Project Title
Interim Report

TU858
BSc in Computer Science (International)

Matiss Jurevics
C22501743

Supervisor
Fadoa Rafii

School of Computer Science
Technological University, Dublin

Date 23/11/2025

Abstract

Commercial smart home security cameras deliver complex AI powered features such as natural language event description and package recognition. The issue with these solutions is that many of the companies have a track record of privacy violations and data leaks (some of which this report will highlight later) With that in consideration, a privacy conscious security camera with advanced AI features built from affordable, easy to obtain and assemble components would interest many.

In this project I examine how a mixture of open source, self hostable software, highly optimised low latency computer vision models and quantised large language models can be used in tandem to create the same experience that commercial smart home cameras provide.

Declaration

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed:

*Matiss Jurevics*_____

Matiss Jurevics

Date 23/11/2025

Acknowledgements

I want to thank my mentor Fadoa Rafii for her insight that helped guide my project throughout the course of this year as well as my friends and family for their support.

Contents

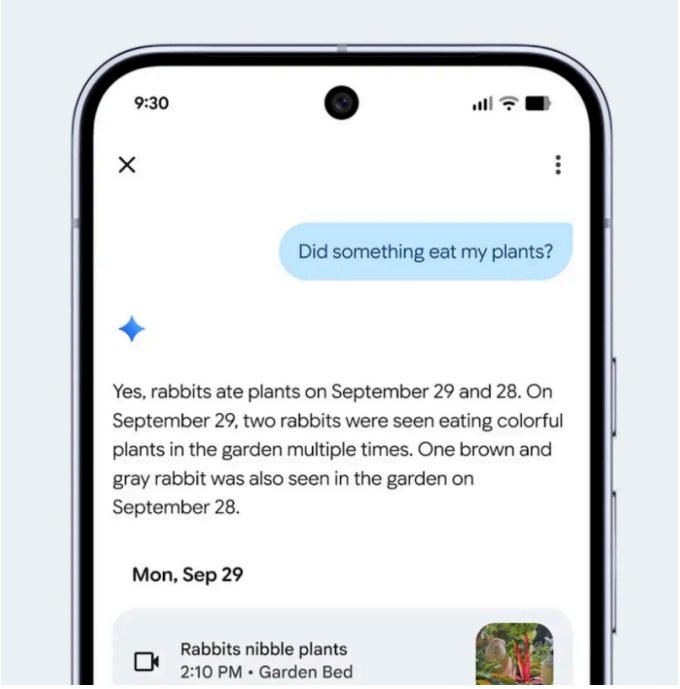
1. Introduction

1.1 Project Background

Over the past 20 years, there has been an explosion in the home security space as IoT devices provide people with devices more capable than devices previously left exclusive to large enterprises and now with new multimodal AI models, we are now able to pass security footage into a model and provide users with vivid descriptions of what is going on both inside and outside your home.

In 2011, Jamie Siminoff developed the first prototype of what would later be known as the ring doorbell. This device allowed for people to spend under €200 and have a device capable of showing them whats going on just outside their front door from anywhere in the world.

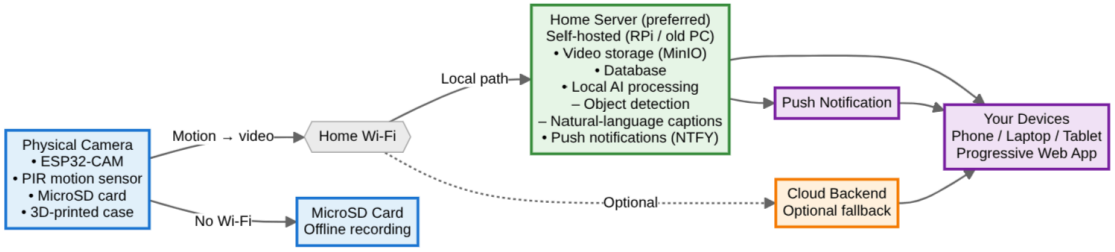
Newer software backends now allow these cameras to send notifications to your phone when it detects someone suspicious on your property or when a noteworthy event happens (postman is waiting with a package, children damaging plants in your garden, etc).



While these new AI powered cameras are incredibly useful, they do pose some privacy concerns typical of products sold by tech giants.

1.2 Project Description

The project will consist of a camera, local server (which is optional), cloud server (the default) a notification service, a MiniIO storage client for blob storage, supabase client for authentication and a python web server for all the image processing and then a frontend progressive web application for providing the user with a ui.



1.3 Project Aims and Objectives

This project aims to create a fully open source, privacy focused and easy to construct security camera using easily accessible parts.

Objectives

- Find the features that users value most from the currently available home security solutions.
- Develop an image processing pipeline that provides actionable data that the software can use.
- Design the physical camera in such a way that makes it as easy as possible to construct with minimal knowledge of electronics.
- Develop an easy to navigate user interface that lets the user interact with the data obtained from the camera.
- Develop self-hostable infrastructure that the user themselves could set up if willing while providing cloud based solutions for users who are not proficient with technology.
- Minimize reliance on external cloud based solutions in order to minimize privacy concerns.

1.4 Project Scope

In scope:

- A single camera prototype with built in motion detector and flash for dark environments.

- Web server for retrieving image/video from the camera after motion detector is triggered.
- Image processing pipeline that detects if the event triggering the motion sensor contains people who are registered in the software and that detects what they are doing.
- The Web application through which the user can check the camera and see previous events.

Out of Scope:

- Multi camera management system.
- Integration with Ai assistants like siri, google assistant.
- Mobile app.

1.5 Thesis Roadmap

Chapter 2 reviews existing commercial and open-source solutions and surveys relevant edge-AI technologies.

Chapter 3 analyses system requirements and presents the initial specification.

Chapter 4 details the software methodology, architecture, and detailed system design.

Chapter 5 describes the testing strategy and evaluation methodology.

Chapter 6 documents the prototype implementation, presents results, and evaluates performance.

Chapter 7 discusses issues encountered, reflects on the project outcomes, and proposes future work.

2. Literature Review

2.1 Introduction

This chapter examines the current home security landscape and examines existing open and closed source solution. I'm going to focus on the way information is processed and historic privacy issues these products have had. Im also going to look at the specific features they have and find what would be considered the bare minimum to enter the market.

