DEVELOPMENT OF A SURVEILLANCE SYSTEM WITH MOTION DETECTION, CLOUD LOGGING AND AUTOMATED ALERTS

OLA AYODEJI EBENEZER
(19/30GR039)

A REPORT SUBMITTED TO THE
DEPARTMENT OF COMPUTER ENGINEERING
FACULTY OF ENGINEERING AND TECHNOLOGY,
UNIVERSITY OF ILORIN, ILORIN, NIGERIA.

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF BACHELOR OF ENGINEERING (B.Eng.) IN COMPUTER ENGINEERING

SUPERVISED BY:
PROF. A.T. AJIBOYE

28 JULY, 2025

DECLARATION

I hereby attest that, under the supervision of Prof. A.T. Ajiboye, I carried out all the work detailed in this report, submitted to the Department of Computer Engineering, Faculty of Engineering and Technology, University of Ilorin, Ilorin, Nigeria. All my sources of reference are properly cited and acknowledged.

OLA, Ayodeji Ebenezer

19/30GR039

……..

Signature & Date

APPROVAL

This is to certify that the project report titled "Development of a Surveillance System with Motion Detection, Cloud Logging and Automated Alerts" submitted by OLA, Ayodeji Ebenezer with matriculation number 19/30GR039 in partial fulfillment of the requirements for the award of Bachelor of Engineering (Hons) in Computer Engineering at University of Ilorin, Ilorin, Nigeria, is a record of original work carried out by him under my supervision and guidance. The report has been reviewed and is hereby approved for submission.

_____ _____

Prof. A.T. Ajiboye **DATE**

Project Supervisor

_____ _____

Prof. J.F. Opadiji **DATE**

Head of Department

_____ _____

External Supervisor **DATE**

DEDICATION

This project is dedicated to God Almighty, who has given me the wisdom, grace and strength to see this journey through. To my parents for their love, encouragement and sacrifice, which has been my greatest motivation.

And to the University of Ilorin, for providing the platform, resources and environment that made this academic journey possible.

Thank you all for being pillars of support through every step of this academic endeavor.

ACKNOWLEDGEMENT

I wish to sincerely appreciate God for life and the grace He supplied throughout my years in this great Institution, the peace of mind needed for learning, and the strength to persevere when faced with challenges.

I appreciate my parents Pst. Solomon Ola and Pst. Mrs. Toyin Ola, for their unwavering support, guidance, immense love and care, and for being the financial backbone of my endeavors in the field of Engineering.

I acknowledge the Head of Department, Department of Computer Engineering, Prof. J.F. Opadiji, for his guidance and tutoring and for his laudable efforts to ensure the learning experience of every student was up-to-par with world standards. I also appreciate my supervisor, Prof. A.T. Ajiboye for his perceptive leadership and supervision in the course of carrying out the project. I extend my thanks as well to the entire staff of the Department of Computer Engineering who have in one way or another contributed to the success of this project, Engr. J.A. Adesina for his guidance and continual support, and Engr. H.O. Mahmud for pointing me in the right direction and offering a professional perspective in all matters.

I also sincerely appreciate my partner, Nabila Oyiza Balogun, whose cooperation was instrumental to the project's success.

Thank you all!

ABSTRACT

In recent times, security has become of primary concern in residential, commercial, and industrial settings. Thus, the demand for intelligent surveillance systems has been on the rise to safeguard spaces from intrusion, theft and other attacks. Various solutions have emerged over the years, each offering its own strengths and having its limitations. This project titled **"Development of a Surveillance System with Motion Detection, Cloud Logging and Automated Alerts"**, aimed to produce a motion detection-based surveillance system that integrates cloud-based data storage and automated alerts to provide real-time security monitoring. At its core, the proposed system utilizes an ESP32 camera module paired with a Passive Infrared (PIR) sensor to detect unauthorized movement in a designated area. Once motion is detected, the system captures relevant footage, securely logs event data to a cloud-based storage platform, and instantly notifies users via email alerts. The cloud integration ensures remote access to surveillance records, allowing for efficient incident review and response. The surveillance system includes a web application for users to remotely access logged events and captured images. It provides a secure, user-friendly interface to review motion detection records, filter logs, and view stored footage. This facilitates convenient, remote monitoring from any internet-enabled device, enhancing security management and responsiveness. Designed with affordability and scalability in mind, the system leverages cost-effective hardware and software to offer a versatile security solution adaptable to various environments. The system is structured to provide an efficient and responsive approach to modern surveillance needs, satisfying all the objectives of the project.

TABLE OF CONTENTS

TABLE OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Background

The goal of surveillance systems is to protect spaces of value from unauthorized access by monitoring activity in such areas and providing a means for security personnel or system administrators to effectively and promptly respond to potential threats. Over the years, various innovations have emerged that enhance surveillance capabilities, offering accessible and effective monitoring solutions for residential, commercial, and industrial environments, from simple CCTV systems to AI-powered smart surveillance that can detect and identify objects in real time.

One common issue with traditional surveillance systems is their inefficiency in, storage, power and data consumption. Many conventional setups record continuously, capturing hours of footage even when no activity occurs within the monitored area. This results in unnecessary storage consumption, increased data usage for cloud-based systems, and continuous power draw, making such solutions less sustainable and costly in the long run. Additionally, security teams are often burdened with reviewing large amounts of irrelevant footage, reducing the speed and effectiveness of investigations.

To tackle these issues, modern surveillance technologies have evolved to incorporate motion detection, cloud-based storage for optimization, and automated

alerting mechanisms (Abordo, et al., 2024). Motion-activated recording reduces redundant footage by only capturing events when movement is detected, significantly optimizing storage and bandwidth usage. Cloud logging allows for remote access to surveillance data without the need to spend more on storage, improving scalability and data security. Furthermore, automated alert systems ensure that users are immediately notified of suspicious activity, reducing threat response time.

The goal is to develop a motion detection-based surveillance system that integrates these modern features to provide a more efficient, scalable, and responsive security solution. By leveraging an ESP32 camera (ESP32-CAM) module with a PIR (Passive Infrared) sensor, cloud logging, and automated email notifications, the system is designed to overcome the inefficiencies of traditional surveillance setups while remaining cost-effective and adaptable for various applications.

1.2 Aim and Objectives

1.2.1 Aim

The aim of this project is to design and develop a low-cost surveillance system which is capable of detecting motion within a space of interest, keep a record of such events on the cloud, and alert the user automatically and in real-time, when they occur.

1.2.2 Objectives

i. Design a motion-activated surveillance system using the ESP32-CAM microcontroller and a PIR sensor.
ii. Implement automatic image capture when motion is detected within a monitored area.
iii. Develop a mechanism to log detected events and associated images to a cloud-based storage platform.
iv. Integrate an automated email notification system to alert users of detected activity in real time.
v. Create a simple web interface for users to view logged events, timestamps, and captured images remotely.

