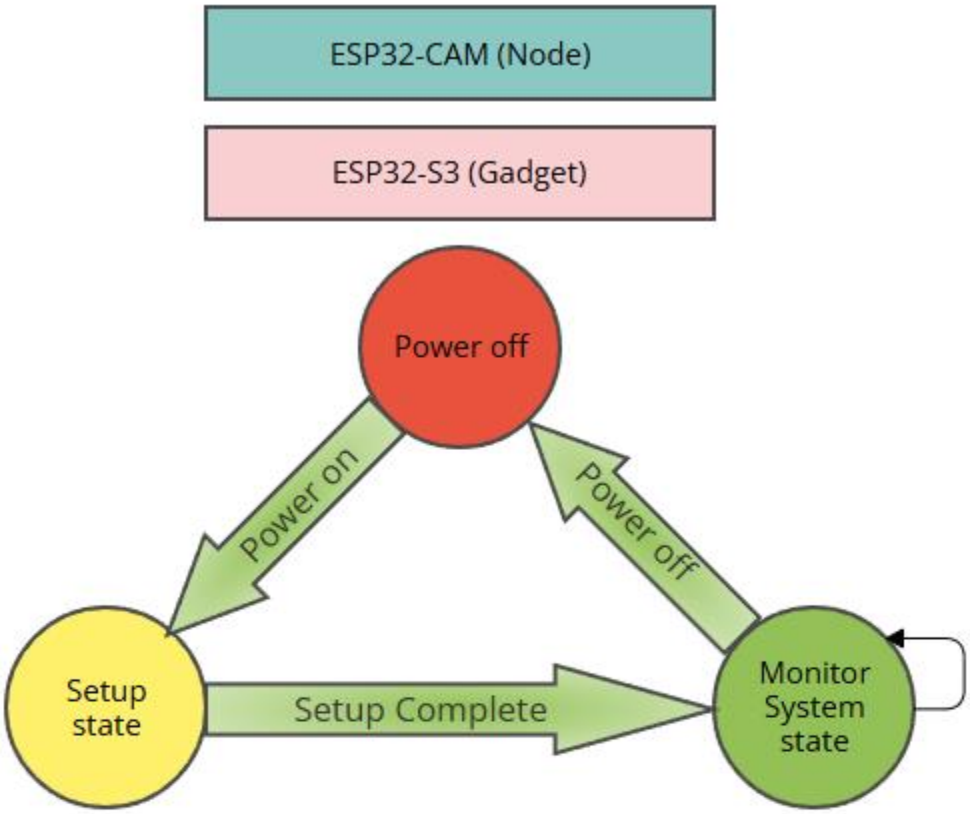



Figure 7.4: Software State Diagram

## 7.5 Back-End Software Structure Class Diagram:

Our software structure class diagram can be seen below in figure 7.5. For our ESP32-CAM (Node), we only have one class (file) that we will be putting onto the microcontroller, named main.ino. Inside of main.ino, as seen in the figure, are all the different functions that have been created, each serving its own purpose in regard to meeting the objective for each board's functionality. Inside the class there are the built in Arduino ide functions setup() and loop(). Setup() is called as soon as power hits the board and is intuitive in the fact that it sets up everything for us such as our camera configuration, hardware peripherals, and connects the board to the network. Loop() is then run right after setup and has the same functionality of a while loop, constantly running until the power gets turned off. The other functions we have in our code serve the purpose of assisting the loop() call. They are captureAndServeImage(), the 3 handleJpg() function's, checkAlarmNotification(), checkForTurnOnLights(), checkForTurnOffLights(), and triggerAlarm(). All these functions names are exact descriptions of their purpose and are intuitive except for handleJpg, these functions handle the image at a different resolution and ensure camera configuration is setup properly, called at the end of setup().

For our ESP32-S3 (Gadget), there are 4 different files that will be uploaded to our board. First will be main.ino, where the code will run from. Second is ObjectDetection.cpp, this will be called from main.ino. And last is the model that will be instantiated (person_detect_model_data.cpp) inside of ObejctDetection.cpp. Main.ino has the two built in Arduino ide functions, loop() and setup(), both serving the same purpose as they do on the nodes, just different functionality that is required out of the gadget. Setup() will connect our gadget to the router and set up all of the hardware peripherals. Loop() will monitor the router's network for any messages from the node or from the mobile app, and also monitor the switches located on the gadget for any voltage changes. Please keep in mind that not all functions have been developed on the gadget for serving the functionality required of the hardware so there will more than likely be a few more functions in main.ino that help that out. The next function in main.ino is handleImageUpload() that will allocate memory for the image sent over the network via psram buffers and then call detectPerson() inside of ObjectDetection.cpp. Moving over to ObjectDetection.cpp, detectPerson() will be passed in an image and its size and will invoke that image onto the model that we will have initialized in there. DetectPerson() at the beginning of the function calls initializeModel(), that simply instantiates the model from person_detect_model_data.cpp. After that, it does some model pre-processing, fine tuning, and allocation for our tensor arrays. For a more detailed explanation of the algorithms and methods used for the image processing, please see section 7.11 Image Processing Algorithm Description.

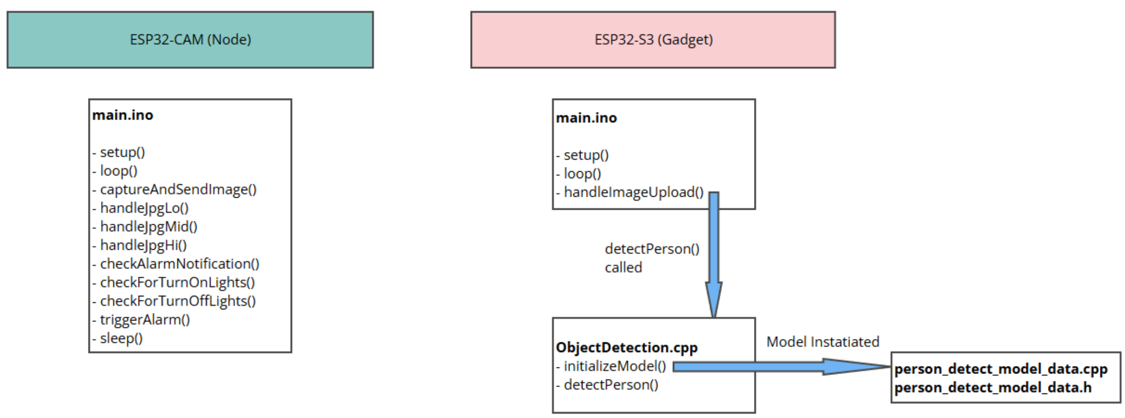



Figure 7.5: Back-End Software Class Diagram

## 7.6 Front-End Software Structure Class Diagram:

Our frontend structure class diagram for the SecureScape mobile app, shown in Figure 7.6, provides a comprehensive visualization of how the user interface (UI) components are structured and interconnected. At the core of the design is the BasePage class, an abstract class that ensures consistency across all pages by providing shared functionality and styling. This foundational class includes common attributes such as title, which defines the page's heading, and backgroundTheme, which determines the visual design. Additionally, it contains essential methods like render() for displaying content and navigateTo(destination: Page) for handling transitions between pages, promoting a standardized user experience. The Router class complements this design as a utility component, offering navigation between pages through its navigateTo(page: Page) method, allowing users to access different app features.

Each specific page—WelcomePage, AboutUsPage, ConnectingPage, HomePage, and ViewPhotoPage—inherits from the BasePage and implements its own specialized functionality to meet the app's needs. For example, the WelcomePage provides intuitive buttons for pairing the SecureScape device and accessing the AboutUsPage, while the ConnectingPage manages the pairing process with attributes like statusMessage and methods such as startPairing() and onPairSuccess() to guide users through the setup. The HomePage acts as the central hub, consolidating core functionalities like triggering alarms, capturing photos, and viewing logs, while the ViewPhotoPage allows users to manage images captured by the SecureScape system, with features for viewing and saving images. The NotificationBanner class operates independently to handle real-time alerts, ensuring users stay informed of critical events by displaying actionable notifications that integrate seamlessly with the HomePage. This modular and scalable design guarantees consistent user experience while maintaining the flexibility needed for future enhancements and feature additions.

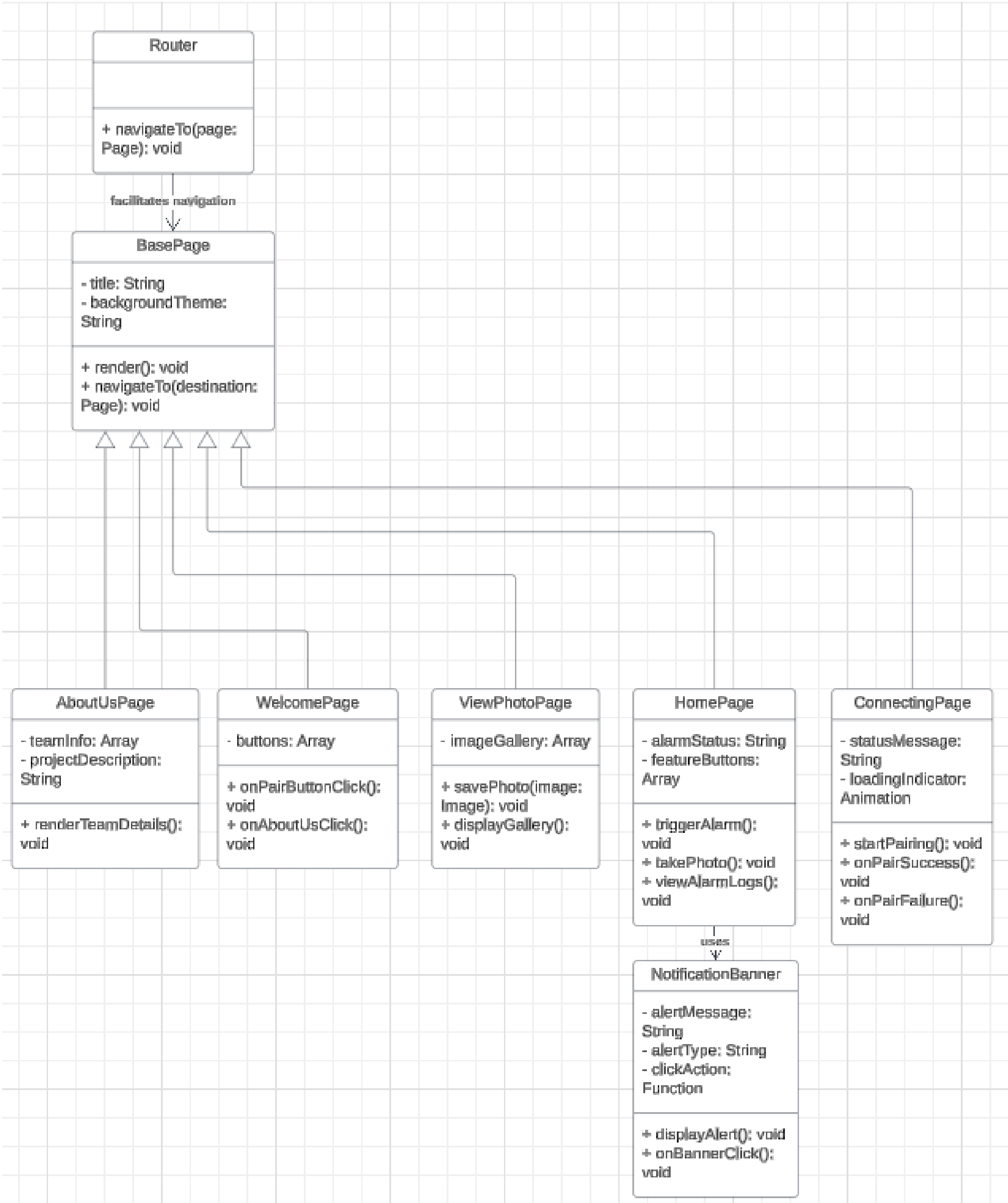



Figure 7.6: Frontend Class Diagram

# 7.7 Software Data Structures:

## 7.7.1 Data Structures Introduction:

In this section we will be covering any and all primitive data types and complex data structures between the Node and Gadget. Our Software on both the node and the gadget did not require many complex data structures to meet the requirements of our project's goal due to the abstracted architecture created by Jaxon and Kirby. Covered in section 7.7.2 and

7.7.3 you will find all primitive data types used as well as the complex data structures and functionalities provided by the TensorFlow Lite Micro Library.