2.2 Alternative Existing Solutions

Ring Doorbell:

Launched in 2013, the ring doorbell popularized the video doorbell. Current iterations of the camera record at 1536p, have person/package detection and deliver natural language notifications such as "a package has been delivered". The advanced AI features are locked behind a subscription and are processed on Amazons cloud. The data being held with Amazon has raised privacy concerns among certain groups.

Google Nest Cam:

Googles Nest cameras provide 1080p-2K HDR video and have on device basis detection (detecting motion, people and vehicles). They use Google's Gemini AI for face recognition and activity zones. Users can receive periodic event histories with larger storage and advance alerts with Google's monthly premium subscription.

Eufy SoloCam S340

Eufy markets itself as a privacy-focused alternative. It performs person, vehicle, pet and face recognition fully locally resulting in it being a case study for privacy oriented cameras. It doesnt require subscription for cloud based processing and data is stored on an sd card. The one downside is the fact that notifications are simpler than the previous two only providing messages like "person" or "car".

Solution	**Local Processing**	**Natural Language Alerts**	**Subscription**	**Setup Difficulty**	**Open source**	
Ring Doorbell	no	yes	required	Very easy	no	
Google Nest Cam	partial	yes	required	Very easy	no	

Eufy Solocam s340	yes	no	none	easy	no	

2.3 Technologies Researched

Svelte

Svelte is a moder front end framework that precompiles files in order to prevent having additional overhead in the form of a virtual dom. While not as popular as similar frameworks, it is known for providing a better developer experience.

Typescript

Typescript is a superset of javascript with static types. It catches bugs at compile time instead of when they occur in production. It also provides interfaces, enums, generic types and compiles to clean javascript upon build.

Bun JS

Bun JS is an ultra fast alternative to javascript written in Zig and Javascriptcore. It replaces the nodejs runtime, npm package manager, the esbuild bundler and jest testing library all at once. It has a startup time 10-30 times faster than NodeJS.

OpenCV

OpenCV is an open source computer vision library with 30+ years of features and over 2500 algorithms.

Supabase Auth

Supabase is an open source backend as a service platform. It provides people with a way to quickly iterate and develop their applications. For this project I will only use it for authentication through which they cover authentication with most major OAuth clients such as google, apple, facebook and many, many more.

MiniIO

MiniIO is a high performance, S3 compatible distributed object storage server written in go. S3 storage applications are used for blob storage, a blob (binary large object) is essentially a file whos contents dont matter. Blob storage allows for people to easily upload and retrieve files (such as videos or images from security cameras). MiniIO is self hostable which means that if configured properly, a user could run this on their own local device at home.

Platform IO

PlatformIo is a cross platform build system, library manager and debugger for thousands of embedded boards. It integrates into IDE's like VSCode and supports OTA updates and filesystem uploads. OTA or Over-The-Air updates means that you can update the firmware on an embedded device without it being physically connected to your device.

ESP 32 CAM

An ESP32 Cam is an Esspresif 32 based System-on-chip with an integrated 2 megapixel camera. These are great for a budget project as they have an incredibly low per unit cost and the devboards can be used to connect to external modules like a motion sensor without the need to solder components together. The main drawbacks of these devices is their slow image processing times and the low camera quality.

Express JS

Express is a minimal web framework for NodeJS or Bun that provides a thin layer of basic web-application features without obscuring the underlying HTTP server capabilities. Express will serve the purpose as a proxy for the python opencv web server while also storing data in MiniIO.

ShadCN

ShadCN is a tailwind based component library which provides customisable and robust UI elements. These components are built on radix UI primitives and TailwindCSS.

TailwindCSS

TailwindCSS is a Utility first CSS framework. This means that instead of writing custom CSS classes, you compose pre defined atomic classes in your HTML. An atomic class is essentially a class that contains a single attribute such as border-radius: 1em;.

Linux

The server running all of the backend infrastructure will run Linux (specifically debian) This will provide a lightweight system on which i can run all of my applications.

Progressive Web Application

A Progressive Web App (PWA) is a web application that uses modern web technologies (such as Service Workers, Web App Manifest, HTTPS) to deliver a native-app-like experience directly in the browser. It loads instantly even on slow networks, works offline, can be installed on a phone or desktop home screen with an icon, sends push notifications, and feels smooth and responsive — all while remaining a normal website that anyone can visit with a single URL.

2.4 Other Research

There are a lot of papers and new articles that mention privacy concerns related to the ring doorbell and nest camera and how their data is handed to the police for investigation. There is also research that shows how vision models and large language models can be run on laptops using methods such as quantisation. Quantisation is when you convert high precision 32 bit floats into a much smaller size so that they can fit in a computers RAM.

2.5 Existing Final Year Projects

A few years ago, a student name Michal Chojnacki in TU856 created a project called "Remote home Security", it claimed to develop a comprehensive modular framework for developing home security systems. This project used a raspberry pi to stream data from and had an android app that the user could navigate through and see the video stream.

2.6 Conclusions

After going through this in detail, I can see that there is in fact an interest in devices that people have full ownership of and there are genuine privacy concerns stemming from widely soled home security solutions like the google nest and ring doorbell.

3. System Analysis

3.1 System Overview

From the users perspective, there is a plastic case with the camera and sensor inside, they plug this camera into their laptop and through some user interface they configure if they want to go with recommended settings or customise it themselves. This allows them to configure the device to run at home on their local devices

3.2 Requirements Gathering

Key stakeholders

1. End user (non-technical homeowner or renter – the "grandma test")
2. Installer (often the same person – must be beginner-friendly)
3. Developer/maintainer (me, and potentially future open-source contributors)
4. Family members who receive notifications

Collecting Requirements

- Informal interviews with friends, family and strangers.
- Search the internet for reviews and complaints from users online.
- Examine other open source projects.

Requirements Collected

- People do not want to pay a monthly subscription for the duration of their cameras lifetime.
- People (especially tech savvy people) want to own their data.
- Some people often feel nervous when they receive notifications from their cameras when they are not at home as they are not informed as to whats going on until they see the footage.
- People want it to work when the internet is down.

3.3 Requirements Analysis

From my analysis I have found that there are 4 main things that need to be the focus of the project.

Emphasis on Privacy

Users have repeatedly described how they are uncomfortable with the privacy issues presented with commercial solutions that upload footage to third party servers. Some people pointed out some high profile incidents where user data was not properly handled and made publicly available, violating peoples privacy.