1.3 Problem Statement

Surveillance systems traditionally rely on continuous recording, thus consuming excess storage and using up power and data unnecessarily, especially in cloud-based setups. These inefficiencies not only increase operational costs but also make it difficult for users to promptly identify relevant security events because of to the large amount of irrelevant footage. Also, cheaper surveillance solutions generally lack real-time alert features, making them ineffective for proactive security. There is therefore a need for a smart, affordable, and power-saving surveillance solution that captures and logs only important events, provides timely alerts, and allows users to remotely monitor their establishments without the inconvenience of manual monitoring around the clock.

1.4 Significance of the Study

The development of a surveillance system with motion detection, cloud logging and automated alerts addresses several demands in the world of IoT (Internet of Things). By offering a budget-friendly alternative that is cloud-based and responsive, it satisfies the growing demand for accessible, intelligent systems in security technology, which can reliably provide relevant information when thefts, vandalism and other criminal offences occur. The project improves these major areas in surveillance technology:

i. Logged records are safely stored in the cloud, easy to access when they are needed and are less likely to be lost in contrast to conventional systems that use physical storage devices and are susceptible to theft, damage, or tampering (OPSWAT, 2024).
ii. Real-time notifications enhance the responsiveness of not only the system itself, but also the user and relevant personnel who will be able to handle the security threat in time to prevent further damage or escape of perpetrators.
iii. The potential of integrating low-cost microcontrollers with modern web technologies to solve real-world problems is harnessed, making it suitable for various applications.

1.5 Scope of the Study

This project covers the development of a smart surveillance system using an ESP32-CAM to detect motion, capture images, and transmit them to a Node.js backend. The backend handles image upload to Cloudinary, logs events in a MongoDB database, and sends real-time alerts via email. A web interface also gives

users access to event logs and images. The system is designed for small-scale use, such as in homes or single-location setups. Advanced features like facial recognition, video streaming, or multi-camera networks are beyond the scope of this study.

1.6 Project Organization

This report consists of five chapters, organized in sections. Different aspects are treated by each chapter in detail and relevant references are provided. The consisting chapters are:

Chapter One (Introduction) introduces the project by outlining the background, problem statement, objectives, and scope. It sets the foundation for the work done.

Chapter Two (Literature Review) reviews existing research and technologies related to the project. Identifies gaps and justifies the need for the proposed solution.

Chapter Three (Methodology) explains the tools, design approach, and processes used in building the system, including system architecture and workflow.

Chapter Four (Implementation, Testing and Result) details how the system was developed and tested. Includes implementation steps, testing procedures, and analysis of results.

Chapter Five (Conclusion and Recommendations) summarizes the project's outcomes, reflects on objectives achieved, and suggests improvements or future work.

CHAPTER 2

LITERATURE REVIEW

2.1 Overview

Over the years, surveillance solutions have helped to secure valuable spaces and improve response times to threats or intrusions in sensitive areas. These solutions have undergone remarkable transformations and upgrades to their capabilities. Surveillance systems are now smarter, faster, more lightweight and highly interconnected in comparison to solutions which were previously deployed.

Initially, surveillance systems were analog, making use of closed-circuit television (CCTV) cameras which recorded footage on tapes, the predominant storage medium at that time. However, they had limited storage capacity, low image quality and the need for manual review of footage made their operation inconvenient. Nevertheless, they laid the foundation for the global adoption of surveillance technology across domestic and industrial areas.

Digital technology significantly transformed the capabilities of surveillance systems, replacing analog tapes with Digital Video Recorders, improving image quality, using efficient storage media and providing faster access to recorded footage (National Institute of Standards and Technology, n.d.). The introduction of the internet further enhanced the capabilities of such systems in terms of automation, remote access, automation and scalability.

Today, surveillance systems are no longer passive recording tools but active components in security ecosystems. Coupled with sensors, alarms, General Packet Radio Service (GPRS), and internet technology, they can trigger alerts, contact law enforcement, and even interact with other components in smart buildings such as door locks and lighting.

These applications can easily be implemented through the use of a microcontroller, which can handle inputs from these sensors and initiate responses via the alarms, alerts, etc. This integration enhances the system's ability to detect and respond to various events, providing a comprehensive and intelligent approach to security (Temitope, 2024).

2.2 Motion-Based Surveillance

Activity-based surveillance uses activity detection to trigger footage capture or alerts only when activity or event is identified to have occurred e.g. intrusion in a sensitive space, loud noises, a trip switch being activated by movement, fire, or smoke detection, etc. This eliminates the problem of storage wastage due to continuous recording and shifts focus to relevant events. Event detection can be achieved through a variety of sensors such as Infrared sensors, ultrasonic sensors and proximity sensors, smoke detectors, heat sensors, acoustic sensors, or through the use of AI. According to the National Institute of Standards and Technology (NIST), such intelligent event detection improves the efficiency and scalability of

modern surveillance systems (National Institute of Standards and Technology, n.d.). A microcontroller can be programmed to trigger alarms, contact relevant personnel or prevent access to a space when activity is detected in an area of interest.

2.3 Cloud Data Logging

The process of collecting, storing, and managing data on remote cloud servers rather than on local devices is referred to as cloud data logging. It finds application in surveillance systems, providing a secure medium of data storage, e.g. upload of captured images, video clips and sensor readings to cloud platforms. In the event that the device is damaged or stolen, important data and records are preserved and can be securely accessed from other devices remotely. This improves flexibility and scalability in these systems, ensuring they can handle growing data volumes without the need for physical upgrades. In large-scale deployments, cloud-based storage frameworks provide improved scalability, reliability, and data durability. As noted by Li, Zhang, and Shen (2018), cloud replication techniques can significantly enhance system performance and availability, making the surveillance infrastructure more fault-tolerant and responsive.

Many cloud platforms such as Google Cloud, Amazon Web Services, etc., provide built-in encryption, back-ups and enforce access control to protect the integrity of logged data.

2.4 Data Exchange between Components via APIs

In contemporary software engineering, Application Program Interfaces (APIs**)** facilitate communication between different components or services, following a set of predefined rules. Each component, either an interface, local service or a third-party service may perform a specific task, which may require communication between them. APIs act as a bridge that allow these components to share data and perform actions in a structured manner.

The microcontroller in a surveillance system, when it detects motion or receives an indication of a threat, will need to capture an image or video and somehow upload the video to the cloud for storage and notify the user remotely. This could be achieved by sending a POST request to the server via Hyper Text Transfer Protocol (HTTP). The server then handles the upload and real-time alert independently. We can implement such interactions through Representational State Transfer (REST) APIs which use standard HTTP methods (GET, POST, PUT, and DELETE) to send and receive data in JavaScript Object Notation (JSON) or Extensible Markup Language (XML).