### 7.7.2 ESP32-CAM (Node) Data Structures:

Inside of our ESP32-CAM, there is one file named main.ino. While there are primitive data types used in this file, to meet the requirements of our system, no complex data structures were required. That being said the primitive data types used were integers, strings, Booleans, and pointers. Integers were stored for pin numbers, resolution values, loop counters, and http response codes. Strings were used for storing the SSID and password for connecting to the Wi-Fi, URLs, and filenames for writing a specific resolution to the network. Booleans were used for checking the status of connecting to the Wi-Fi and for monitoring the state of our sensors for checking hardware peripherals. Pointers were used temporarily in the captureAndSendImage() function to allocate memory for the captured image. There are however complex data structures used with our built-in functions that have been abstracted for us such as capturing the image from the camera.

### 7.7.3 ESP32-S3 (Gadget) Data Structures:

Inside of the ESP32-S3, there are two main files that are run, main.ino and ObjectDetection.cpp. Inside of main.ino, similar to main.ino from the node, there are no complex data structures used. The primitive data types used in this file however are Strings, Integers, Booleans, and pointers. We have Strings for storing the Wi-Fi credentials for the SSID and the password. Integers used for counters and for allocating memory in the handleImageUpload() function. We use booleans when allocating memory for PSRAM when checking for availability and also to store the results of detectPerson() located inside of ObjectDetection.cpp. Lastly, we use pointers to store the uploaded image if we have memory for the image. In our ObjectDetection.cpp file, while we do not create any complex data structures, we take advantage of many of the complex libraries included inside of the TensorFlow Lite Micro Library. This being our only file containing any complex data, the primitive data types in this model such as Arrays, Pointers, Integers, and Booleans will not be covered with detail. The TensorFlow Lite Micro data structures used inside of the .cpp file are the MicroInterpreter, Tensors, Model, and the MicroErrorReporter. The MicroInterpreter is the main object that manages the TensorFlow Lite model’s execution. It allocates memory for tensors, invokes the model, and provides access to input and output tensors. The tensor stored is a multidimensional array that stores data, shape, data type, and all the quantization parameters. The model represents the actual model itself, containing the architecture of the model, operations, and parameters. Lastly, the MicroErrorReporter is used to report errors and warning during the model's execution.

With everything together, as you have just read, our system is very complex, and most components held within are tightly interconnected. We designed our software to not use many complex data structures to save power consumption through both minimizing computation time and memory storage.

## 7.8 Software User Interface Design:

### 7.8.1 User Interface Introduction:

In this section, we will detail the design and implementation of the SecureScape mobile app's user interface (UI). The goal of our UI is to create an intuitive and user-friendly experience, ensuring seamless interaction between the user and the security system. The design prioritizes ease of navigation, simplicity, and accessibility, reflecting SecureScape’s purpose.

The app’s UI consists of multiple interconnected pages that guide users through key functionalities, including device setup, image requests, alarm controls, and notifications. The logical flow between these pages was planned to ensure that users could quickly access essential features without unnecessary complexity. Covered in sections 7.8.2 through 7.8.6, we will break down the design decisions and features of the Welcome Page, Connecting Page, Home Page, Photo Page, and Notification Page, each of which contributes to the overall user experience.

The SecureScape app was designed with simplicity and efficiency in mind, making it accessible to all users, regardless of technical expertise. Each page and feature were carefully crafted to provide clear guidance and immediate feedback, ensuring a smooth and engaging interaction with the app.

### 7.8.2 Welcome Page:

The Welcome Page is the entry point of the SecureScape app and serves as a welcoming and informative interface for users. It prominently features the title “Welcome to SecureScape” alongside a visually engaging background of an outdoor scene. Below the title, a motivational subtitle, “Take Control of Your Safety,” guides users toward the app’s purpose. The page provides two clear options: a “Pair Your SecureScape” button, which directs users to the Connecting Page to begin the device setup process, and an “About Us” button, which leads to an informational page about the app and its development team. The layout prioritizes simplicity and accessibility, with large, tappable buttons that ensure ease of navigation. The Welcome Page effectively establishes the app’s branding while offering an intuitive starting point for users to connect their device or learn more about SecureScape.

### 7.8.3 Connecting Page:

The Connecting Page facilitates the pairing process between the SecureScape app and the user’s device, ensuring a seamless transition to full functionality. At the top, a loading animation provides real-time feedback, reassuring users that the connection process is underway. Below the animation, a clear and concise prompt, “Ensure your SecureScape device is powered on and located nearby for connection,” provides guidance on the necessary conditions for successful pairing. The layout maintains a clean and visually engaging design consistent with the app’s overall theme, featuring the same outdoor background for continuity. This page is designed to minimize user confusion while keeping the process straightforward and visually appealing.

### 7.8.4 Home Page:

The Home Page serves as the central hub for the SecureScape app, offering users immediate access to the system's core functionalities. At the top, a status indicator provides real-time feedback on the device connection, ensuring users are always aware of their system's operational state. Below this, the page features three primary buttons: "Trigger Alarm," "Take Photos," and "View Alarm Logs." Each action is accompanied by an intuitive icon and large, clearly labeled buttons for easy navigation and accessibility. A toggle switch is provided for activating or deactivating the alarm state, allowing users to manage security settings effortlessly. The layout is designed with simplicity in mind, minimizing visual clutter and maintaining the app's consistent outdoor theme to enhance user engagement.

### 7.8.5 View Photos Page:

The View Photo Page provides users with access to images captured by SecureScape's deployed nodes, ensuring efficient monitoring and data management. At the top, the page maintains a connection status indicator to reassure users of the system's functionality. Below this, the page features an Image Gallery section, organized by nodes (e.g., Node 1, Node 2, Node 3), where placeholder icons represent available images. This intuitive layout allows users to easily navigate and identify images captured from specific nodes. A prominent "Save Photos" button is provided for users to download and store selected images locally, ensuring critical data is easily accessible. The View Photo Page is designed to facilitate straightforward access to and management of captured images.

### 7.8.6 Notifications:

The notification banner in the SecureScape app is designed to deliver real-time alerts with clarity and immediacy, ensuring users are promptly informed of critical events. Positioned at the top of the screen, the banner provides key details such as the nature of the alert (e.g., "Alarm Triggered!") and actionable information (e.g., "Node #1 Noticed Movement. Click Here…"). Notifications are actionable, allowing users to directly respond to the alert by tapping the banner to navigate to relevant features or further details. The notification banner ensures that users remain informed and can quickly react to potential security events.

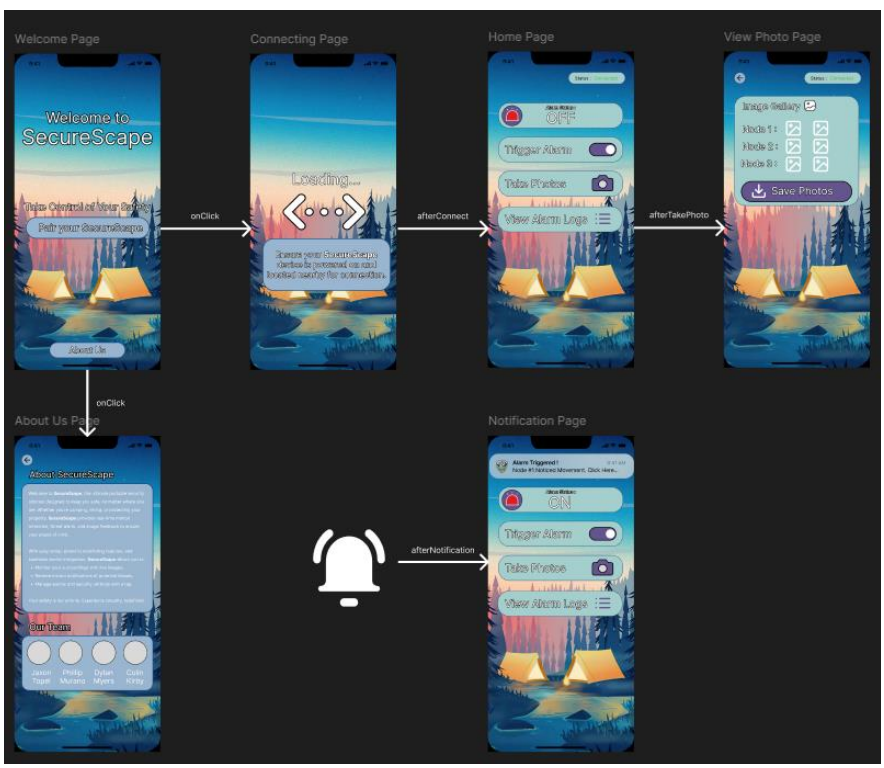



Figure 7.8: UI Design Prototype #1

## 7.9 Software Data Transfer & Communication Protocols:

### 7.9.1 Introduction:

The SecureScape system is built on efficient and reliable communication protocols that facilitate data transfer between its hardware components, the mobile app, and the cloud infrastructure. These protocols were selected to ensure minimal latency, optimized resource usage, and secure transmission of critical information, even in areas with limited connectivity. The following sections outline the data transfer methods and protocols used for communication between the ESP32-CAM nodes, the ESP32-S3 gadget, and the mobile app, as well as the rationale for their selection and implementation.

### 7.9.2 Node-To-Gadget Communication:

The ESP32-CAM nodes communicate with the ESP32-S3 gadget using HTTP, a lightweight and widely supported protocol. When motion is detected, the nodes capture an image, which is sent to the gadget via an HTTP POST request. Each request includes metadata, such as the node ID, timestamp, and image resolution, to ensure proper categorization and traceability. The gadget acts as a central hub, taking data from multiple nodes and preparing it for further processing or transmission to the mobile app.