As a result some of the more tech savvy of the users we were interviewing showed interest in the option of being able to self host all of their infrastructure so that they can be responsible for and own their data.

Presenting the users with an option to use a quantized multi modal model will allow for them to have the natural language processing be done on their own device/network. Reducing the privacy concerns resulting from sending data to ai providers like openai.

Natural Language Notifications

A lot of people have mentioned when presented with the option that they would like to be presented with natural language push notifications when their cameras report movement. People described that not having to open the app to understand whats going on outside or inside their home would help prevent the from worrying too much about the fact that their cameras went off.

Extreme Ease of Use

Many interviewees stated that while they value privacy, they will choose the easier to use and more user friendly product in spite of privacy concerns if the alternative is difficult to setup and use.

This highlights that I need to place a serious focus on developing not just a great user experience but also making the process of installation as easy and simple as possible. Leaning into the DIY aspect may also convert assembly and installation into a fun family activity.

Zero ongoing cost

One thing that people said is absolutely unacceptable is how even after paying full price for the hardware, they still have to pay a monthly fee in order to actually take advantage of the features of the camera. The implication that you do not own the thing you bought and rely on the companies still running the services for the hardware

3.4 Functional Requirements

This section will contain all of the functional requirements that this project will have:

- The system shall capture video and/or still images from a single ESP32-CAM module when triggered, providing the primary source of visual data for all subsequent processing.

- The system shall store all captured video footage and images in user-designated blob storage (MinIO), ensuring persistent and retrievable access to every recorded event.

- The system shall perform hardware-level motion detection using the attached PIR sensor to determine when the ESP32-CAM should initiate photo capture or video recording, thereby conserving bandwidth and storage when no activity is present.

- The system shall employ a quantised multimodal vision-language model (currently Phi-3.5-Vision 4-bit or equivalent) to analyse selected frames after initial object detection and generate detailed, human-readable descriptions of the scene.

- The system shall generate and deliver natural-language notifications (via NTFY or an equivalent push mechanism) to the user's devices, containing the caption produced by the vision-language model along with relevant metadata (timestamp, camera identifier, confidence indicators).

- The system shall provide a responsive, web-based user interface accessible from any modern browser, enabling users to log in securely and interact with the system.

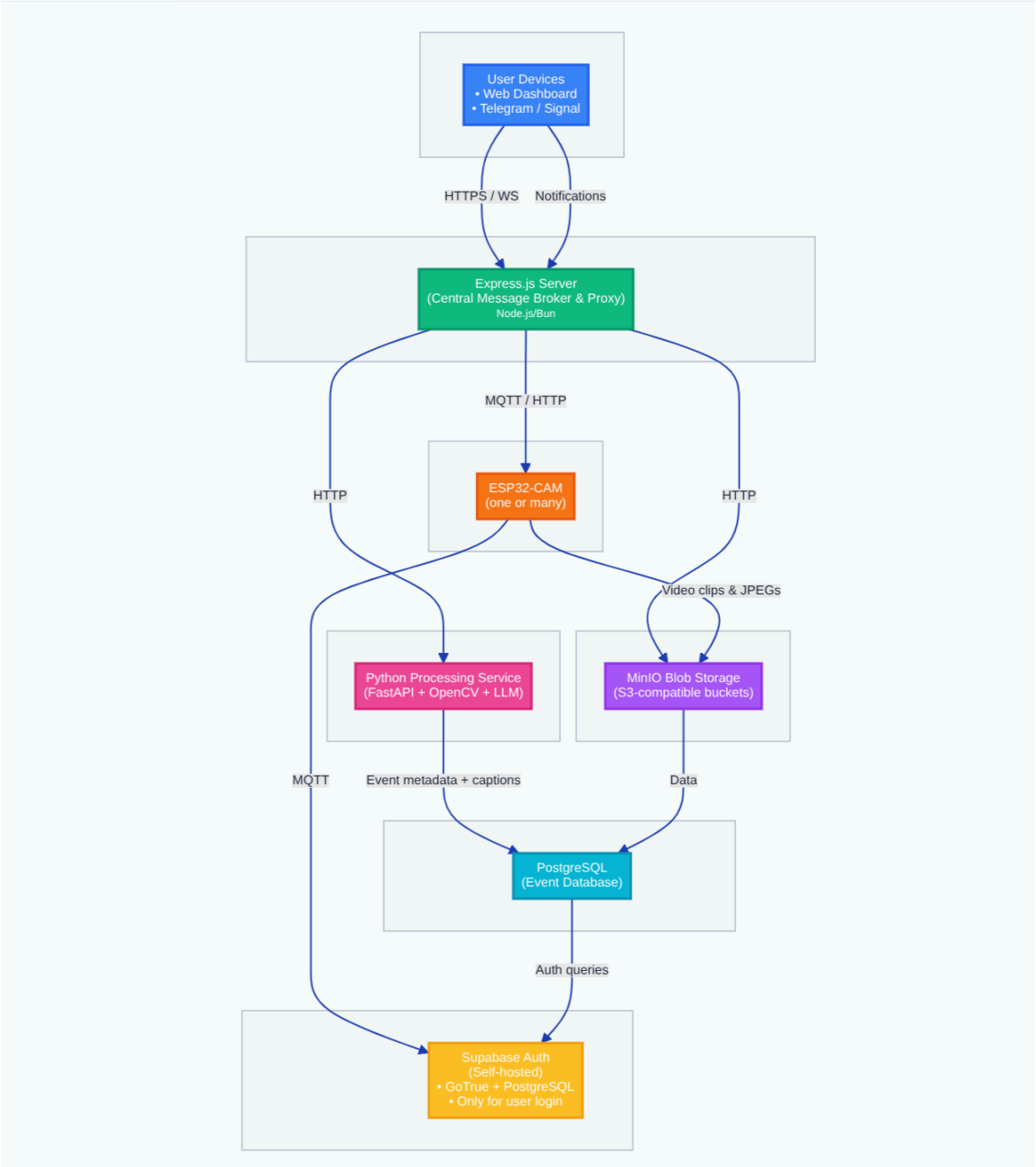
- The system shall visualise recorded events within the web interface in a chronological grid or timeline view, displaying thumbnails, timestamps, generated captions, and any detected objects or recognised faces.