Using APIs for communication brings several benefits:

i. Modularity: Components can be developed, maintained, or upgraded independently.

ii. Scalability: New services can be added or removed without disrupting the whole system.
iii. Reusability: A single API can serve multiple clients or functions.
iv. Security: Authentication and authorization mechanisms can be enforced at the API level.

2.5 Real-Time Automated Alerts

The use of software-driven mechanisms to notify relevant personnel or systems when a relevant event occurs improves the effectiveness of surveillance systems by reducing response times, providing event context and preventing loss of life and property. These alerts can be programmed to be triggered by specific events such as motion detection, environmental changes, intrusion, etc. Notifications may be sent via Short Message Service (SMS), email, mobile push notifications or directly to surveillance dashboards. Modern surveillance systems often make use of AI to distinguish between genuine and irrelevant triggers, thereby reducing false alarms and improving efficiency. For example, the I-SAFE framework ("Instant Suspicious Activity identiFication at the Edge using fuzzy decision making") demonstrates how edge-based processing can drastically reduce alert latency: it identifies suspicious activities in only about 0.002 seconds**,** enabling near-instant notifications and proactive intervention. (Nikouei, 2019).

2.6 Related Works

Several researchers and authors have authored remarkable works on the subject of surveillance systems and many implementations of various ideas exist as well. This section highlights some of the most relevant works related to the scope and objectives of this project.

Okokpujie et al. developed an affordable, real-time IoT-based surveillance system using the ESP32 microcontroller and the Twilio API for alert notifications. The system utilizes motion detection to trigger image capture and sends alerts via SMS to preset contacts. Their research highlights the effectiveness of combining low-cost hardware with cloud-based communication APIs to create real-time surveillance solutions suitable for home and small office environments (Okokpujie et al. 2023). While this was a cost-effective solution, it lacked features such as web-based access, remote image access and cloud data logging, leaving room for improvements in terms of data availability and access.

Abordo et al. (2024) developed a smart surveillance system using the ESP32 microcontroller with camera-based motion detection and IM (image monitoring) technology. Their system focused on real-time monitoring, optimized camera placement, and better server performance compared to traditional CCTV setups. However, the project did not explore cloud storage integration or web-based access to past records. Additionally, real-time alerts and notifications were absent in their

implementation of the system. (Abordo, Pastores, Villaroza, Guerrero, Robles, Mangubat, Borleo, Bautista, & Galay-Limos, 2024).

Ibok, Udom, and Sunday (2024) designed an IoT-based surveillance system that integrates Arduino microcontrollers with Firebase for data logging and remote access. They focused on improving urban security through real-time alerts, remote monitoring, and event data logging. One key strength of their approach is the low cost and ease of deployment in areas with limited connectivity and data resources. However, their implementation lacks advanced threat detection capabilities, is not suitable for multiple users, and heavily depends on AI-based video processing, limiting its usability due to the need for sufficient computational power for AI which its scope does not consider (Ibok, Udom, & Sunday, 2024).

Patil et al. (2023) developed a low-cost surveillance system using the ESP32-CAM module integrated with motion detection and cloud storage capabilities. The system reduces data storage requirements by capturing videos only when motion is detected, and uploading the footage to cloud platforms. It solved the problem of affordability with its compact design, and suitability for small-scale applications. However, the paper does not address network connectivity, data security, or integration of multiple cameras, which are crucial for deployment in larger environments (Patil, Pati, Momin, Patil, & Nalawade, 2023).

Jain, Pandey, and Shrivastava (2020) proposed an IoT system that integrates smart surveillance with home automation. The system utilizes sensors and microcontrollers to monitor environments and control appliances in their vicinity remotely over the internet. A major highlight is its ability to enhance both home security and energy efficiency through real-time decision-making. However, the system lacks a detailed discussion of data privacy, scalability, and robustness under varying network conditions—factors essential for real-world deployment

CHAPTER 3

METHODOLOGY

3.1 Overview

This chapter highlights the development approach and methodology employed in implementing the proposed surveillance system. This project adopted a **modular development approach**, which involves breaking down the entire system into smaller, manageable units that can be developed, tested, and validated independently before final integration. This approach allowed for easier debugging and independent feature implementation, enabling adjustments to individual components without affecting the entire system. These major modules include:

i. **Image Capture Module:** The ESP32-CAM was configured to capture images in response to motion detected by the PIR sensor.
ii. **Data Transmission Module:** This module handles the conversion of captured images to Base64 and their transmission to the backend server over HTTP.
iii. **Event Processing Module:** The web application backend developed using Node.js, receives image data, uploads it to the cloud, and logs event details in the database.
iv. **Email Notification Module**: Using Nodemailer, this module sends real-time alerts to a recipient email when a new event is logged.
v. **Web Interface Module**: A browser-accessible frontend giving access to logged events and their associated images.

vi. **Power Module:** The system is battery powered with a rechargeable 3.7V Lithium-ion (Li-ion) battery and a boost converter to boost the voltage to 5V

3.2 System Architecture

The architecture of the surveillance system is based on a modular client-server model that enables motion-triggered image capture, cloud storage, real-time alerts, and remote access. The system comprises both hardware and software components, working in unison to deliver a reliable surveillance solution which is fully automated. The hardware components of the system include an ESP32-CAM microcontroller module, a Passive Infrared (PIR) sensor, a boost converter, a Li-ion battery and a charging module.

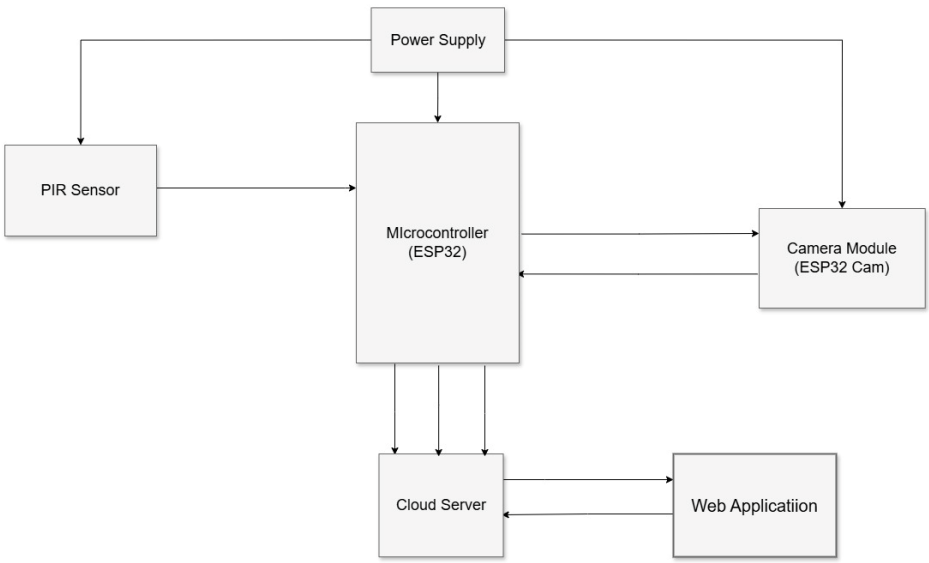

Figure 3.1. Block Diagram of the Smart Surveillance System

3.2.1 Image Capture Module

This module comprises the ESP32-CAM and a PIR motion sensor (HC-SR501), working to detect movement and capture images automatically. The PIR sensor continuously monitors for infrared radiation changes in its field of view. When motion is detected, it sends a HIGH signal to one of the ESP32-CAM's GPIO pins. Upon receiving this trigger, the ESP32-CAM initializes its on-board camera, captures a still image, and encodes it in Base64 format for easy transmission over HTTP. The combination of the PIR sensor and ESP32-CAM ensures that images are only captured when significant movement is detected, optimizing both storage and power usage.