### 7.9.3 Gadget-To-App Communication:

The mobile app communicates with the gadget through a combination of Firebase and HTTP. Firebase handles real-time synchronization, ensuring users receive immediate notifications of detected threats. HTTP GET and POST requests enable the app to retrieve data such as logs, images, and alerts. Firebase Cloud Messaging (FCM) is used for delivering notifications, providing users with live updates and enabling them to respond to events promptly.

### 7.9.4 Protocol Selection and Security Measures:

The SecureScape system utilizes HTTP and Firebase as its primary communication protocols, chosen for their compatibility with ESP32 microcontrollers and their ability to efficiently handle lightweight data packets. HTTP simplifies tasks such as image uploads and device-to-device communication, while Firebase ensures robust real-time synchronization and user notifications.

To safeguard data during transfer, SecureScape employs HTTPS for encrypted communication between devices, ensuring all transmitted data is protected against interception. Authentication tokens and unique device identifiers are implemented to prevent unauthorized access to the system. Additionally, hash-based verification is used to maintain data integrity, ensuring that the information received is accurate and untampered. These protocols and security measures collectively provide a secure and reliable foundation for the system’s data transfer and communication processes.

### 7.9.5 Conclusion:

The communication protocols used in the SecureScape system provide a reliable framework for data transfer between the ESP32-CAM nodes, the ESP32-S3 gadget, and the mobile app. By utilizing HTTP and Firebase, the system supports critical functionalities such as image uploads, real-time notifications, and data synchronization, ensuring users monitor and respond to events effectively as necessary for the desired applications of SecureScape. Security measures, including HTTPS encryption, authentication tokens, and data integrity checks, protect transmitted data from interception and tampering. This design ensures the system operates efficiently, maintains data accuracy, and meets the requirements of a portable and adaptable security solution.

## 7.10 Image Processing Algorithm Description:

### 7.10.1 Introduction & Architecture

The Image Processing Algorithm for our system is really the bread and butter of what makes this such a good smart security product. For clarification, the image processing will happen on the ESP32-S3 in a file called ObjectDetection.cpp. This file will be called from main.ino (Gadget), specifically it is calling the function detectPerson(). In ObjectDetection.cpp, we will be taking advantage of the TensorFlow Lite Micro Library to run a 250kb Convolutional Neural Network to recognize people in images captured by the camera on our ESP32-CAM. The flow of our image processing algorithm is as follows. Once detectPerson() has been called, it immediately calls initializeModel(). Inside of initializeModel(), we first initialize our error reporter, and then we load our model using GetModel() method provided by TensorFlow Lite. We pass in the model as a data array and GetModel() maps the model into a usable data structure in a manner that keeps this a lightweight operation. Next, we initialize the MicroMutableOpResolver with a capacity of 5 custom operations. MicroMutableOpResolver is a class in the TensorFlow Lite Micro framework that allows us to register custom operations that are not part of the standard TensorFlow Lite Kernel Set. We do these custom operations to increase the performance and capabilities of our model. The 5 custom operations are as follows AddAveragePool2D(), AddConv2D(), AddDepthWiseConv2D(), AddReshape(), and AddSoftmax(). Continuing inside of the initializeModel() function, after performing all 5 of those operations on the MicroMutableOpResolver we then build an interpreter to run the model with. After this we then allocate memory for the models' tensors and return to the detectPerson() function. Back in the detectPerson() function, after initializing the model we then perform some image preprocessing such as resizing the image to match the first input layer of our model. We then run the input image through the model and store our output if the output is greater than 0.5, we return true indicating a person has been detected, false otherwise (less than 0.5).

### 7.10.2 Convolutional Neural Network Architecture

#### 7.10.2.1 Average Pooling Operation for 2-Dimensional Spatial Data:

AddAveragePool2D() serves the purpose of reducing spatial dimensions of the input feature maps by computing the average of pixels in a rectangular region.

$$Y[i,j] = \frac{1}{k_h \cdot k_w} \sum_{m=0}^{k_h-1} \sum_{n=0}^{k_w-1} X[i \cdot s_h + m, j \cdot s_w + n] \qquad (7.10.2.1)$$

Where,

- Y[i,j] = The output pixel value at row i and column j of the pooled feature map.
- X[i · S_h +m, j · S_w +n] = The value of the input pixel.

- K_h = Height (number of rows) of the pooling window (kernel).
- K_w = Width (number of columns) of the pooling window (kernel).
- S_h = Vertical stride, the number of rows the pooling window moves down after each step.
- S_w = Horizontal stride, the number of columns the pooling window moves right after each step.
- m = Row index within the pooling window, iterating from 0 to K_h – 1.
- n = Column index within the pooling window, iterating from 0 to K_w - 1

**7.10.2.2 Convolution:**

AddConv2D() is another operation used in our CNN architecture. This operation extracts features from the input data by applying filters. The process of this operation is as follows, a filter is slid over the input feature map and at each position the filter's weights are multiplied by the corresponding input values, and then the results are summed. After this, the sum is then passed through an activation function, in our case ReLu to produce the output value.

$$\text{output}[i,j] = \text{bias} + \sum_{k=0}^{K_h-1} \sum_{l=0}^{K_w-1} \text{filter}[k,l] \cdot \text{input}[i+k, j+l]] \qquad (7.10.2.2)$$

Where,

- output[i, j]: The result of the convolution operation at position (i,j) in the output feature map.
- bias: A learnable parameter added to the output to increase flexibility.
- filter[k, l]: A filter weight at position (K,l), used for feature extraction.
- input[i + k, j + l]: The input value corresponding to the filter's position.
- K_h and K_w: Height and width of the filter.

**7.10.2.3 Depthwise Convolution:**

The next operation in our CNN is AddDepthWiseConv2D(). This operation applies another filter to each input channel, reducing the number of parameters compared to standard convolution. In this operation each filter is applied to its corresponding input channel, producing a new output channel. The output channels are then concatenated to form the final output.

$$\text{output}[i,j,k] = \text{bias}[k] + \sum_{l=0}^{K_h-1} \sum_{m=0}^{K_w-1} \text{filter}[k,l,m] \cdot \text{input}[i+l, j+m, k]] \qquad (7.10.2.3)$$

Where,

- output[i, j, k] = the value of the output feature map at row i, column j, and channel k. This is the result of applying the depthwise convolution for a specific channel.
- bias[k] = scalar bias term specific to channel k.
- filter[k, l, m] = the weight of the depthwise filter for channel k, at row l and column m. Each channel has its own filter, which is applied to the corresponding input channel.
- input[i + l, j + m, k] = value of the input feature map at row i + l, column j + m, and channel k.

- K_h and K_w = the height and width of the depthwise filter.

**7.10.2.4 Softmax:**

The next operation in the CNN is AddReshape(), which serves the purpose of changing the shape of a tensor without changing its data. And finally, the last operation inside of our CNN architecture is AddSoftmax(). This operation converts a vector of numbers into a probability distribution, hence why it is the last operation as we need our output of the model to be a number from 0 to 1.

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{N} \exp(x_j)} \qquad (7.10.2.4)$$

Where,

- softmax(X_i) = ith element of the output vector
- X_i is the ith element of the input vector

# Chapter 8 – System Fabrication/Prototype Construction

## 8.1 System Fabrication/Prototype Construction Introduction:

Our system's prototype will include our gadget and one node. We will test the compatibility of both hardware and software as well as the connectivity. The hardware section will be tested over the breadboard in tandem with the software. There is CAD imagery to help the reader understand our planning process and where each item will be placed in the node and gadget housing.

## 8.2 Breadboard Prototype:

For the end of SD1 we were required to make and display a breadboard prototype. This allowed us to test the various functions that we wanted our project to have. We strived to test more functionality than was required and pushed to get a fully functional node breadboard and LAN set up as well as a functional gadget.

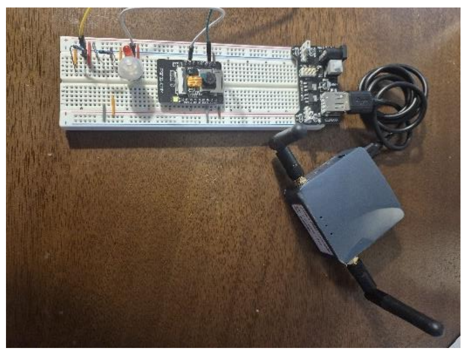


Figure 8.2: Breadboard Prototype

### 8.2.1 Node:

The cross section of the base of the node is shown below. There will be a small cavity inside the base. There will be an insert which will slide in and be screwed down. There will be a small layer outside the inner circle to provide a protective layer and create depth so that the node is not surrounded by a thin film. There will be a section allocated at the bottom of the node for our COB LED strip to attach to. We want the strip to be at the bottom because it is better for consolidating our circuitry and wiring and will not blind the ESP32 CAMs during operation.

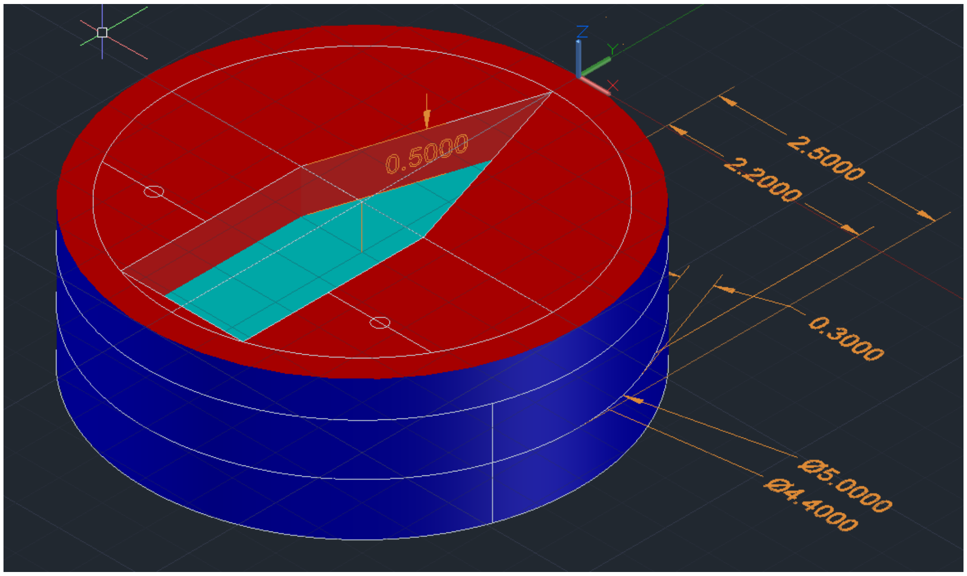



Figure 8.2.1.1: Node Diagonal Cross Section

The next figure shows how the node will look when completely surrounded by its protective wall. There will be a cut out for the cams so that it can see outside the housing unit. Each node houses two ESP32-CAM MCUs which will be placed on the green figure below on both of the angled planes to cover the largest amount of viewing angle possible without creating blind spots. The top is also shown separately because we want to easily access the insides for testing and troubleshooting. We have excess space allocated at the underside of the node for us to attach PVC piping. The plan is to have an insert into the pipe to hold our batteries, and have wires go through the pipe into the base of the node to reach the rest of the system.