- The system shall allow users to manually label detected faces within the web interface, associate them with names or relationships, and configure personalised notification preferences.

3.5 Proposed System Architecture

My proposed architecture consists of:

- The camera hardware itself, there is a built in motion sensor that triggers when movement is detected. Upon activation, it starts streaming to an express server which acts as a proxy.
- The express server will store the video in memory while also transferring it to blob storage, when the video is finished recording it will be sent to a python web server and a database record will be created.
- The python web server will put the media through various computer vision models and all of the output data will be written into the database
- Once all of the data is processed, based off of the information from the database, the express server will decide whether or not to send the user a push notification.
- The frontend (a svelte application) will contain a user interface, this webapp will initially request the user to enable notifications and once the server detects a noteworthy event, it will trigger a webhook to notify the user



3.6 Non functional requirements

Total hardware costs should be less than 50 euro.

The complete single-camera system – ESP32-CAM board, power supply, case, SD card and basic cabling – must cost under €50 in total, making it affordable for anyone and proving that advanced AI security does not require expensive commercial hardware.

Initial assembly and installation time for a non-technical user

A completely non-technical person must be able to unbox the parts, plug everything together, flash the SD card with a single drag-and-drop file, and have the camera fully working and sending its first notification in under 30 minutes.

The total inference time (from motion to notification) should be less than 10 seconds.

From the moment motion is detected by the ESP32-CAM until the user receives a detailed natural-language notification on their phone, the entire pipeline upload, storage, AI captioning and push – must complete in under 10 seconds.

The system must record video with no wifi connection.

Even if Wi-Fi or the main server is completely offline, the ESP32-CAM must still detect motion and save the full video clip to its own micro-SD card so that no event is ever lost.

3.7 Conclusions

The requirements gathering and analysis phase confirmed that the biggest unmet needs in home security are privacy, cost, and genuinely useful alerts. By prioritising full local processing, beginner-friendly installation, and state-of-the-art multimodal AI running on consumer hardware, the proposed system directly addresses the gaps identified in Chapter 2. The functional and non-functional requirements, together with the chosen three-tier edge architecture, now provide a clear and achievable specification for the design and implementation phases.

4. System Design

4.1 Introduction

This chapter turns the requirements from chapter 3 into a concrete, buildable design. The approach was deliberately pragmatic: start with the simplest thing that could possibly work, test it in the real house as quickly as possible, and then iteratively improve performance, reliability, and user experience.

4.2 Software Methodology

Given that this was a solo final-year project with a single stakeholder group (the developer's own household), a strict Agile-inspired iterative approach was adopted. Development followed very short cycles – typically one to three days – consisting of:

1. implementing the smallest new feature that could provide value,
2. immediately deploying it to the real camera(s) mounted in the home,
3. living with the change for 24–72 hours and observing actual usage by non-technical family members
4. refining or rolling back based on direct feedback and personal pain points.

Formal ceremonies were omitted, but daily stand-ups were effectively replaced by real-world testing in the target environment. This lean methodology allowed rapid validation of the core requirements, especially ease-of-use and notification quality. This was far more effective than traditional waterfall or heavily documented processes would have done for a one-person project.

4.3 Overview of System

Physical System

The physical system is made up of just 4 main elements. All of which can be placed inside a users home in order to allow the user to own their data. They are:

The ESP32 CAM units

These will be mounted in a dry space with access to USB power (be it from a wall socket or portable power bank). These will be made up of:

- An ESP32 CAM module

- A 3d printed plastic enclosure.
- An infrared motion sensor.
- An antenna for better reception. (build in with the module)
- An sd card fitted into the sd slot for offline video storage in case of power/ wifi outage.

Home Server (optional for users who don't want to rely on the cloud version that acts as the default)

This is where the user will be able to run a docker compose to quickly set up all of the backend infrastructure and route all of the cameras and their phones traffic to. The server will run:

- A supabase client specifically configured for handling user authentication, including sign-up flows, login processes, password resets, and session management across the application.
- A MiniIO client for blob storage that manages the uploading, retrieval, and deletion of images and videos.
- A python web server running OpenCV for facial recognition and event detection, analyzing captured images and videos to identify faces, detect motion events, and classify activities taking place in the camera's field of view.
- An express server with endpoints both for data storage operations (creating, reading, updating, and deleting records) and also endpoints that have it act as a proxy for the python server, routing requests and handling responses between the frontend and the image processing service.
- A postgres database to store records of events, user data, metadata about processed images, timestamps, and relational data that ties together all the different components of the system.
- A HTTP server that serves the frontend to the client allowing them to view all of the events on their camera.
- An NTFY or adjacent server allowing easy implementation of push notifications.

The end goal is to allow a user to have this run on a cheap device such as a raspberry pi and allow them to forget about it to meet the requirement of 0 ongoing costs.

Local Network.

This is simple enough, three needs to be a network the user can connect to and route all of the data through for their device to work. This can be a standard home wifi network that the ESP32 can connect to using its built in wifi chip.

User Devices

This is simply a tablet, laptop or any device capable of running a web browser through which the user can access the frontend application and view and manage their cameras data.

Data Flow

The system is divided up into five logical services that allow for the entire application to run.

Edge Firmware (the camera)

- On boot, it connects to the wifi and triggers a webhook on the preconfigured server with its id to register itself.
- When the motion sensor goes off it triggers two actions:
    - It streams video over wifi to the preconfigured server
    - Records the same video to the local micro-SD card as fallback in case of footage being lost if the network drops)

Entry Web Server (Bun + Express)

- Server listens for incoming data.
- When data is incoming
    - It creates a unique object key in blob storage.
    - Pipes the incoming video stream to Minio.
    - When the stream ends, it inserts a row into the events table in postgres with a unique id, timestamp, status and ID in blob storage.
    - Sends a request to the python web server with the id of the video.

AI Processing Workers

- Python web server listens for requests from the express server.
- Upon receiving a request, it downloads the video from blob storage and takes multiple images throughout the video to load into different AI models.
- One model is used for detecting if each image contains a person, vehicle, animal or package.
- Second is used for face detection using cosine similarity against known faces in another postgres table.
- A rule engine decides if the event could be described as interesting and worth notifying the user off.
- If interesting. Three of the most representative frames are send to a multimodal model such as Phi-3.5-vision-instruct or similar and a natural language caption is created.
- All metadata is added to the videos record in postgres and the express server is notified of its completion.