3.2.2 Data Transmission Module

The data transmission module is responsible for relaying captured images from the ESP32-CAM to the backend server. After encoding in Base64 format, the ESP32-CAM sends the image to a predefined backend endpoint via an HTTP POST request. This transmission occurs over Wi-Fi, leveraging the ESP32's built-in 802.11 b/g/n network adapter.

The transmitted data includes the Base64-encoded image string wrapped in a JSON (JavaScript Object Notation) object. This makes transmission over HTTP more seamless as the server API can readily process JSON. Also, the need for intermediate memory on the microcontroller is removed and so is the added

complexity of the firmware program. The ESP32-CAM acts as the HTTP client, while the backend, developed using Node.js and Express, receives and processes the incoming payload. This module ensures reliable, asynchronous communication between the ESP32-CAM and the cloud-based server, taking the image from capture to processing.

3.2.3 Event Processing Module

The event logging module handles the storage and management of image capture records on the backend. Once the backend receives the Base64-encoded image from the ESP32-CAM, it uploads the image to Cloudinary, a cloud-based media storage service. Upon successful upload, Cloudinary returns a secure image URL (Uniform Resource Locator), which is then used to create a structured event log. Each log entry includes the image URL and a timestamp indicating when motion was detected (optionally).

This data is stored in a MongoDB database collection using a model schema (named "Event" on the backend). The event logging module ensures that every motion-triggered capture is registered and can be retrieved later for review.

3.2.4 Email Alert Module

This module is responsible for notifying users when motion is detected within the range of the PIR sensor and upload to the cloud storage bucket is done. It uses Nodemailer to send an automated email which includes the timestamp, a preview

of the captured image as well as a link to view the full image on the user dashboard. This informs the user promptly about potential security breaches resulting in quicker response times.

3.2.5 Web Application Module

This provides a user-friendly interface for accessing the logged events and their corresponding images, allowing users to easily monitor activity, review captured images and verify the existence of a threat and relevance of the alert. It is protected with time-based token authentication and role-based access control to ensure only authenticated users can access event logs and images. Stored data can be accessed easily without direct contact with the microcontroller.

3.3 Hardware Architecture

The hardware composition of the system is both simple, effective and more cost-effective than most surveillance options available. The ESP32-CAM module is a microcontroller which handles the motion detection and image capture logic. It includes a port which can accommodate very specific camera types (OV2640 by default). It also has built-in Wi-Fi capabilities making it the ideal choice for this application. A PIR sensor is included for the motion detection capability required. A Passive Infrared (PIR) sensor works by detecting changes in infrared (heat) radiation within its field of view. It doesn't emit any radiation itself, but rather senses the infrared energy given off by objects, like humans or animals and triggers

a signal when there is a change in radiant energy. In order to power the system efficiently, a Li-ion battery is employed due to its high energy density, low self-discharge, as it can reliably power the ESP32-CAM and sensor for extended periods while maintaining stable output. It is coupled with a boost converter to raise the voltage to the required level to ensure reliable operation of the system.

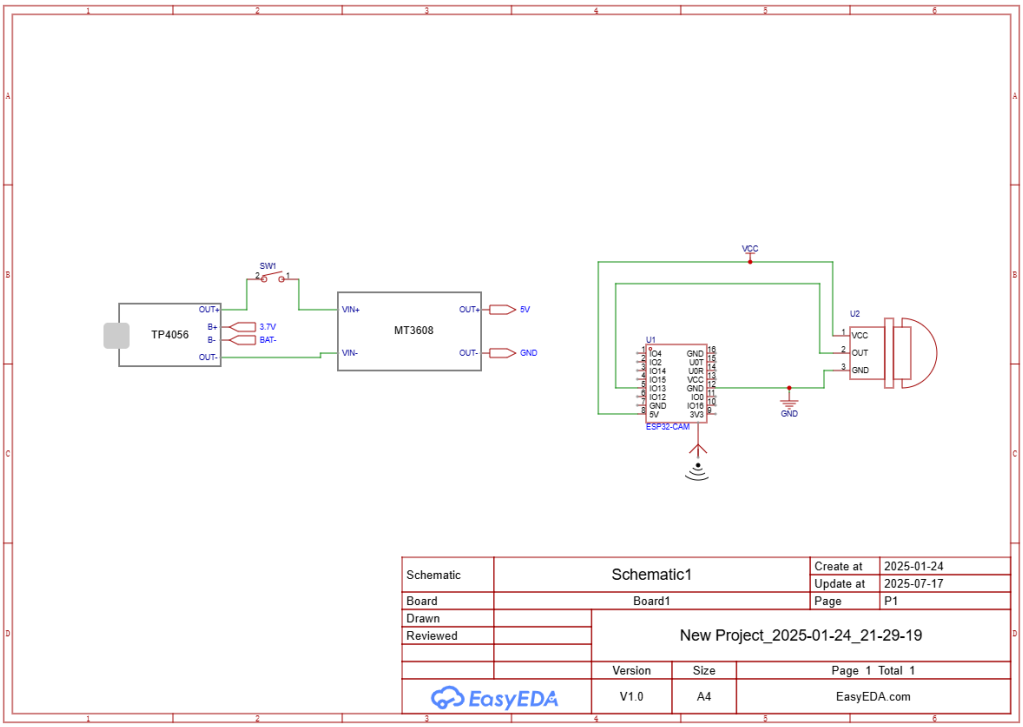

Figure 3.2 Circuit Schematic of the Smart Surveillance System

3.3.1 Hardware Component Specifications

1. ESP32-CAM Module

i. Dual-core Tensilica LX6 @ 240MHz)
ii. Camera: OV2640 2MP camera module
iii. RAM: 520 KB SRAM + 4 MB PSRAM

iv. Flash Memory: 4MB

v. Wireless Connectivity: Wi-Fi 802.11 b/g/n; Bluetooth 4.2 BLE

vi. GPIOs: 10 usable pins (shared with camera interface)