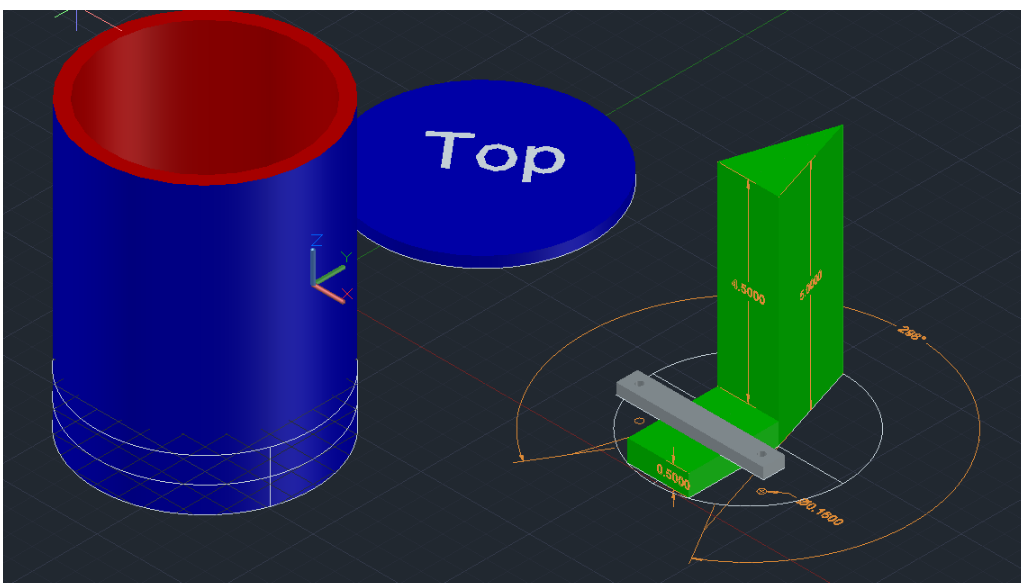



Figure 8.2.1.2: Node Outer View with Insert

**GPIO Connections**

| Component | ESP32-CAM GPIO Pin | Purpose |
|---|---|---|
| IR Sensor 1 (EK8460) | GPIO 14 | Motion detection input |
| IR Sensor 2 (EK8460) | GPIO 13 | Motion detection input |
| IR Sensor 3 (AM312) | GPIO 12 | Motion detection input |
| LED for Sensor 1 | GPIO 4 | Visual indicator for Sensor 1 |
| LED for Sensor 2 | GPIO 5 | Visual indicator for Sensor 2 |
| LED for Sensor 3 | GPIO 16 | Visual indicator for Sensor 3 |
| Buzzer | GPIO 15 | Audible alert output |
| Camera PWDN | GPIO 32 | Power down control |
| Camera XCLK | GPIO 0 | External clock signal |
| Camera SIOD | GPIO 26 | Data (I2C) |
| Camera SIOC | GPIO 27 | Clock (I2C) |
| Camera D0 - D7 | GPIO 5, 18, 19, 21, 36, 39, 34, 35 | Camera data pins (Y2 - Y9) |
| Camera VSYNC | GPIO 25 | Vertical sync signal |
| Camera HREF | GPIO 23 | Horizontal reference signal |
| Camera PCLK | GPIO 22 | Pixel clock |

Table 8.2.1.1: Node GPIO Connections

**Power Connections**

| Component | Voltage Source | Breadboard Rail | Comments |
|---|---|---|---|
| ESP32-CAM | 3.3V | 3.3V rail | Connect to 3.3V output from power module |
| EK8460 IR Sensors (x2) | 5V | 5V rail | 5V VCC; GND to ground rail |
| AM312 IR Sensor | 3.3V | 3.3V rail | 3.3V VCC; GND to ground rail |
| HiLetgo Power Supply Module | 4V lithium battery pack / outlet | --- | Provides both 3.3V and 5V outputs |
| LEDs (x3) | 3.3V via GPIO | Ground rail | LEDs connected via resistors to GND |
| Buzzer | 3.3V via GPIO | Ground rail | GND connection to ground rail |

Table 8.2.1.2: Node Power Connections

## 8.2.2 Gadget:

In the gadget our general housing unit will look like a rectangular block as shown in the image below. There will be a router attached to the top of the block. The batteries will be set at the bottom and will be easily accessible. The PCB is seated between the batteries and the router. Each section is given some wiggle room to ensure that everything will fit, and can be easily filled with spacers.

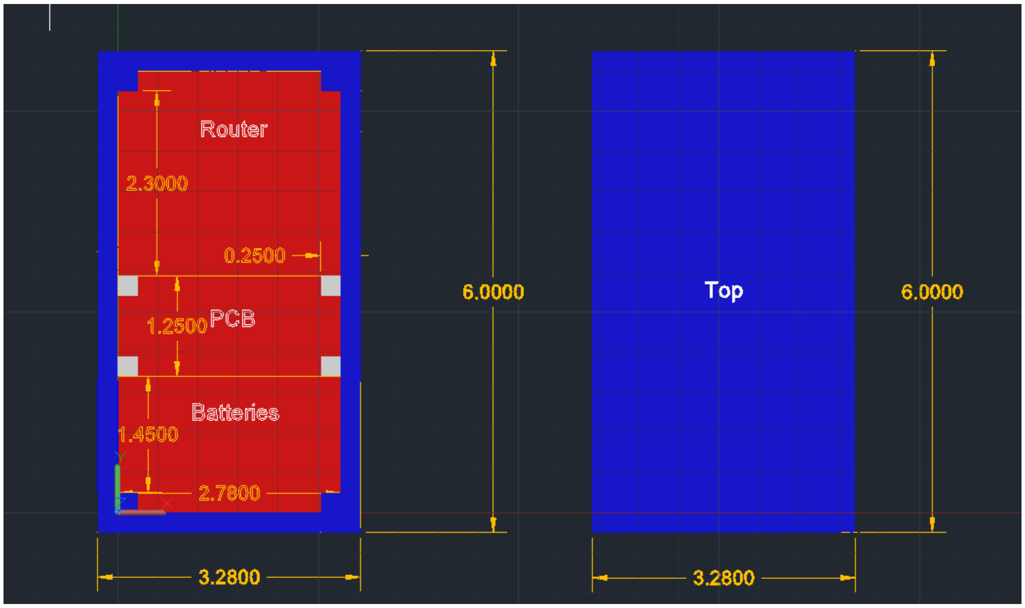



Figure 8.2.2.1: Top-Down Cad Drawing of the Gadget

The router seated near the top of the gadget will have holes for the antennas to stick out of. In this angle, it is now possible to see the extensions for the PCB section to make it easier to install. The dimensions shown have all been slightly increased to ensure that everything fits with some room to spare. After the installation of the antennas, we plan to fill the gaps with a cut out to prevent water and dust from entering.

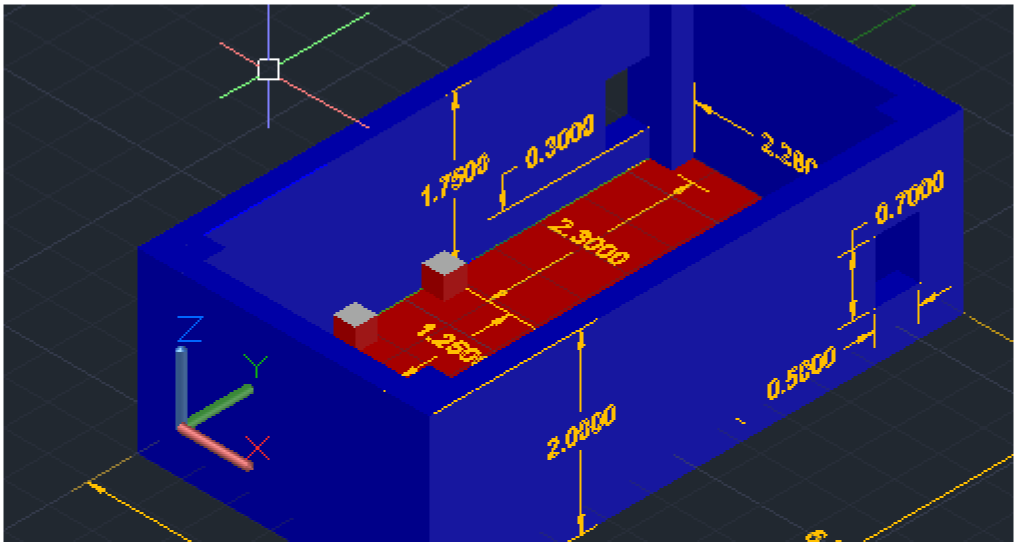



Figure 8.2.2.2: Side-view Cad drawing of the Gadget

**GPIO Connections**

| **Component** | **ESP32-S3 GPIO Pin** | **Purpose** |
|---|---|---|
| Switch 1 | GPIO 14 | Digital input for switch 1 |
| Switch 2 | GPIO 13 | Digital input for switch 2 |
| Switch 3 | GPIO 12 | Digital input for switch 3 |
| LED 1 | GPIO 4 | Visual indicator for LED 1 |
| LED 2 | GPIO 5 | Visual indicator for LED 2 |
| LED 3 | GPIO 16 | Visual indicator for LED 3 |
| Website Server Control | GPIO 15 | Hosts server for website access |

Table 8.2.2.1: Gadget GPIO Connections

**<u>Power Connections</u>**

| **Component** | **Voltage Source** | **Breadboard Rail** | **Comments** |
|---|---|---|---|
| ESP32-S3 | 3.3V | 3.3V rail | Connect to 3.3V from power module |
| Switches (x3) | 3.3V | 3.3V rail | Connect VCC to 3.3V, GND to ground rail |
| LEDs (x3) | 3.3V via GPIO | Ground rail | Use current-limiting resistors between GPIO and LEDs |
| Power Supply Module | 5V input | --- | Powers both 3.3V and 5V rails as needed |

Table 8.2.2.2: Gadget Power Connections

# Chapter 9 – System Testing and Evaluation

## 9.1 System Testing and Evaluation Introduction:

In this chapter we will include our testing results in a separate manner. The hardware sections will test part compatibility and cohesion while the software sections will test the algorithms implemented for image processing, and the network connectivity for multiple end points. We have set expectations for specific outcomes and will label each section with comments to give more information when needed. There will be a checklist linked to our testing plan to ensure proper testing occurs.

## 9.2 Hardware Testing:

Our testing for the Hardware design is broken up into two parts, Node testing and Gadget testing. This is due to these systems working independently of each other. While they do communicate and work together. Where hardware is concerned, they don't interact.