Notification Service

- NTFY or similar has permission on users phone/PC to send notification.
- Once the express server has received a request from the python web server notifying it that the event is worth awaring the user of, It will hit an endpoint on NTFY with the event description which the user will promptly receive on their device(‘s) of choice.

Frontend

- The frontend will us a tunneling service such as ngrok or similar to allow the user to access it from wherever they are without the need to configure DNS.
- A Progressive web app will provide the user with the ability to navigate through all of the events recorded by the camera.
- There is a face management section where the user can see through all of the faces the the facial recognition model has detected and assign a name and whether or not it should notify the user of their presence.

Offline and resilience behaviour

- The core focus is to make sure that if an event occurs while there is no network connection, it is still recorded and can be accessed by the user. This is done by recording to the built in microsd card on the device and uploading it once a network connection is obtained.

4.4 Design System

This subsection will present technical designs derived from the requirements outlined earlier. The design follows the structured, layered approach mentioned earlier.

Hardware design

The physical hardware should be easy to obtain, and the 3d printed housing should be easy to put together to allow the user to put everything together themselves within 5 minutes provided that they have all the necessary parts. The hardware will include:

- The ESP 32 CAM module
- The HC-SR501 PIR motion sensor module
- A microsd card to plug into the pre-existing slot in the ESP32 CAM
- A micro usb cable that can be plugged into the camera and another power source.

All of these electronics will be wired togher using color coded DUPOont cables and the user will be provided a source of easy to follow assembly instructions.

Firmware design

This firmare will be written in Arduino-C++ using platformIO and the final version will have a web interface through which the connected ESP32 can be flashed. Its key function will be

- Deep-sleep mode for power saving when not active.
- When the motion sensor goes off, the device wakes and video is streamed to the pre configured web server until the motion sensor no longer detects anything.
- If network connection is missing, fallback to recording to the sd card.

Backend Service Design

The backend services include:

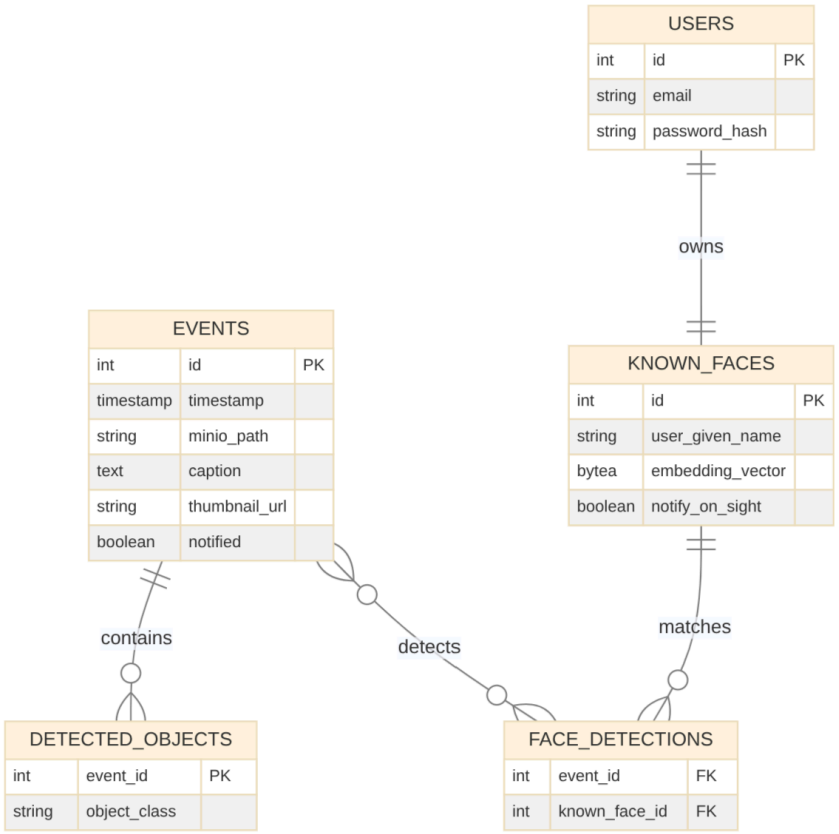
- Express server which acts as a gateway which receives and stores the video stream, creates the event record, and queues the AI job.
- MiniIO is used for blob storage which stores the raw video with a unique key.
- A postgresql database stores the record of the event as well as event metadata and face embeddings.
- The AI worker/ python server runs a variety of models including
    - YOLOv8n-int8
    - InsightFace
    - Phi-3.5-Vision-4bit

- A supabase client handles authentication for the client side application.

- An NTFY client is used to send notifications to the user

Database Schema

- events – id, timestamp, minio_path, caption, thumbnail_url, notified (bool)
- detected_objects – event_id ↔ object_class (person, dog, package, etc.)
- known_faces – id, user_given_name, embedding_vector, notify_on_sight (bool)
- users – id, email, password_hash (Supabase handles auth separately)



-

Frontend

- The frontend is a sveltekit based Progressive web application that uses tailwindcss for styling.
- It will contain at least these pages
  - Login/ dashboard
  - Live camera view
  - Event timeline and history
  - Face management page
  - Setting

4.5 Privacy Design

- All traffic between the camera and the server uses HTTPS
- No footage ever leaves the users network unless the user specifically chose to use the cloud server.
- Authentication with supabase makes sure that only that specific user is allowed to access their data.
- Face embeddings are stored exclusively on the users network unless they allowed it to be stored in the cloud where it will be strictly allowed only to the user and people who they permit to access it. A Time To Live will also be applied on the embeddings making sure that they do not stay on the system for longer than permitted.

4.6 Conclusions

Chapter 4 has transformed the requirements and high-level architecture from Chapter 3 into a detailed, implementable design. Every functional requirement now maps to specific hardware components, firmware functions, backend services, database tables, or frontend views. The resulting design remains fully consistent with the original aims of privacy, affordability, ease of assembly, and high-quality natural-language notifications while staying within the single-camera scope and non-functional constraints defined earlier

5. Testing and Evaluation

5.1 Introduction

This chapter outlines the testing and evaluation strategy that have and will be followed for this project. Testing focuses on 5 aspects that directly map to the project's core objectives and requirements (both functional and non functional).