vii. Camera Resolution: Up to 1600 × 1200 (UXGA)

viii. Video Streaming: MJPEG, JPEG, BMP formats supported

ix. Power Supply: 5V input, typically powered via 3.3V regulator

x. Interface: UART, SPI, I2C, PWM, ADC, DAC

xi. Operating Voltage: 3.3V

xii. Operating Temperature: -40°C to 125°C

xiii. Programming Support: Arduino IDE, ESP-IDF

xiv. Dimensions: ~27 × 40 mm

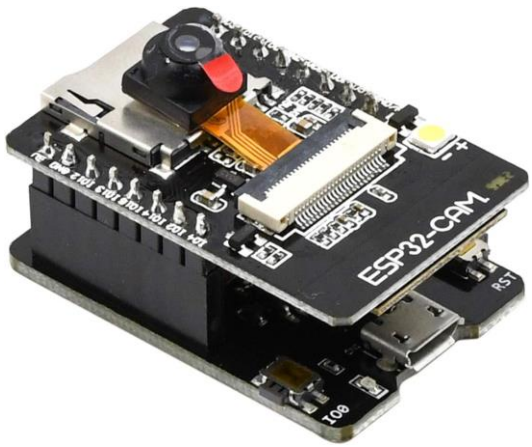

Figure 3.3 ESP32-CAM Module

2. Camera Module (OV5640)

i. Sensor Type: CMOS Image Sensor

ii. Resolution: Up to 5 Megapixels (2592 × 1944)

iii. Output Formats: JPEG, RAW, RGB565, YUV

iv. Lens View Angle: ~60 to 70 degrees (varies by lens)

v. Interface: DVP (Digital Video Port), I2C for control

vi. Frame Rate:

   a. 15 fps @ 2592 × 1944

   b. 30 fps @ 1080p

   c. 60 fps @ VGA

vii. Focus Type: Fixed Focus or Auto Focus (model dependent)

viii. Operating Temperature: -30°C to 70°C

Figure 3.4 ESP32-CAM Module

2. PIR Motion Sensor (HC-SR501)

i. Sensor Type: Passive Infrared (PIR)

ii. Detection Range: 3 to 7 meters

iii. Detection Angle: 120 degrees

iv. Operating Voltage: 5V to 20V DC

v. Output: Digital HIGH/LOW

vi. Trigger Modes: Single and repeatable trigger

vii. Adjustable Delay: 3 seconds to 5 minutes

viii. Sensitivity Adjustment: Potentiometer included

ix. Dimensions: 32 by 24 mm (sensor board only)

x. Applications: Motion detection, home automation, security systems

Figure 3.5 HC-SR501 PIR Sensor

3. Boost Converter (MT3608)

i. IC Used: MT3608 step-up converter

ii. Input Voltage Range: 2V - 24V

iii. Output Voltage: Adjustable up to 28V

iv. Max Output Current: 2A (with proper heat dissipation)

v. Efficiency: Up to 93%

vi. Switching Frequency: about 1.2MHz

vii. Adjustable Potentiometer: For tuning output voltage

viii. Dimensions: ~22 × 17 mm

ix. Application: Voltage step-up for powering 5V devices from Li-ion cells

Figure 3.6 MT3608 Boost Converter Module

4. Charging Module (TP4056)

i. Charging IC: TP4056 lithium-ion charger
ii. Battery Type: 3.7V Li-ion or Li-Po single cell
iii. Charging Current: 1A (default; adjustable by resistor)
iv. Input Voltage: 4.5V to 5.5V (via micro-USB or solder pads)
v. Protection: Overcharge, over-discharge, and short-circuit protection
vi. Charging Status: Two LED indicators (charging/full)
vii. Applications: Portable electronics, battery-powered microcontroller systems
viii. Dimensions: ~25 × 19 mm

Figure 3.7 TP4056 Charging Module

5. 3.7V Li-ion Battery

i. Cell Type: 18650 or flat-cell Li-ion rechargeable battery

ii. Nominal Voltage: 3.7V
iii. Capacity: Typically ranges from 2000mAh to 3000mAh
iv. Max Charging Voltage: 4.2V
v. Discharge Cutoff: About 3.0V
vi. Cycle Life: 500+ charge cycles
vii. Safety: Requires proper BMS or protection module

Figure 3.8 Li-ion Battery

3.4 Software Architecture

The software architecture of the system, as shown in Figure 3.2, is responsible for coordinating the interactions between all components involved in motion detection, image capture, image processing and event logging, automated alerts and log retrieval. Detected motion triggers image capture, after which the image must be converted into a format that can be transmitted over HTTP to a web server. A POST request which contains the details of the motion event i.e., the converted image and event timestamp, is sent and processed by the server which must also handle the

storage of the image either on a local database or a Database-As-A-Service (DBaaS), which is used in this application. Simultaneously, the server must alert the user that an event has occurred either by e-mail, social media or mobile push-notifications.

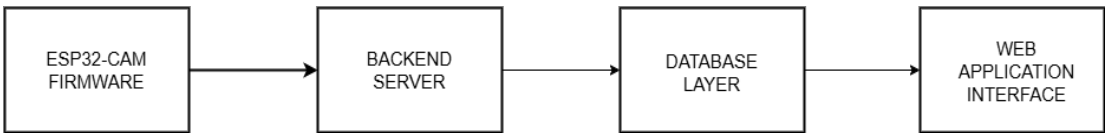

Figure 3.9 System Software Block Diagram

The ESP32-CAM Firmware Layer handles motion detection logic, receiving a HIGH digital signal from the PIR sensor when it detects motion, triggers an image capture, converts the image into base64 format for transmission, and then sends the image via a HTTP (Hyper Text Transfer Protocol) POST request to the backend layer.

The Backend Server Layer exposes an API endpoint that handles POST requests containing the Base64-encoded image data. It processes each request by uploading the image to cloud storage and logging the event in the database, including the image URL and the exact time the event occurred. It is also responsible for notifying the user of detected motion via email.

The Database Layer stores event logs in a structured format, following a specified pattern known as schema. It also safely stores user data and preferences such as the recipient email for alerts and event image URLs.

The Web interface (or Application Frontend) allows the user to view past event logs, their associated images and the date and timestamp the event was recorded at.

Table 3.1 below is a summary of the software layers and the technologies used:

Table 3.1 *Software Architecture Components and Technologies*

Component	Technology	Responsibility
Microcontroller Firmware	Arduino (C++)	Motion detection & capture
Backend Server	Node.js + Express	API, logic, routing
Database	MongoDB	Log storage
Media Storage	Cloudinary	Image hosting
Notifications	Nodemailer on the server	Email alerts
Web Interface	HTML/CSS/JS	Viewing logs

Figure 3.3 below shows the system flowchart, detailing the entire process from when motion is detected, to when the user receives the automated email alerts.