### 9.2.1 Node Test:

The node is tested on the sensors equipped, its communication with the LAN, and the response we observe (alarm and video). This test plan and failure checklist is designed to be used with both our bread board and our final PCB Build.

| Test ID | Test Case | Expected Outcome | Pass/Fail Criteria | Pass/Fail | Comments |
|---|---|---|---|---|---|
| 1 | Power-On Test | ESP32-CAM powers up, connects to Wi-Fi, and starts camera stream. | Camera feed accessible. | PASS | Cam successfully connect to Wi-Fi, and can access camera stream from ESP's IP. |
| 2 | Sensor Detection Test - EK8460 | Each EK8460 sensor should activate its assigned LED when movement occurs. | LED turns on when motion is detected in each EK8460 sensor field. | PASS | LED turns on during movement |
| 3 | Sensor Detection Test - AM312 | The AM312 sensor should activate its assigned LED when movement occurs. | LED turns on when motion is detected in AM312 sensor field. | Test in Progress for SD2 | Implementation in SD2 |
| 4 | Buzzer Activation Test | The buzzer should sound briefly when any sensor detects movement. | Buzzer activates for each motion detection event. | PASS | Buzzer sounded during motion |
| 5 | LED Response Test | LEDs for each sensor respond only to respective sensor activation. | Correct LED turns on per sensor. | Test in Progress for SD2 | Implementation in SD2 |
| 6 | Camera Stream Test | Stream from ESP32-CAM should be stable and accessible during sensor activation events. | Camera feed remains stable and clear with active sensors. | PASS | Upon Motion Detection, Image is viewable. |
| 7 | Power Supply Stability Test | System should function correctly for at least 10 minutes without power fluctuation. | All components remain powered and functional for the duration. | PASS | No power fluctuation |

Table 9.2.1.1: Node Hardware Test Plan

### Fail State Checklist:

Power-On Failure:

- ☐ Check Connections → Reconnect components and restart.
- ☐ Check Battery Voltage → Replace if below operational level.

Wi-Fi Connection Failure:

- ☐ Reconnect Wi-Fi → Check SSID and password settings, then reboot.
- ☐ ESP32 Reboot → Reset ESP32 if still unable to connect.

Sensor non-activation:

- ☐ Check Sensor Wiring → Ensure power, ground, and GPIO connections are correct.
- ☐ Replace Sensor → If still unresponsive, replace it with a new sensor.

Buzzer non-activation:

- ☐ Check Buzzer Wiring → Ensure connections to GPIO pin and ground.
- ☐ Test Buzzer with Direct Voltage → Apply voltage directly to confirm functionality.

Camera Stream Unavailable:

- ☐ Check Wi-Fi Signal → Verify the ESP32-CAM is within range.
- ☐ Restart ESP32-CAM → Reboot device if still no stream.

LED Activation Failure:

- ☐ Check GPIO Connections → Ensure LEDs are properly connected to assigned GPIOs.
- ☐ Test LED Functionality → Replace LEDs if still unresponsive.

## 9.2.2 Gadget Test:

The gadget is being tested on general functionality. This includes the functionality of the buttons/LEDs, its ability to maintain the website on the wireless LAN, and its stability over the LAN.

| Test ID | Test Case | Expected Outcome | Pass/Fail Criteria | Pass/Fail | Comments |
|---|---|---|---|---|---|
| 1 | Power-On Test | ESP32-S3 powers up and connects to Wi-Fi. | Web interface is accessible. | PASS | Can access interface. |
| 2 | Switch Functionality Test | Each switch changes state in the website interface when toggled. | Switch status updates correctly in real-time on the website. | PASS | Can monitor changes, cannot make changes yet from Site. |
| 3 | LED Control Test | Each LED can be turned on/off via the website interface. | LEDs respond accurately to website controls. | Not Yet Tested | Plans to test early months of SD2 semester |
| 4 | Website Accessibility Test | Web interface is accessible and displays switch/LED status. | Website loads without errors. | PASS | The Demo Web Interface works, however, plans to test with Mobile App are set for SD2. |
| 5 | Real-Time Updates Test | Switch and LED statuses update in real-time on the website. | Website reflects real-time changes in switch and LED states. | PASS | Demo Web can pick up changes, however strong delays in updates as of now. |
| 6 | Network Stability Test | System maintains connection to the router for 10 minutes. | Website remains accessible without disconnection. | PASS | Does remain connected however need to test long term active use cases. |
| 7 | Power Supply Stability Test | System operates correctly for 10 minutes without power issues. | All components remain functional with stable power. | PASS | No power fluctuation |

Table 9.2.2.2: Gadget Hardware Test Plan

## Fail State Checklist:

Power-On Failure:

- ☐ Check Connections → Reconnect components and restart.
- ☐ Check Power Supply Voltage → Replace or recharge battery if voltage is below operational level.

Wi-Fi Connection Failure:

- ☐ Reconnect Wi-Fi → Verify SSID and password settings, then reboot.
- ☐ ESP32 Reboot → Reset ESP32-S3 if still unable to connect.

Switch Non-Activation:

- ☐ Check Switch Wiring → Ensure each switch is properly connected to its designated GPIO pin.
- ☐ Replace Switch → Replace if still unresponsive after wiring check.

LED Non-Activation:

- ☐ Check LED Wiring → Verify correct connections to GPIO and ground.
- ☐ Replace LED → If unresponsive, test with a new LED.

Website Inaccessible:

- ☐ Check Wi-Fi Signal → Verify the ESP32-S3 is within range.
- ☐ Restart ESP32-S3 → Reboot device if the website is still inaccessible.

No Real-Time Updates:

- ☐ Check JavaScript/Server Code → Ensure that real-time updates are enabled.
- ☐ Restart ESP32-S3 → Reboot if real-time updates are still not functional.

## 9.3 Software Testing:

### 9.3.1 Software Testing introduction:

Software Testing is a critical part of developing any complex system. For our project we will be choosing 3 tests that will ensure the functionality of our security system meets the specific requirements it needs to. In this section we will be going through how we will perform testing on our embedded code, meaning interacting with the hardware and reading from our sensors properly. We will also be testing our communication network and the accuracy of our image processing algorithms.

### 9.3.2 Embedded/Hardware Peripherals test:

Embedded software testing is critical to the success of our project. In this section we will be going through what tests will be performed on the hardware peripherals connected to both the node and the gadget. The software that will be going on our ESP32-CAM and ESP32-S3 is going to be interacting with the hardware peripherals 100% of the time while the nodes are active. Most of the tests described below will be in some shape or form reading values from the GPIO pins and printing to the serial monitor to ensure a successful connection with our hardware peripherals. See below for the variety of tests we will be performing on our system to ensure correct functionality.

#### 9.3.2.1 ESP32-CAM (Node) Embedded Tests:

All tests for the embedded code will be performed to ensure successful system integration can be seen in table 9.3.2.1. All of these tests are very intuitive and can be understood from the table below as they simply test whether a certain hardware peripheral is able interact with our ESP32-CAM dev board properly.

| Test ID | Test Case | Expected Outcome | Pass/Fail Criteria | Pass/Fail | Comments |
|---|---|---|---|---|---|
| 1 | PassiveIR Sensor Software Integration | PassiveIR sensor software integration success message in serail monitor. | Must see success message for reading passive IR sensor data in serial monitor. | PASS | We were able to see motion being detected through print statements in the serial monitor |
| 2 | ActiveIR Sensor Software Integration | ActiveIR sensor software integration success message in serail monitor. | Must see success message for reading Active IR sensor data in serial monitor. | Pass | We were able to see the active IR sensor detect movement through print statements in the serial monitor |
| 3 | Camera software integration | Success message for capturing and storing an image. | Must see success message for image being captured in serial monitor. | PASS | Images were able to be captured and stored successfully |
| 4 | LED Software Integration | Success message for turning lights on and off printed in serial monitor. | Must see success message for LED’s being turned on/off in serial monitor. | PASS | Serial monitor output indicates LEDs are able to be turned on/off. |

Table 9.3.2.1: ESP32-CAM (Node) Embedded Tests

#### 9.3.2.2 ESP32-S3 (Gadget) Embedded Tests:

All tests for the embedded code for the gadget can be seen in Table 9.3.2.2. The three tests for the gadget were chosen as in this section we wished to test if our software and hardware could interact as needed to meet system requirements. All 3 tests below ensure that all embedded code works as needed for our project to function properly.

| Test ID | Test Case | Expected Outcome | Pass/Fail Criteria | Pass/Fail | Comments |
|---|---|---|---|---|---|
| 1 | Triggering alarms switch software Integration | Serial monitor message for successfully triggering the alarm. | Must see message that alarm has been triggered in serial monitor and the actual alarm being turned on. | Not Tested Yet | Plans to test early months of SD2 semester |
| 2 | Turning off/on lights switch software integration | Serial monitor message that switches are being turned on. | Must see Lights turn on and serial monitor message that lights have been turned on. | Not Tested Yet | Plans to test early months of SD2 semester |
| 3 | Turning off alarms switch | Serial monitor message saying alarms being turned off. | Alarm must be turned off and serial monitor message must print that the alarm has been turned off. | Not Tested Yet | Plans to test early months of SD2 semester |

Table 9.3.2.2: ESP32-S3 (Gadget) Embedded Tests

## 9.3.3 Communication Network Tests:

In this section, we will be testing the communication network that our system will live within. The tests below in table 9.3.3 cover the 5 tests needed to ensure that our project's communication network works for all areas required for our system to meet the requirements of the project.

| Test ID | Test Case | Expected Outcome | Pass/Fail Criteria | Pass/Fail | Comments |
|---|---|---|---|---|---|
| 1 | Gadget Connection | Serial monitor gadget connection success message | Must connect to the router successfully | PASS | Gadget can connect to router successfully |
| 2 | Node Connection | Serial monitor node connection success message | Must connect to the router successfully | PASS | Node can connect to the router successfully |
| 3 | Mobile to Gadget Connection messages | HTTP Requests can be sent from the mobile app to the gadget | Gadget must be able to recognize and fulfill requests from the mobile app. | Not Tested Yet | Plans to test early months of SD2 semester |
| 4 | Node to/from Gadget Connection messages | Node can send/receive HTTP messages from/to the gadget | Node must be able to recognize messages from the gadget and send messages to the gadget | PASS | Communica tion is a success between the node and the gadget |

Table 9.3.3: Communication Network Tests

## 9.3.4 Image Processing tests:

Tests for the image processing algorithm will be very simple in the fact that our main metric for a successful person detection algorithm must be able to detect a person. Please note that 100% accurate machine learning models are highly unlikely and almost unheard of. We will be testing our algorithm in different environments and on different people, allowing us to ensure that our software can detect people given reasonable conditions.

### 9.3.4.1 Image Processing Test 1:

The first test we will be performing will be on our model's performance with static environments. The objective will be to evaluate the algorithm's ability to detect a person in a controlled. Static environment without any background movement or noise.

**Test Description:**

In this test, a single individual is positioned in a well-lit room with no background activity. The goal is to measure the accuracy and precision of person detection when the algorithm is presented with clean input data. Success in this case is consistent detection of the person in all frames.

### 9.3.4.2 Image Processing Test 2:

The second test we will be performing will be on our model's performance with dynamic environments. The objective will be to test the algorithm's robustness in environments with significant background activity or moving objects.

**Test Description:**

In this test, a person is introduced into an environment with moderate activity, objects moving, wind, vehicles passing and more. This test evaluates the algorithm's ability to distinguish the target person from irrelevant movements in the scene. Success in this case is correctly detecting the target person while ignoring false positives from background activity.

### 9.3.4.3 Image Processing Test 3:

The third test we will be performing will be on our model's performance with varying light conditions. The objective of this test will be to determine the algorithm's performance under challenging lighting conditions, including low light, shadows, and high contrast.