1. Functionality: Does every feature work as intended
2. Performance and latency: does this meet the performance requirements expected of it. For example does the motion detection to notification time meet the 10-15 second requirement
3. Privacy: Are the privacy needs of each user met.
4. Usability: Is installation, setup and general usage as easy as it can be.
5. Reliability: Does the system continue to work in spite of network failures.

5.2 Plan for Testing

The testing plan combines a variation of automated unit/integration tests, scripted systems tests as well as real world tests to make sure the system is as robust as it could be.

Unit testing

Unit test will be used to verify that the code the correctness of individual functions in isolation ensuring that each block of code works as intended before they are combined with others. This allows you to preemptively prevent issues from appearing in the larger system by making sure individual components work as intended and when an uncaught issues is found and fixed, you can implement new tests to verify that they do not break again.

Integration tests

Integration tests will confirm that separately developed components work correctly together when connected. Custom testing scripts will simulate the full pipeline allowing you to catch issues before deploying them.

Performance Tests

Performance benchmarks will be used to measure the end to end latency from the motion detection to notification delivery. This will allow use to examine what parts of the system are taking longest to run thus allowing us to identify and optimise the system as a whole by making individual components more efficient.

Offline Resilience Tests

By placing stress upon the system by simulating events such as power outages, and network shortages, we can determine how resilient the system as a whole is based off of whether it goes back to working as intended the moment the stressors are resolved.

Usability Tests

Usability tests will have us test how non-technical participants will struggle with setting up the device and using the software. The design of the device and User interface will then be iterated upon to accommodate for the issues the user had.

These tests will measure the amount of time it takes a user to complete a task, the error rate (the amount of users who make errors throught the process) and the failure rate (the number of users who fail to complete a task). With this data we can find key points in the process where users consistently fail and fix them.

5.3 Plan for Evaluation

This section describes how each major part of the system and the system as a whole will be evaluated to prove that the project has succeeded in meeting its aims and goals.

ESP32 Cam

Evaluation will focus on reliability and effectiveness. It will be kept running for an extended period of time to see if it can function without stopping. 2 cameras will be placed side by side connected to separate systems to test consistency.

Video Ingestion and storage Layer

This subsystem will be evaluated based of off the success rate at which the files can be uploaded in their entirety and retrieved. The key focus will be how the files are dealt with when a network connection is lost between the camera and upload server. Optimally, the file upload should restart once the connection is regained.

AI processing pipeline

This is a core part of the system and one that is likely to not work as intended. It will be tested to make sure that each feature works as intended as well as benchmarked to make sure that everything runs at an acceptable speed.

Notifications system

The notification system will be evaluated by having the camera working in a real world environment and have it so that the user can categorise notifications as useful or not when they receive us, this will allow us to check and fine tune multiple things including the false positive rate of the entire system. Testing for false negatives however is far more difficult so the system will be optimised for decreasing the false negative rate at the cost of increased false positives.

Frontend and User Experience

The frontend will be trialed at the same time as the live test where we will see using basic analytics tools how much time is spent on different pages. Live user testing will be done with other students to see how intuitive the user interface is.

5.4 Conclusions

The testing and evaluation strategy presented in this chapter is comprehensive, practical, and directly aligned with the project's core goals of privacy, affordability, speed, ease of use, and reliability. By using a mixture of automated tests, controlled benchmarks and real time deployments of the system, enough proof will be created to show that this system meets all of the requirements that were set out for it.

The results of these tests will be presented in the final report and will conclusively demonstrate that the system is reliable, fast and robust.

6. System Prototype

6.1 Introduction

This chapter will document the current state of the working prototype as of november 2025 and will showcase the hardware and software used to create a minimum viable product.

As specified before the system is composed of:

- The ESP32 CAM and motion sensor
- The Express backend that acts as a gateway for the camera
- A postgress database, MiniIO blob storage and supabase client for auth.
- A python web server with the vision models running
- An ntfy client for notifications
- A sveltekit frontend

6.2 Prototype Development

The upload function for the ESP32 CAM

```
void captureAndUploadPhoto() {
  camera_fb_t * fb = NULL;
  fb = esp_camera_fb_get();
  if (!fb) {
    Serial.println("Camera capture failed");
    return;
  }

  Serial.printf("Captured image, size: %zu bytes\n", fb->len);

  WiFiClientSecure *client = new WiFiClientSecure;
  if (!client) {
    Serial.println("Failed to create secure client");
    esp_camera_fb_return(fb);
    return;
  }

  client->setFingerprint(serverFingerprint);  // For TLS verification

  HTTPClient https;

  if (https.begin(*client, serverURL)) {
    https.addHeader("Content-Type", "image/jpeg");

    // Create multipart boundary
    String boundary = "----ESP32CamBoundary";
    String head = "--" + boundary + "\r\n";
    head += "Content-Disposition: form-data; name=\"file\"; filename=\"photo.jpg\"\r\n";
    head += "Content-Type: image/jpeg\r\n\r\n";

    String tail = "\r\n--" + boundary + "--\r\n";

    // Full body size
    size_t bodyLength = head.length() + fb->len + tail.length();

    https.addHeader("Content-Type", "multipart/form-data; boundary=" + boundary);
    https.addHeader("Content-Length", String(bodyLength));

    // Start POST
    int httpCode = https.POST((uint8_t*)(head.c_str()), head.length());
    if (httpCode > 0) {
      // Send image data
      https.writeToStream((Stream*)fb->buf, fb->len);
      // Send tail
      https.writeToStream((Stream*)tail.c_str(), tail.length());
      https.end(); // This sends final chunk and gets response
    }

    // Get response
    if (httpCode == HTTP_CODE_OK || httpCode == HTTP_CODE_CREATED) {
      String response = https.getString();
      Serial.println("Upload successful!");
      Serial.println("Server response: " + response);
    } else {
      Serial.printf("HTTP POST failed, code: %d\n", httpCode);
      Serial.println("Response: " + https.getString());
    }

    https.end();
  } else {
    Serial.println("Unable to connect to server");
  }

  delete client;
  esp_camera_fb_return(fb);
}
```

This function is triggered when a high state is detected from the motion sensor and it takes an image and sends it to the web server. Right now for testing and validation purposes, only an image is being sent, this is to maximise the resolution of the data that the python server receives so that it can work in optimal conditions. Before the creation of the final report some optimisations or hardware changes may be made to facilitate high quality video streaming.