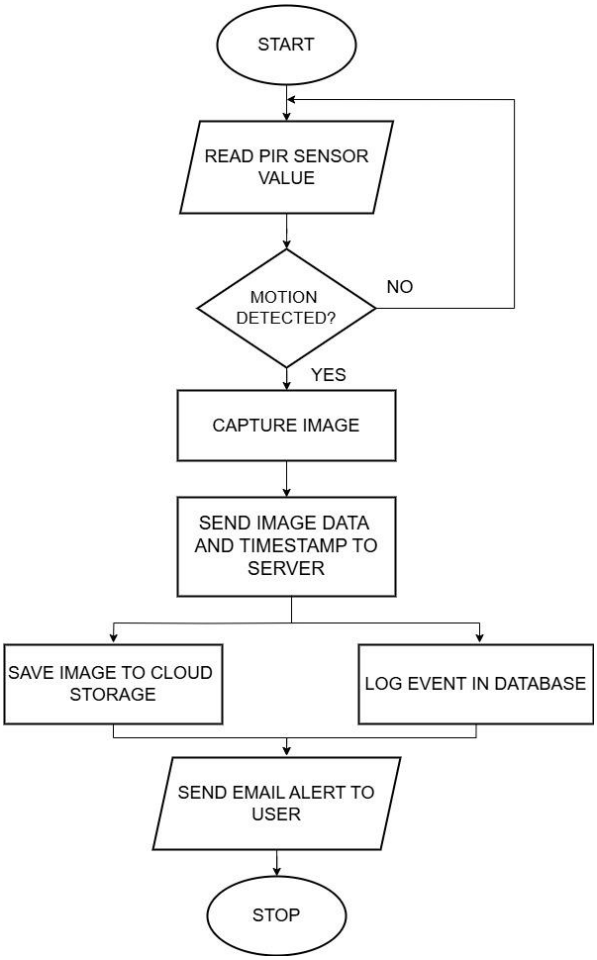

Figure 3.10 System Flowchart

CHAPTER 4

IMPLEMENTATION, TESTING AND RESULTS

4.1 Overview

In this chapter, the implementation, testing and outcomes of the development of the surveillance system, detailing how both software and hardware components of the system were implemented and how they performed in testing. The system's modular design made the implementation more efficient and straightforward.

4.2 Hardware Implementation and Testing

4.2.1 Power Supply Modules

The system is powered by a 3.7V 1200mAh Lithium-ion (Li-ion) battery. The 3.7V supplied by the battery is insufficient to power both the ESP32-CAM and PIR sensor which require 3.3 or 5V, and 5-12V respectively. Hence a boost converter (MT3608) is used to boost the output from the battery to the required 5V minimum. The on-board regulator on the microcontroller ensures that even when 5V is exceeded, it still operates safely at 5V. However, the boost converter's output had to be calibrated using the potentiometer on the module to ensure voltage was not excessive and wouldn't damage the other components. A TP4056 module with a USB Type-C input was employed for battery charging and management. It safeguards the battery against overcharging by cutting off the supply when the

battery is fully charged. The current requirements for each component were also satisfied with the battery output being 1.2A, the charging module supplying 1A and the microcontroller and sensor only consuming about 160mA and about 300mA during image capture or uploads.

Figure 4.1 Surveillance System Power Supply and Charging Circuit

4.2.2 Motion Detection and Image Capture Modules

As described previously, the ESP32-CAM microcontroller is at the center of the system coordinating motion detection, image capture and server upload. The module comes with the OV2640 camera attached and while this camera works, the resolution (2MP) and clarity it produced proved to be inadequate. It was also difficult to adjust its focus as the dial around the lens was glued in place and could not be moved. Thus, it was replaced with an OV5640 module. This required additional changes to the functional firmware code but provided better performance

than its predecessor, with a resolution of 5MP, better clarity and image post-processing.

A Passive Infrared Sensor was used for motion detection. The sensor accurately detects motion within its range which can be adjusted to a minimum of 3 meters and a maximum of 7 meters. It allows selection between single trigger (output goes high once and stays high for a set time) or repeating trigger (output goes high each time motion is detected, restarting the delay timer) modes using 3 jumper pins. The repeating trigger mode was used for this application. Each time motion was detected, the sensor fed a HIGH signal to the microcontroller which instantly captured an image and initiated communication with the backend server to log the records.

Figure 4.2 Final System Integrated Circuit

The final product was implemented and packaged in a 3D-printed enclosure tailor-made to fit this particular application. The enclosure is shown in Figure 4.2 below.

4.2 Software Implementation and Testing

The software components of the system include the ESP32-CAM firmware, server API and the web application interface. Each component was implemented and tested independently, and performs a specific function in the chain of processes the system performs. The server API was built using Node.js and Express.js to handle user authentication, user sessions, image upload, event logging and sending automated alerts. It exposes Representational State Transfer (REST) endpoints which the microcontroller uses to upload images via HTTP. The API is hosted on a third party platform, “Render”, allowing access over the internet, maximizing cloud storage and enhancing application security.

When the server receives a request to upload an image, it verifies that the complete and correct base64 string is what was sent by the microcontroller by carrying out a checksum. The received image is then uploaded to Cloudinary asynchronously while the event timestamp and a secure URL to the uploaded image are stored on MongoDB, the selected database service for the system.

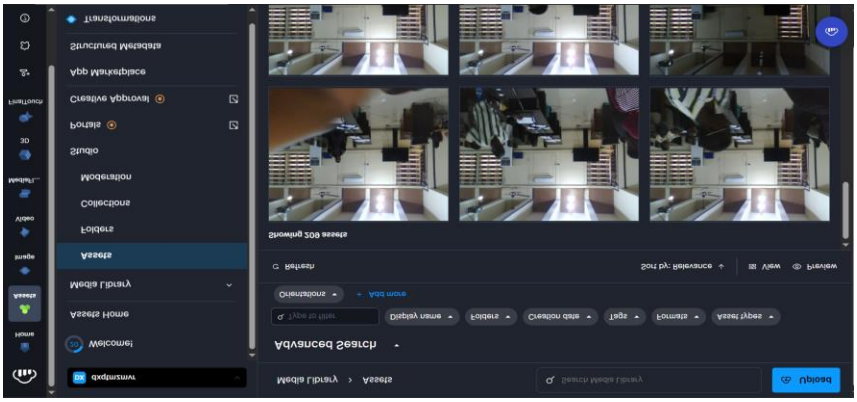

Figure 4.3 Cloudinary Dashboard

```
const EventSchema = new mongoose.Schema({
    imageUrl: { type: String, required: true },
    timestamp: { type: Date, default: Date.now }
});

export default mongoose.model('Event', EventSchema);
```

Figure 4.4 Event Schema for MongoDB Database

The Nodemailer library sends an alert to the user via email, which for most mobile devices, triggers a push notification automatically. Other devices may need to have their notification settings changed to enable such functionalities. The alert is sent immediately the image upload is finished, and the content is generated by the server using a predefined template that uses state to determine dynamic values such as the timestamp and image link.