**Test Description:**

In this test, it is conducted in a dimly lit room and another strong directional light creating shadows. The algorithm is evaluated on its ability to adapt to varying lighting conditions while maintaining detection accuracy. Success in this case will be reliable detection of a person in at least 80% of the frames, even in poorly lit or high contrast environments.

### 9.3.4.4 Image Processing Test 4:

The fourth test we will be performing will be on our model's ability to detect a person given the fact that the whole person may not be shown by the image, being partially obscured. The objective of this test assesses the algorithms' ability to detect a person when parts of their body are partially obscured.

**Test Description:**

In this test, the test subject is partially covered by objects such as furniture, doors, other people, or clothing. The test determines whether the algorithm can identify and recognize the subject despite partial occlusion. Success is defined as maintaining accurate detection in at least 75% of the frames, where significant portions of the person remain visible.

### 9.3.4.5 Image Processing Model Evaluation:

**Accuracy:** Measures how often the detection algorithm correctly identifies a person compared to the total number of predictions made.

$$Accuracy = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Predictions}}\} \qquad (9.3.4.5.1)$$

**Precision:** Quantifies how many of the detected instances classified as a person are actually correct. It focuses on the reliability of positive predictions. High precision value indicates fewer false alarms, making it important for reducing unnecessary alerts.

$$\mathrm{Pr}\,e\,cision = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}\} \qquad (9.3.4.5.2)$$

**Recall:** Measures how many actual people the algorithm successfully detects out of all the people present. High recall value ensures that most of the relevant subjects are identified, even at the risk of detecting some incorrect instances.

$$Recall = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}\} \qquad (9.3.4.5.3)$$

## 9.3.5 Frontend Testing:

### 9.3.5.1 Introduction:

Testing the SecureScape mobile app frontend ensures the user interface performs as intended, focusing on key interactions and transitions between pages. Based on the UI flow depicted in Figure 7.8, tests will validate the functionality of navigation, responsiveness, and backend integration. These tests ensure the app is intuitive, responsive, and accurately reflects the system's real-time state.

### 9.3.5.2 Frontend Tests:

The following table outlines the key tests designed to validate the functionality, usability, and integration of the SecureScape mobile application’s frontend. Each test case is structured to evaluate specific features of the app, such as button functionality, UI rendering, and backend communication, ensuring a user-friendly experience.

| Test ID | Test Case | Expected Outcome | Pass / Fail Result | Comments |
|---|---|---|---|---|
| **1** | Welcome Page Buttons | Buttons provide correct navigation to corresponding pages. | PASS | Navigation through UI is successful. |
| **2** | Connection Page Transition | Device Pairing transitions to Home Page or shows errors | Not Yet Tested | Planned for Early SD2. |
| **3** | Home Page Button | Buttons Trigger Correct Backend Actions | Not Yet Tested | Planned for Early SD2. |
| **4** | About Us Page Rendering | Team Info and Description Displayed Properly. | PASS | Can successfully view About Us Page. |
| **5** | UI Response | UI Has “Active” Feel for User-Friendliness. | PASS | CSS has been implemented to ensure Users know Status of Connection & App. |
| **6** | Real Time Updates | Status Updates Immediately on Home Page. | Not Yet Tested | Has Not Been Tested with Gadget Connection, Can Verify Connection however. |
| **7** | View Photo Functionality | Views Photo from Mobile App. | Not Yet Tested | Can Obtain Images through HTTP with Site, however not implemented with Mobile App. |

Table 9.3.5.2: Frontend Tests

**9.3.5.3 Fail State Checklist:**

This checklist focuses on common frontend failures, including navigation errors, UI response issues, and real-time update failures and/or incorrect error handling, and ensuring a user-friendly experience. By providing clear symptoms and simple troubleshooting steps, this section aims to guide developers in quickly resolving issues while maintaining the application's functionality and design integrity.

1. *Navigation Failures*

- **Symptom** : Buttons do NOT navigate to intended pages.
- **Troubleshooting Steps** :
    - Verify onClick Event Handlers per button.
    - Check routing configurations.

2. *Connection Page Transition Failure*

- **Symptoms** : Device Pairing does not successfully transition to Home Page or display errors.
- **Troubleshooting Steps** :
    - Verify Pairing API request and ensure no error in response.
    - Test error-handling logic to confirm appropriate messages are displayed.

3. *Real Time Update Failures*

- **Symptom** : Status does not update immediately on Home Page.
- **Troubleshooting Steps** :
    - Check polling logic for live updates in backend.
    - Verify state management to confirm real-time updates are caught correctly.
    - Simulate backend events to test real-time updates.

4. *View Photo Error*

- **Symptom** : Photos are not displayed or saved correctly.
- **Troubleshooting Steps** :
    - Verify HTTP Endpoints for retrieving images and check API response structure.
    - Ensure image gallery component properly renders images if API response works.
    - Test file-saving on device for handling downloads.

5. *Home Page Trigger Failure*

- **Symptom** : Buttons on Home Page do not trigger their corresponding backend actions correctly, moreover, the errors associated with trigger failure are not displayed correctly.
- **Troubleshooting Steps** :
    - Confirm API Endpoints are accessible.
    - Ensure connection to Hardware.
    - Verify onClick triggers correct API payloads.

# Chapter 10 - Administrative Content

In this chapter we will go over our budgeting process and the pacing of our project. Our purchases will be included as well for our protype and the final system. Our milestones for Senior Design 1 and 2 will be displayed and updated as we progress our project. Finally, the last item is the work distribution, which shows our team's allocated work shown in this project.

## 10.1 Budget/Cost Estimated:

The budget is self-imposed and was decided by every member's agreement. We will limit ourselves to $1,000 to incentivize our group to choose every part carefully. We want high performance only when needed, and not any costly unnecessary features.

Because the budget of our group is self-imposed, we also understand that our estimations may be off and that some parts may need to push our budgets boundaries. Our PCB budget calculation will include the cost to print the boards, and the parts on the PCB. Our system will have peripherals that cannot be attached to a PCB, thus having their own budget section.

| Section | Budget |
|---|---|
| Gadget Peripherals | $ 100.00 |
| Node Peripherals | $ 750.00 |
| PCBs | $ 150.00 |
| Total | $ 1,000.00 |

Table 10.1: Budget Table

## 10.2 Bill of materials:

| Bill of Materials | | | | |
|---|---|---|---|---|
| **Part** | **Suggested Part** | **Quantity** | **Cost / per** | **Cost Total ($)** |
| ESP32 CAM | DORHEA | 1 | 22.99 | 22.99 |
| ESP32 S3 | DORHEA | 1 | 23.42 | 23.42 |
| Gadget batteries | PAOWANG | 1 | 19.99 | 19.99 |
| Node batteries | POAWANG | 1 | 19.99 | 19.99 |
| PCB Node Build | TBD | ~3 | ~29.99 | 89.97 |
| PCB Gadget Build | TBD | ~1 | ~29.99 | 29.99 |
| Passive IR | STEMEDU | 1 | 9.98 | 9.98 |
| Active IR | GIKFUN | 1 | 5.98 | 5.98 |
| LEDs | CHANZON | 1 | 5.31 | 5.31 |
| Buzzer | WEICHUANG | 3 | 5.49 | 5.49 |
| Housing nodes | TBD | 3 | ~24.99 | 74.97 |
| Housing gadget | TBD | 1 | ~24.99 | 24.99 |
| SSD | SanDisk 256GB | 1 | 25.44 | 25.44 |
| Micro Router | GL.iNET | 1 | 31.90 | 31.90 |
| Gadget Switch | Same Sky | 10 | .53 | 5.30 |
| Wires | Sunxeke | 1 | 9.99 | 9.99 |
| Total Cost | | | | $ 405.70 |

Table 10.2: Bill of materials

For this project, the development cost is steep regarding early prototype design. This analysis focuses on purely the material needed to do a first build. For this cost analysis, we have disregarded the tools and equipment needed for the development and any disposable items. This is to show the price breakdown for each part, and the total for a single build.

## 10.3 Project Milestones for SD1:

| Task | Deadline | Status |
|---|---|---|
| Group Formation | Aug 22nd, 2024 | Completed |
| Idea Brainstorming | Aug 22nd – 24th, 2024 | Completed |
| Idea Selection | Aug 24th, 2024 | Completed |
| Design Software Architecture | September 5th, 2024 | Completed |
| Kickstart Meeting | September 6th, 2024 | Completed |
| Divide and Conquer Document | September 7th, 2024 | Completed |
| Divide and Conquer Meeting | September 10th, 2024 | Completed |
| Begin Software development | September 15th,2024 | Completed |
| Divide and Conquer Document Revision | September 27th, 2024 | Completed |
| Software Hardware Compatibility Test #1 | October 10th, 2024 | Completed |
| Hardware Prototype #1 | October 10th, 2024 | Completed |
| Software Image Processing Algorithm development started | October 10th, 2024 | Completed |
| Software Hardware Compatibility Test #2 | October 24th, 2024 | Completed |
| 60-Page Draft Report | October 25th, 2024 | Completed |
| 60-Page Draft Report Revision | November 8th, 2024 | Completed |
| Final Report | November 26th, 2024 | Completed |
| Mini Demo Video | November 26th, 2024 | Completed |

Table 10.3: Project Milestones for SD1

## 10.4 Project Milestones for SD2:

| Task | Deadline | Status |
|---|---|---|
| Image processing Algorithms complete | February 1st, 2025 | Incomplete |
| All Hardware Parts Ordered | February 1st, 2025 | Incomplete |
| Software Development Complete | February 20th, 2025 | Incomplete |
| Software integration with hardware | March 1st, 2025 | Incomplete |
| Testing and Revision Phase 1 | March 1st, 2025 | Incomplete |
| Testing and Revision Phase 2 | March 15th, 2025 | Incomplete |
| Testing and Revision Phase 3 | April 1st, 2025 | Incomplete |
| System Integration complete | May 15th, 2025 | Incomplete |
| Project completion | May 20th, 2025 | Incomplete |