Entry Express server, File storage

```
app.post('/upload', (req, res) => {
    const bb = busboy({ headers: req.headers });

    bb.on('file', async (name, file, info) => {
        const { filename, mimeType } = info;
        const fileId = uuidv4();
        const key = `${fileId}-${filename}`;

        try {
            // 1. Stream Upload to MinIO
            const parallelUploads3 = new Upload({
                client: s3,
                params: {
                    Bucket: process.env.MINIO_BUCKET_NAME,
                    Key: key,
                    Body: file,
                    ContentType: mimeType,
                },
            });

            await parallelUploads3.done();

            const s3Url = `${process.env.MINIO_ENDPOINT}/${process.env.MINIO_BUCKET_NAME}/${key}`;

            // 2. Create DB Entry in 'events' table
            // Schema based on report: id, timestamp, status, s3_url (blob_id)
            const insertQuery = `
        INSERT INTO events (id, timestamp, status, s3_url)
        VALUES ($1, NOW(), 'processing', $2)
        RETURNING id;
      `;
            // We use the UUID as the DB ID as well for consistency, or let DB generate one.
            // Let's use the generated UUID for the DB ID to ensure we have it before insertion if needed,
            // but here we are inserting. Let's assume ID is a UUID column.
            const dbRes = await db.query(insertQuery, [fileId, s3Url]);
            const dbId = dbRes.rows[0].id;

            // 3. Forward to Python Server
            // We send the DB ID and the S3 URL (or Key) so the python server can download it.
            // Python server expects: id, url, key
            axios.post(process.env.PYTHON_SERVER_URL, {
                id: dbId,
                url: s3Url,
                key: key
            }).catch(err => console.error('Python Server Error:', err.message));

            // We respond to the camera immediately after upload is done.
            // Note: If multiple files are sent, this might need adjustment to wait for all.
            // Assuming single file upload per request.
            res.json({ success: true, id: dbId, message: 'File uploaded and processing started' });

        } catch (error) {
            console.error('Upload Error:', error);
            // If headers already sent (e.g. if we had multiple files and one failed), this might crash.
            if (!res.headersSent) {
                res.status(500).json({ error: 'Internal Server Error', details: error.message });
            }
        }
    });

    bb.on('error', (err) => {
        console.error('Busboy Error:', err);
        if (!res.headersSent) {
            res.status(500).json({ error: 'Upload Failed' });
        }
    });

    req.pipe(bb);
});
```

The function above triggers when the camera streams to the upload endpoint, it takes the stream being sent to it and streams it to blob storage, upon completion a new record is created n the database and a request is made to the python server.

AI Processing Pipeline

Object Detection was implemented using YOLOv8n-int8, It checks if an object matches one of the selected types (person, car, dog, etc) and if it doesnt, drops it.

```
results = yolo_model(frames, verbose=False)
detected_classes = set()
for r in results:
    for box in r.boxes:
        cls_id = int(box.cls[0])
        detected_classes.add(yolo_model.names[cls_id])

if not detected_classes.intersection({"person", "car", "dog", "cat", "package", "truck"}):
    await update_status(event_id, "ignored")
    return
```

The natural language notification message generation is done using Phi-3.5-Vision 4-bit. This is a locally runnable LLM which will provide detailed descriptions of the images without using excessive amounts of compute.

```
# 3. Generate natural language caption with Phi-3.5-Vision 4-bit
caption = llm.create_chat_completion(
    messages=[
        {"role": "system", "content": "Describe this security camera footage in one clear, natural sentence for a push notification."},
        {"role": "user", "content": [
            {"type": "text", "text": "Describe what is happening. Use names if known."},
            *[ {"type": "image_url", "image_url": {"url": frame_b64}} for frame_b64 in encode_frames_base64(frames[:3]) ]
        ]}
    ],
    max_tokens=64,
    temperature=0.3,
)["choices"][0]["message"]["content"]
```

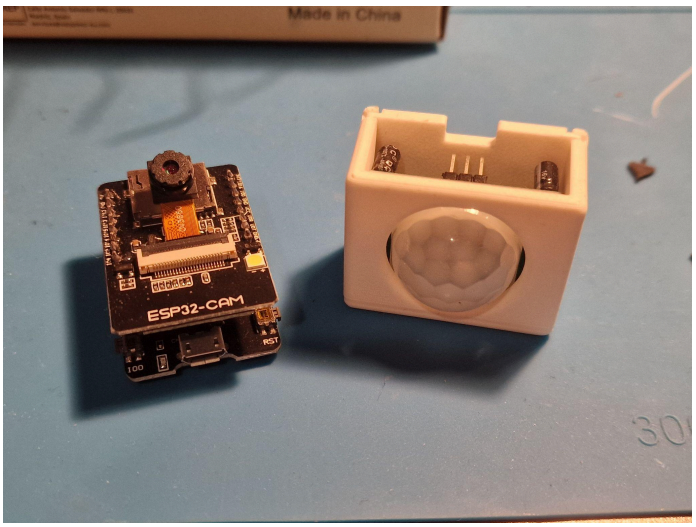
Frontend Snippets

The fronend app is veery simple as of right now, it has a grid containing events in the database that the user can go through.

```
{#each events as event}
  <Card class="mb-4">
    <video src={minio.presignedUrl(event.videoKey)} controls />
    <p class="font-medium mt-2">{event.caption}</p>
    <small>{format(event.timestamp, 'PPP p')}</small>
  </Card>
{/each}
```

Physical Hardware

The physical hardware can be seen below



The case for the sensor above is 3d printed and designed to slide into the main camera body. The body will roughly look like this:

With the interior containing racks on which the sub components will slide in on. The goal is to make it as easy as possible for the user to put the items in without making mistakes and creating confusion.

6.3 Results

Test performed on the current version of the prototype have show that it is lacking in almost all of the departments previously mentioned. One unexpected issue that was encountered was that the quality of the video sent to the server was sub-par to an extent that it was causing

issues in the AI processing step. This resulted in me switching to photos only for the prototype while working on a fix to allow for video streaming.

The idea of users constructing the camera while possible right now, is not easy by any stretch of the definition. The slide in place concept is a result of my finding that placing all of the components into one housing causes a lot of confusion so the goal is to make it really easy for users to place the components into sub components, slide them in, wire them up and be finished.