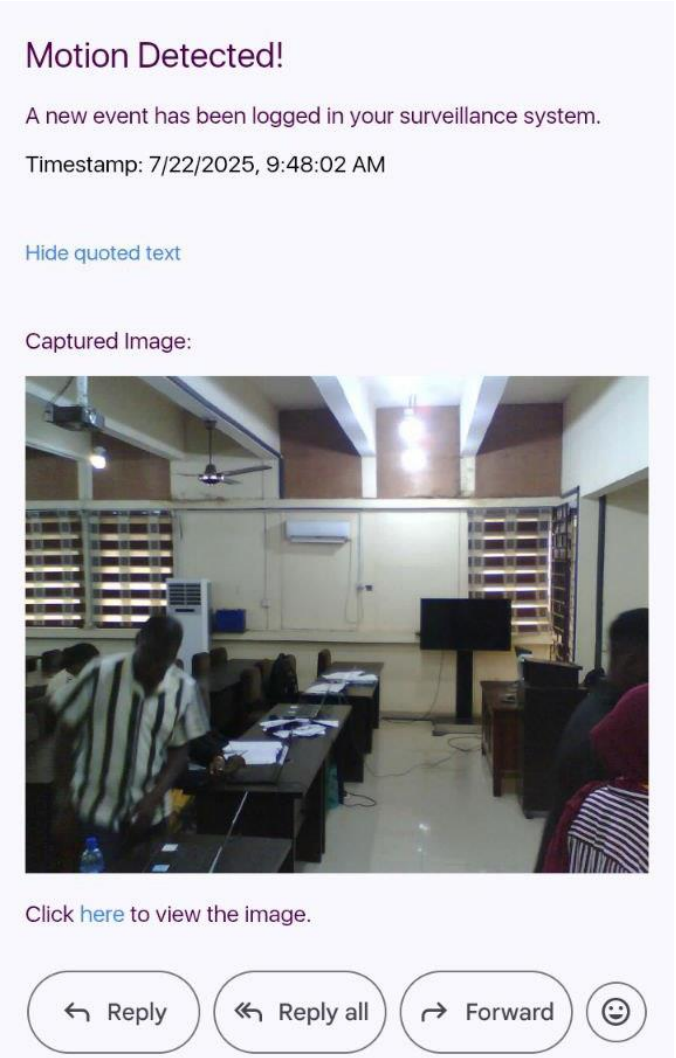

Figure 4.5 Alert Email Template

When the entire process is complete, a response is sent to the microcontroller which indicates whether the upload and alert operations were successful (HTTP response code 200) after which the ESP32-CAM starts a delay of three seconds before resuming motion detection.

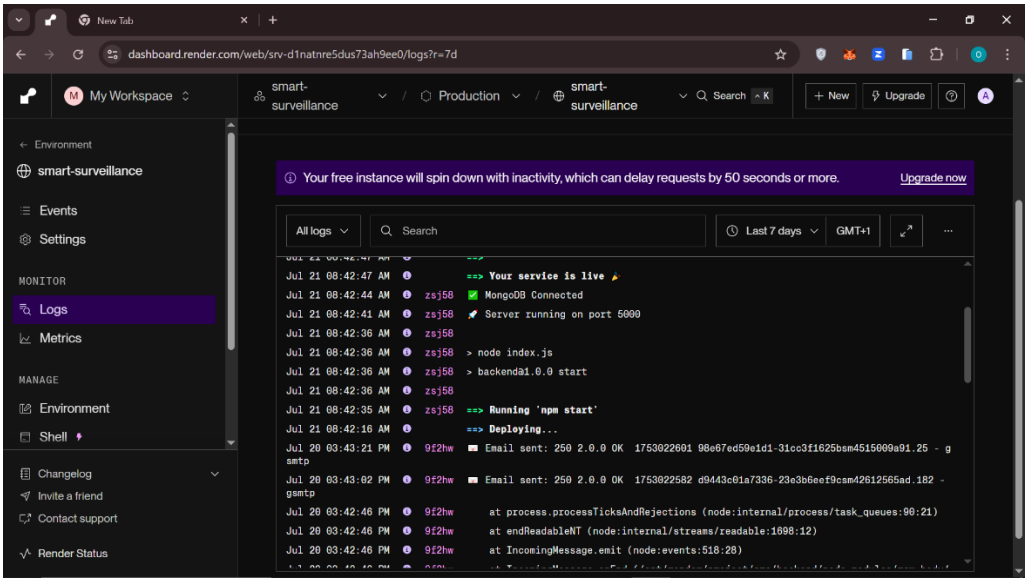

Figure 4.6 Backend Server Dashboard

The Web application User Interface (or Frontend) is built using React.js and Tailwind CSS for structure and styling. React.js was used due to its exceptional development experience and functional state management capabilities that handle user interactions, data fetching and errors gracefully. It consists of authentication pages, a user dashboard and a settings page for modifying certain system configuration parameters. The dashboard is where the user can access the past event logs and their images. It is equipped with search functionalities which are further enhanced with time-based filters that aid the user in extracting the specific data they need from the array of event logs. The site is fully responsive and functions adequately on a variety of devices.

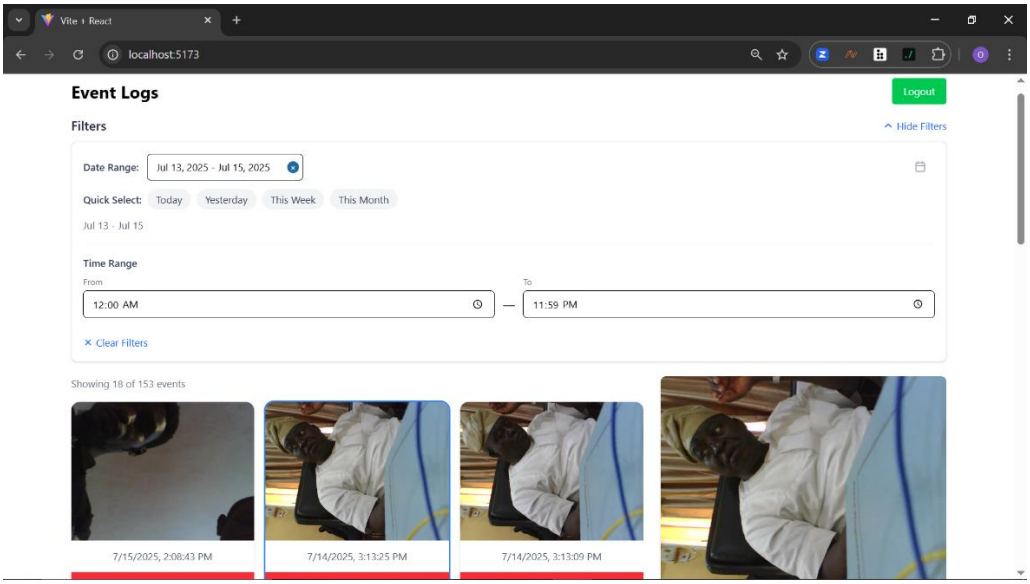

Figure 4.7 Web Application User Interface

All the code for the frontend and backend were uploaded to a GitHub repository for secure storage and version control. The application itself was deployed and hosted on Render, a cloud hosting platform for web services.