Table 10.4: Project Milestones for SD2

## 10.5 Work Distribution:

| Chapter | Topic | Pages total | Responsible Eng. | # pages responsible | # pages finished |
|---|---|---|---|---|---|
| 1 | Executive Summary | 1-2 pages | Jaxon | 1 | 1 |
| | | | Dylan | 0 | 0 |
| | | | Phillip | 0 | 0 |
| | | | Kirby | 0 | 0 |
| 2 | Project Description | 10 pages | Jaxon | 4 | 4 |
| | | | Dylan | 4 | 4 |
| | | | Phillip | 2 | 2 |
| | | | Kirby | 3 | 3 |
| 3 | Research and Investigation | 45-60 pages | Dylan | 7 | 7 |
| | | | Jaxon | 9 | 9 |
| | | | Phillip | 18 | 18 |
| | | | Kirby | 12 | 12 |
| 4 | Standards and Design Constraints | 10-15 pages | Jaxon | 2 | 2 |
| | | | Dylan | 5 | 5 |
| | | | Phillip | 2 | 2 |
| | | | Kirby | 2 | 2 |
| 5 | Comparison of ChatGPT with other Similar Platforms | 5-8 pages | Jaxon | 3 | 3 |
| | | | Dylan | 2 | 2 |
| | | | Phillip | 1 | 1 |
| | | | Kirby | 2 | 2 |
| 6 | Hardware Design | 15-20 pages | Dylan | 15 | 15 |
| | | | Jaxon | 0 | 0 |
| | | | Phillip | 2 | 2 |
| | | | Kirby | 0 | 0 |
| 7 | Software Design | 10-20 pages | Dylan | 0 | 0 |
| | | | Jaxon | 8 | 8 |
| | | | Phillip | 0 | 0 |
| | | | Kirby | 6 | 6 |
| 8 | System Fabrication/Prototype Construction | 5-10 pages | Dylan | 3 | 3 |
| | | | Jaxon | 0 | 0 |
| | | | Phillip | 3 | 3 |
| | | | Kirby | 0 | 0 |
| 9 | System Testing and Evaluation | 5-10 pages | Dylan | 5 | 3 |
| | | | Jaxon | 4 | 4 |
| | | | Phillip | 1 | 1 |
| | | | Kirby | 0 | 0 |
| 10 | Administrative Content | 5-10 pages | Dylan | 3 | 3 |
| | | | Jaxon | 2 | 2 |
| | | | Phillip | 2 | 2 |
| | | | Kirby | 0 | 0 |
| 11 | Conclusion | 2-3 pages | Dylan | 0 | 0 |
| | | | Jaxon | 1 | 1 |
| | | | Phillip | 1 | 1 |
| | | | Kirby | 0 | 0 |

Table 10.5: Work Distribution

# Chapter 11 – Conclusion

## 11.1 Hardware Conclusion:

From the conception of our system, we've had our designs' functionality decided upon. Each function had its own complications through part decisions and the connecting circuitry development. The decision-making process was ultimately decided on familiarity for many of the parts we will use in our project. The next biggest concern was to stay within our self-imposed budgeting constraint. After much deliberation, our team had decided to choose a much cheaper option over the statistically better but very expensive option. The choices made based on our budget overall were not very impactful to the performance, however, there would need to be more consideration if this project were to become available on the market. Connecting circuitry was the second complication that was mostly configured through testing and research. Our research showed the basic groundwork for many of our devices through the part's respective datasheets. Our testing resulted in needing to insert passive devices like resistors and capacitors to help stabilize many parts of both gadget and node circuits.

Hardware design plays a crucial role in the overall success of the project. By integrating robust, high-performance components with meticulous attention to detail, the hardware architecture ensures reliability, efficiency, and scalability. Our part selection and design choices create a foundation to support the advanced software capabilities of our project.

## 11.2 Software Conclusion:

The software design of our deployable security system represents a robust balance between performance, scalability, and user needs. By integrating advanced tools such as TensorFlow Lite for on device image processing and leveraging the versatility of the ESP32 microcontrollers, the system achieves real-time detection, efficient data transfer, and secure communication. Our iterative design and development process has emphasized complexity and adaptability. Key features such as enhanced memory allocation, robust network communication protocols, and the leveraging of inherited complex libraries ensure that the system can handle complex tasks without compromising reliability. These considerations not only support the requirements of the project, but also position the system for future upgrades, such as more sophisticated image processing models, more sensor configurations, and extended communication ranges. Through the detailed flowcharts, diagrams, and structured implementations in chapter 7, we have outlined how the systems leverage the hardware capabilities into meaningful functionalities.

In conclusion, the software design underscores the importance of clear, well-documented architecture in achieving the project's goals. By combining thoughtful design principles with cutting-edge technologies, the system delivers an efficient, secure, and scalable solution for modern security challenges.

## 11.3 Final Conclusion:

SecureScape represents the culmination of our efforts to design and develop a robust, user-friendly security system. Combining innovative hardware and software, SecureScape delivers a seamless and intuitive experience for users while maintaining the complexity and functionality required for modern security challenges. The gadget serves as the central hub, equipped with clearly labeled switches and a distinct power switch featuring an LED indicator for clear system status visibility. Additionally, the user-friendly application enhances the setup experience by providing real-time feedback on connection status, ensuring clarity and ease of use.

The security features of SecureScape begin with reliable motion detection, forming the foundation for the system's ability to respond to potential threats. From there, advanced software components, such as on-device image processing with TensorFlow Lite and secure communication protocols, enable real-time alerts and data handling, ensuring that the system performs efficiently and reliably. The hardware integration, including versatile ESP32 microcontrollers, robust sensor configurations, and careful power management, further underscores the system's adaptability and scalability. These features collectively ensure that SecureScape is not only functional but also prepared for future enhancements.

While this document marks the completion of our initial research and design, we recognize that refinement and iteration are an ongoing process. Changes and upgrades are inevitable, and we remain committed to documenting and communicating these developments to maintain the integrity and direction of our project.

In summary, SecureScape stands as a testament to the importance of integrating thoughtful hardware design, advanced software development, and a user-focused approach. The result is a comprehensive and dependable security system that exemplifies both innovation and practicality, delivering a powerful solution for modern safety needs.

# Citations, Research, and Resources:

## Appendix A – References


[1] Wikipedia contributors. (2024, August 22). Digital image processing. In *Wikipedia, The Free Encyclopedia*. Retrieved 19:18, August 30, 2024, from https://en.wikipedia.org/w/index.php?title=Digital_image_processing&oldid=1241614779

[2] Wikipedia contributors. (2024, August 13). Signal processing. In *Wikipedia, The Free Encyclopedia*. Retrieved 19:19, August 30, 2024, from https://en.wikipedia.org/w/index.php?title=Signal_processing&oldid=1240021905

[3] Hulppre Solar Outdoor Motion Sensor Alarm-Dog Barking&Gunshot sound, support recording,130db loud noise maker,13 kinds of sound strobe light with USB,say goodbye to the intruders - amazon.com. (n.d.). https://www.amazon.com/HULPPRE-Outdoor-Alarm-Dog-Recording-Intruders/dp/B0C5HMS1PN

[4] *Understanding conservation*. National Wildlife Federation. (n.d.). https://www.nwf.org/Educational-Resources/Wildlife-Guide/Understanding-Conservation

[5] Wikipedia contributors. (2024, October 17). C++. In *Wikipedia, The Free Encyclopedia*. Retrieved 14:22, October 17, 2024, from https://en.wikipedia.org/w/index.php?title=C%2B%2B&oldid=1251670103

[6] Wikipedia contributors. (2024, October 16). Python (programming language). In *Wikipedia, The Free Encyclopedia*. Retrieved 14:26, October 17, 2024, from https://en.wikipedia.org/w/index.php?title=Python_(programming_language)&oldid=1251534510

[7] Wikipedia contributors. (2024, October 11). Arduino. In *Wikipedia, The Free Encyclopedia*. Retrieved 14:28, October 17, 2024, from https://en.wikipedia.org/w/index.php?title=Arduino&oldid=1250690945

[8] Wikipedia contributors. (2024, October 16). JavaScript. In *Wikipedia, The Free Encyclopedia*. Retrieved 21:01, October 18, 2024, from https://en.wikipedia.org/w/index.php?title=JavaScript&oldid=1251534737

[9] Wikipedia contributors. (2024, February 19). OpenCV. In *Wikipedia, The Free Encyclopedia*. Retrieved 20:57, October 18, 2024, from https://en.wikipedia.org/w/index.php?title=OpenCV&oldid=1208982530

[10] Wikipedia contributors. (2024, October 15). TensorFlow. In *Wikipedia, The Free Encyclopedia*. Retrieved 21:11, October 18, 2024, from https://en.wikipedia.org/w/index.php?title=TensorFlow&oldid=1251402385

[11] Wikipedia contributors. (2024, October 11). HTTP. In *Wikipedia, The Free Encyclopedia*. Retrieved 21:26, October 18, 2024, from https://en.wikipedia.org/w/index.php?title=HTTP&oldid=1250568137

[12] Wikipedia contributors. (2024, October 10). WebSocket. In *Wikipedia, The Free Encyclopedia*. Retrieved 21:32, October 18, 2024, from https://en.wikipedia.org/w/index.php?title=WebSocket&oldid=1250439164\

[13] Wikipedia contributors. (2024, September 29). Flutter (software). In Wikipedia, The Free Encyclopedia. Retrieved 22:30, October 21, 2024, from https://en.wikipedia.org/w/index.php?title=Flutter_(software)&oldid=1248514788 4

[14] *What is react native? Beginner's Guide + FAQ*. Coursera. (n.d.). https://www.coursera.org/articles/what-is-react-native?msockid=1a27620d21d963e41447771e204962fd

[15] Wikipedia contributors. (2024, July 9). Ionic (mobile app framework). In Wikipedia, The Free Encyclopedia. Retrieved 22:35, October 21, 2024, from https://en.wikipedia.org/w/index.php?title=Ionic_(mobile_app_framework)&oldid=1233499145

[16] Wikipedia contributors. (2024, September 27). Xamarin. In Wikipedia, The Free Encyclopedia. Retrieved 22:35, October 21, 2024, from https://en.wikipedia.org/w/index.php?title=Xamarin&oldid=1248078093

[17] GeeksforGeeks. (2024, July 3). *Aws Lambda*. https://www.geeksforgeeks.org/introduction-to-aws-lambda/#

[18] GeeksforGeeks. (2021, July 15). *Firebase - Introduction*. https://www.geeksforgeeks.org/firebase-introduction/

[19] GeeksforGeeks. (2022, July 15). *Getting started on Heroku with python*. https://www.geeksforgeeks.org/getting-started-on-heroku-with-python/

[20] Wikipedia contributors. (2024, July 26). DigitalOcean. In Wikipedia, The Free Encyclopedia. Retrieved 22:41, October 21, 2024, from https://en.wikipedia.org/w/index.php?title=DigitalOcean&oldid=1236707123

[21] Google. (n.d.). *Firestore documentation | google cloud*. Google. https://cloud.google.com/firestore/docs/

[22] Wikipedia contributors. (2024, August 5). MongoDB. In Wikipedia, The Free Encyclopedia. Retrieved 22:43, October 21, 2024, from https://en.wikipedia.org/w/index.php?title=MongoDB&oldid=1238821960

[23] *W3schools.com*. W3Schools Online Web Tutorials. (n.d.). https://www.w3schools.com/postgresql/index.php

[24] Supabase. (2024, October 21). *Database*. Supabase Docs. https://supabase.com/docs/guides/database/overview

[25] *Why you should not charge a lithium battery below 32 degrees*. REDARC. (2022, September 19). https://www.redarcelectronics.com/us/resources/chargers-isolators-faqs/do-not-charge-lithium-battery-below-32-degrees/

[26] *Taper*. Taper | Analog Devices. (n.d.). https://www.analog.com/en/resources/glossary/logarithmic_linear_taper.html#:~:text=When%20used%20in%20an%20amplifier,used%20for%20audio%20volume%20controls.