6.4 Evaluation

Evaluation shows that while there is a lot left to be desired, some key functionality works and works consistently. For example, the upload function that creates the database entry and uploads the data to a bucket does not fail.

When the endpoint is hit NTFY will always send notifications.

Currently there is no function in the ESP32 that makes it resilient to network shortages and the system itself requires some configuration to automatically reboot after a power shortage but these are fixes that can be implemented with relative ease.

Overall, demos of the device and its functionality have impressed users even if it is not what we could consider production ready yet.

6.5 Conclusions

While there is a lot to be left desired, this project is on pace to be functional, affordable and reliable by the end of 2025. The majority of the work left to be done involves making setup easy, accounting for edge cases and improving the user experience when setting up the device and when using it.

7. Issues and Future Work

7.1 Introduction

This chapter reflects on the current state of the prototype and the development process and identifies the problems and issues encountered so far. It will also address how the remaining time we have will be used to develop a better, more robust and polished device and system with a detailed roadmap.

7.2 Issues and Risks

ESP32 low video quality

By far the biggest issue and risk stems from the poor video quality from the ESP32 CAM. It advertises the ability to do 20fps of 720p video but in the environment it is currently in it is incapable of even doing just 1 frame per second. This issue needs to be thoroughly examined to find the root cause as the current quality of the video stream coming from this device is unacceptable.

Testing how the quality of the video can be improved is instrumental to providing the user with a genuinely useful product. Tests have been performed on the algorithm and functions used to send the video but no solution has been found as of yet. It is within reason to expect needing to switch to some other SOC in order to provide a better video stream however due to the abundance of online documentation showing that the current SOC is running to specifications, its malfunction is likely due to human error and not the devices inability to operate.

Physical Assembly difficulties.

Due to the focus on developing the system itself, the assembly of the hardware has been overlooked, as stated before, a slide in rack system is being investigated to see we can simplify the assembly process by breaking it down into instructions in this form:

- Take electronic and place into small container
- Close the container
- Slot into main camera body
- Attach color coded wires

With this assembly process, users may be able to put everything together with only written instruction in 5 minutes and have everything up and running in 15.

No automatic recovery of offline recording

As of right now, if there is no network connection, the camera will not save video to the sd card and attempt an upload at a later time. It is crucial that this is implemented by the time the final report is complete due to the security implications present when a camera doesn't work when the wifi is down.

Night time footage issues

Partly due to the aforementioned camera issues, low light image quality, even when using the extremely bright built in LED is too low to get any meaningful data from. This needs to be investigated and fixed because it is important that the camera is able to record regardless of the conditions its in.

This will likely be fixed once the issues with the cameras image/video loading times is fixed as we will be able to take high quality images in the dark environments that are not made unusable due to artifacting and compression.

7.3 Plans and Future Work

There is a considerable amount of work to be done in the future, the highest priority tasks that needs to be sorted out is making sure that the SOC provides the highest quality data that it can. Many of the issues and roadblock the project has so far will be resolved once that happens.

In the circumstance in which there is a fundamental hardware limitation preventing high quality footage from being obtained. A new SOC will need to be acquired that meets the specifications required for this project.

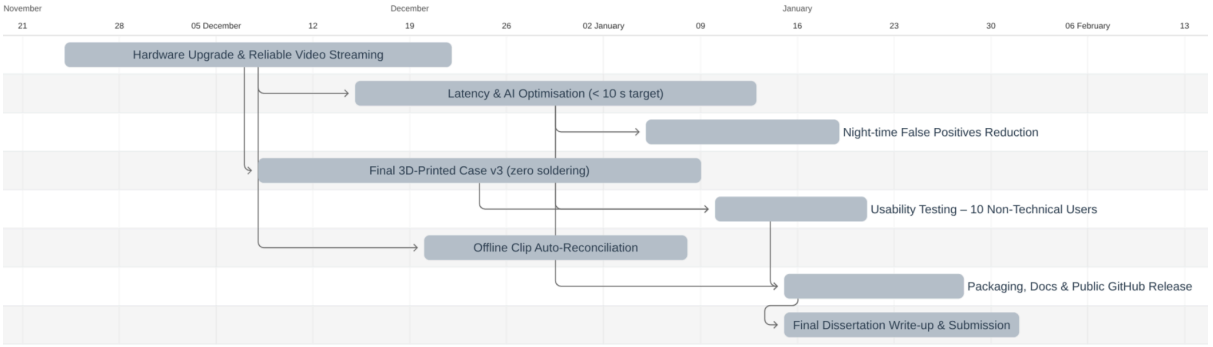
Another important tasks that needs to be completed is the containerisation and creation of a simple, one click deployment solution that allows anyone who really wants to prioritise privacy to be able to run the entire system in their own home. This will involve multiple different moving parts, most important of which is a way to configure the camera and the URLs of the web servers that it sends its data to.

Designing the new housing for the components is not the highest priority issue that needs to be solved but it most certainly isn't to be ignored. Overall, this will take some iteration and testing with users in order to get the average assembly time down.

Partially related to the last point, the process of flashing the SOC with the firmware it needs to run needs to be made far simpler as it currently is using platform IO's upload functions. Implementations online exist where you can configure some variable in a form in a web interface and can flash an ESP32 through a web UI. This is likely the approach that will be taken for this project as it allows the UX of the flashing process to be modified to be made simpler for less tech savvy users.

Finally, additional polish needs to be applied throughout the system in order to be able to present something that is deemed not just acceptable but remarkable. The user interface for one, needs to be not just intuitive but also interesting in the way which it presents data.

7.3.1 Project Plan with GANTT Chart



References

Hint:

Use Zotero to manage your references (see Brightspace resources).

Use the **Harvard** or **IEEE** numbering referencing style

- https://www.zotero.org/support/quick_start_guide

A) Appendix A: System Model and Analysis

Details of Requirements gathering and analysis method

B) Appendix B: Design

Details of design approach used

System elements, components, classes

C) Appendix C: Prompts Used with ChatGPT

“Give me a history of product in the home security space since 2005”

what software methodology should i claim to use, take into cosideration that this is a solo project

D) Appendix D: Additional Code Samples

Provide additional code samples

Organised by Logical Architecture

E) Appendix E:

Other Appendix