[27] *A comprehensive guide to PMIC*. FPT Semiconductor .,JSC. (2024, February 22). https://fpt-semiconductor.com/blogs/a-comprehensive-guide-to-power-management-integrated-circuit-pmic/

[28] *Understanding conservation*. National Wildlife Federation. (n.d.). https://www.nwf.org/Educational-Resources/Wildlife-Guide/Understanding-Conservation

[29] *Battery voltage*. PVEducation. (n.d.). https://www.pveducation.org/pvcdrom/battery-characteristics/battery-voltage

[30] *Different types of batteries for electronic products (Importer's Guide)*. Sofeast. (2021, January 6). https://www.sofeast.com/resources/different-types-of-batteries-for-electronic-products-guide/

[31] Battery University. (2021, December 3). *BU-107: Comparison table of secondary batteries*. https://batteryuniversity.com/article/bu-107-comparison-table-of-secondary-batteries

[32] How many months of a lithium polymer battery life. (n.d.). https://www.lipolbattery.com/lithium-polymer-battery-life.html

[33] *The fundamentals of LDO Design and Applications*. The Fundamentals of LDO Design and Applications | Analog Devices. (n.d.). https://www.analog.com/en/lp/001/fundamentals-of-ldo-design-and-applications.html#:~:text=A%20low%20dropout%20regulator%20(LDO,is%20recommended%20to%20ensure%20stability.

[34] *When should you choose LDO or Buck Converter?*. CircuitBread. (n.d.). https://www.circuitbread.com/ee-faq/when-should-you-choose-ldo-or-buck-converter#:~:text=A%20buck%20regulator%20is%20preferred,converter%20is%20the%20ideal%20choice.

[35] *The benefits of a compact power management IC and Power Loss Protection*. element14 Community. (n.d.). https://community.element14.com/technologies/power-management/b/blog/posts/the-benefits-of-a-compact-power-management-ic-and-power-loss-protection

[36] Nqtronix, & Instructables. (2017, September 24). *Practical guide to leds 1 - pick your led!*. Instructables. https://www.instructables.com/Practical-Guide-to-LEDs-1-Pick-Your-LED/

[37] Ou, M. (2024, June 18). *The ultimate guide to cob LED STRIP*. MSHLED. https://www.stripsledlight.com/the-ultimate-guide-to-cob-led-strip/

[38] Ibe. (2024, September 14). *Guide to light-emitting diode (LED) : Types, application and manufacturers*. IBE Electronics. https://www.pcbaaa.com/light-emitting-diodes-led/

[39] Justin. (2024, June 12). *SMD vs. COB LED STRIP: A comprehensive comparison 2024 - led controller manufacturer, supplier, factory: Led Smart Light Supplier: Led Strip Lights Supplier*. LED Controller Manufacturer, Supplier, Factory| LED Smart Light Supplier | LED Strip Lights Supplier. https://onesmartlighting.com/blog/smd-vs-cob-led-strip/

[40] Ou, M. (2023, March 13). *IP rating: The definitive guide -MSHLED lighting*. MSHLED. https://www.stripsledlight.com/ip-rating-the-definitive-guide/

[41] Patel, S. (2023, November 20). *What is through Hole Technology*. Candor Industries. https://www.candorind.com/blog/what-is-through-hole-technology/

[42] *Piezo buzzers vs. Magnetic Buzzers: APC International*. americanpiezo. (2023, November 27). https://www.americanpiezo.com/blog/piezo-buzzers-vs-magnetic-buzzers/

[43] *Piezo vs. magnetic buzzers: Which to choose?:news*. FHD Electronics Corporation. (n.d.). https://fhdmfg.com/news/news_detail/magnetic-buzzer

[44] ChatGPT assisted text and formating; https://chatgpt.com/

[45] *How do speakers work: A guide on how speakers produce sound*. soundcore. (n.d.). https://www.soundcore.com/blogs/speaker/how-do-speakers-work-speakers-and-sound-production-explained

[46] Wikipedia contributors. (2024, September 16). Redux (JavaScript library). In *Wikipedia, The Free Encyclopedia*. Retrieved 05:00, November 21, 2024, from https://en.wikipedia.org/w/index.php?title=Redux_(JavaScript_library)&oldid=1245939574

[47] Mobxjs. (n.d.). Mobxjs/mobx: Simple, Scalable State Management. GitHub. https://github.com/mobxjs/mobx

[48] Provider: Flutter Package. Dart packages. (2024, February 28). https://pub.dev/packages/provider

[49] What is Vuex?. Vuex. (n.d.). https://vuex.vuejs.org/

[50] easy eda, jonmill https://oshwlab.com/jonmill/esp32-s3-base

[51] easy eda, GarryChen, https://oshwlab.com/gerrychen/LAN8720_Ethernet-Fe8qk0icu

[52] Espressif, PCB Design, https://docs.espressif.com/projects/esp-hardware-design-guidelines/en/latest/esp32s3/pcb-layout-design.html

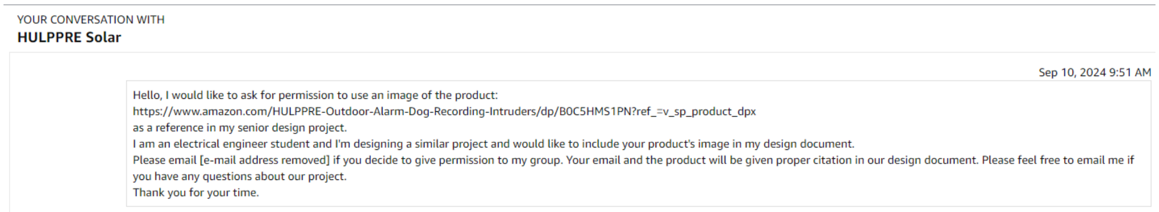
[53] Smart home security with Remote 911. Ooma.com - Smart solutions for home and business. (2024, November 15). https://www.ooma.com/home-security/diy-security-system/

[54] Macdonald, J. (n.d.). From portable door locks to personal alarms, these are the 15 Best Travel Safety Devices T+L editors use. Travel + Leisure. https://www.travelandleisure.com/best-travel-safety-products-6833148

## Appendix B – Copyright permission

- ChatGPT
  - ChatGPT was utilized throughout the document to review and refine written paragraphs, enhancing their clarity and flow.
- Lucid Flowchart
  - Lucid Flowchart was used to make the software diagram.
  - As stated in Permission for academic purposes . Since I have created the diagram I am free to use it however I like, mentioning the software used to make it is completely optional.
- Draw.io
  - Draw.io was used to make the hardware diagram
- HULPPRE
  - Used an illustration to provide a reference of our competitors

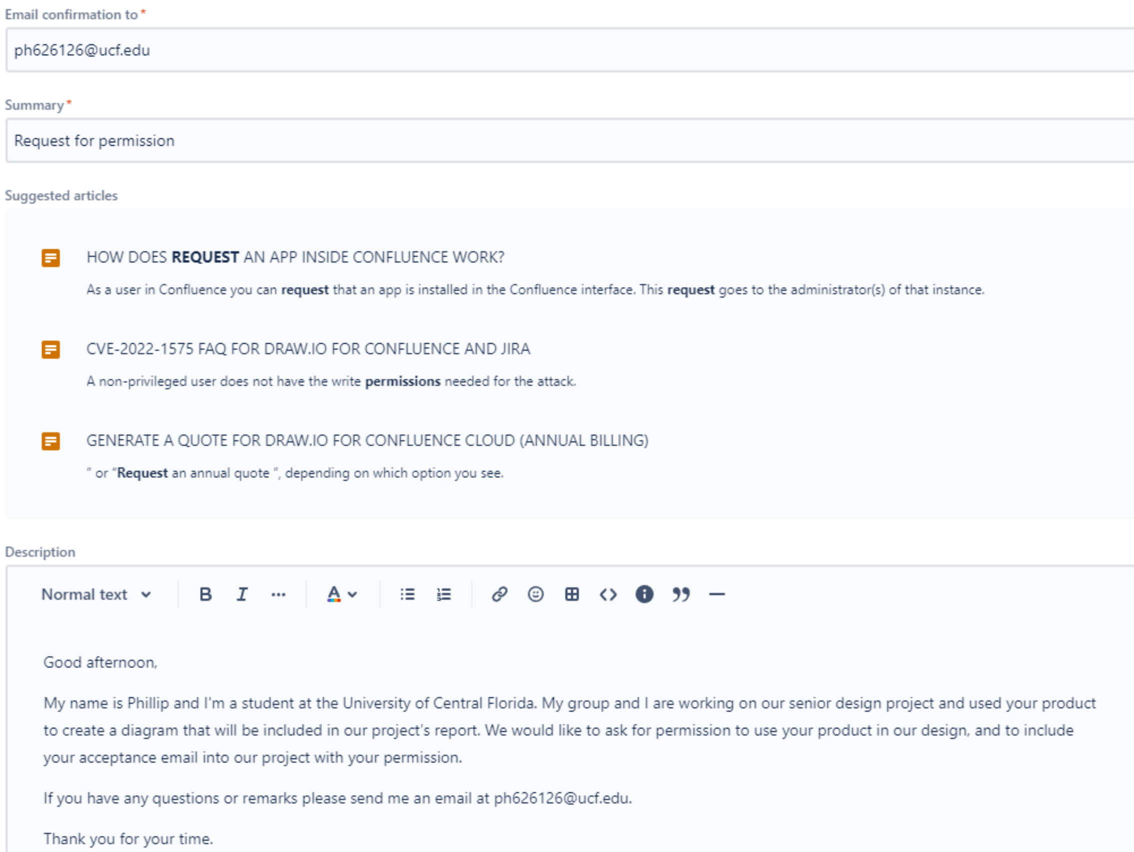
Figure 2.4 Requested:


YOUR CONVERSATION WITH
**HULPPRE Solar**

Sep 10, 2024 9:51 AM

Hello, I would like to ask for permission to use an image of the product:
https://www.amazon.com/HULPPRE-Outdoor-Alarm-Dog-Recording-Intruders/dp/B0C5HMS1PN?ref_=v_sp_product_dpx
as a reference in my senior design project.
I am an electrical engineer student and I'm designing a similar project and would like to include your product's image in my design document.
Please email [e-mail address removed] if you decide to give permission to my group. Your email and the product will be given proper citation in our design document. Please feel free to email me if you have any questions about our project.
Thank you for your time.

HW Diagram Requested:



Email confirmation to *

ph626126@ucf.edu

Summary *

Request for permission

Suggested articles

HOW DOES **REQUEST** AN APP INSIDE CONFLUENCE WORK?

As a user in Confluence you can **request** that an app is installed in the Confluence interface. This **request** goes to the administrator(s) of that instance.

CVE-2022-1575 FAQ FOR DRAW.IO FOR CONFLUENCE AND JIRA

A non-privileged user does not have the write **permissions** needed for the attack.

GENERATE A QUOTE FOR DRAW.IO FOR CONFLUENCE CLOUD (ANNUAL BILLING)

" or "**Request** an annual quote ", depending on which option you see.

Description

Normal text

Good afternoon,

My name is Phillip and I'm a student at the University of Central Florida. My group and I are working on our senior design project and used your product to create a diagram that will be included in our project's report. We would like to ask for permission to use your product in our design, and to include your acceptance email into our project with your permission.

If you have any questions or remarks please send me an email at ph626126@ucf.edu.

Thank you for your time.

HW Diagram Accepted:

N/A