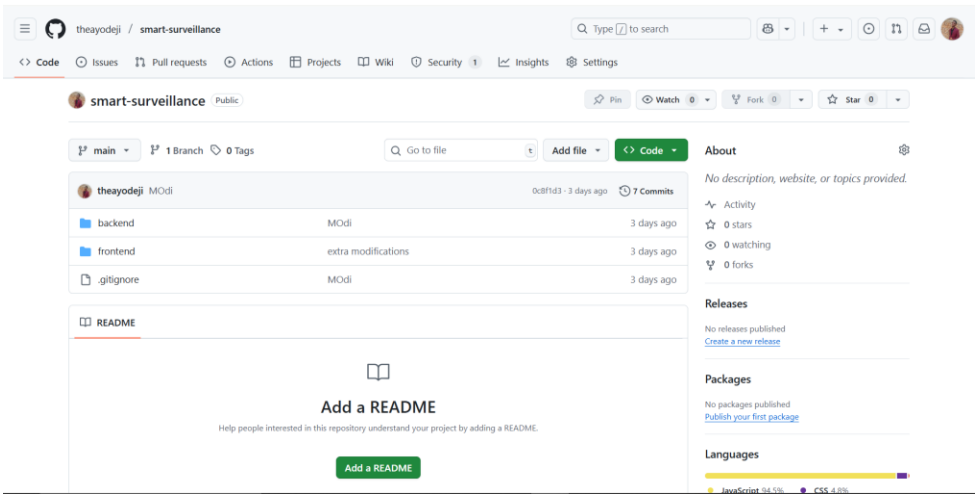

Figure 4.8 Surveillance System GitHub Repository

CHAPTER 5

CONCLUSION AND RECOMMENDATIONS

5.1 Conclusion

The effectiveness of surveillance systems has been greatly increased due to the advances in microcontroller technology, API-based communication and sensor technology. The ESP32-CAM has been successfully utilized in this project to capture relevant activity when motion is detected by a PIR sensor within a specific (or calibrated) range. A REST API was also developed to facilitate communication between the components over the internet and provide room for scalability for multi-camera system. Cloud storage was employed to minimize the memory load on the ESP32-CAM and backend server, and the additional cost of said memory. Storing the images in the cloud also made provision for remote access to these images and protects them from loss due to device damage.

Automated alerts instantly notify the user of activity and potential threats via email with an image to provide context and remote monitoring. Past event logs, their date and time, as well as the associated image were made accessible to the user through the web application from anywhere in the world. It allowed users to search for specific events by date and other time-based filters.

5.2 Limitations

Despite achieving its primary objectives, the system has several limitations. First, the reliance on motion detection can lead to false positives in busy environments, such as areas with pets or frequent human activity. Second, internet dependence may pose a problem when the system is used in a location where network connectivity is poor. Additionally, the system's current implementation lacks encryption, exposing it to potential security vulnerabilities. Scalability remains a concern, as the system was optimized for small-scale deployment with a single camera unit per user and may require significant adjustments to handle larger surveillance networks for larger applications.

Finally, the capture of images rather than video clips may be an issue where event context is required. For instance, a criminal investigation may require proof of an action being performed and the occurrences leading up to that particular event, which single images or even an array of images may not provide.

5.3 Recommendations

To enhance the system's overall effectiveness, several improvements are suggested. First, replacing still images with short video clips can provide more context during motion-triggered events, improving clarity and decision-making. Addition of AI features for situation analysis to distinguish real threats and minimize false alarms should be considered. Additionally, incorporating encryption will enhance data

privacy and system security. To support deployment in areas with poor connectivity, local storage buffering and temporary SD card storage should be considered.

References

Abordo, P. R. C., Pastores, A. G. J., Villaroza, J. J. C., Guerrero, I. F. S., Robles, J. K. A. Q., Mangubat, J. D. D., Borleo, K. G. M., Bautista, J. N., & Galay-Limos, J. A. (2024). *Smart surveillance system using ESP32*. ResearchGate. https://www.researchgate.net/publication/382266789_Smart_surveillance_system_using_ESP

Obodoeze, F. C., Okoye, F. A., & Obiokafor, N. I. (2018). Holistic security for Internet-of-Things (IoTs) implementation in Nigeria. *International Journal of Trend in Scientific Research and Development (IJTSRD)*, 2(4), 804–813. https://doi.org/10.31142/ijtsrd9457

Ibok, I., Udom, E., & Sunday, A. (2024). Smart surveillance systems for enhancing urban security: An IoT-based approach using Arduino and Firebase. *FNAS Journal of Basic and Environmental Research*, *5*(1), 45–52. https://fnasjournals.com/index.php/FNAS-JBER/article/view/720

Jain, A., Pandey, S., & Shrivastava, S. (2020). Smart surveillance and home automation using IoT. *International Journal of Creative Research Thoughts (IJCRT)*, *8*(5), 1294–1300. https://ouci.dntb.gov.ua/en/works/4zPGXVJ9/

Kennedy, C. G., Okokpujie, K. O., Okokpujie, I. P., Young, F. T., Akingunsoye, A. V., & Asuna, A. R. (2023). Development of an affordable real-time IoT-based surveillance system using ESP32 and Twilio API. *International*

Journal of Safety and Security Engineering, 12(7), 123–129. https://doi.org/10.18280/ijsse.120701

Li, R. Z. (2018). Replicas strategy and cache optimization of video surveillance systems based on cloud storage. *Future Internet, 10*(4), 34. https://www.mdpi.com/1999-5903/10/4/34

National Institute of Standards and Technology. (n.d.). *Digital video surveillance technology*. https://www.nist.gov/publications/digital-video-surveillance-technology

Nikouei, S. C. (2019). I SAFE: Instant Suspicious Activity identiFication at the Edge using fuzzy decision making. *arXiv*. https://arxiv.org/abs/1909.05776

OPSWAT. (2024, June 21). *Data storage security: Challenges, risks, and best practices*. https://www.opswat.com/blog/data-storage-security

Patil, A., Pati, S., Momin, L., Patil, Y., & Nalawade, S. (2023). Surveillance system using ESP32-CAM with motion detection and cloud storage. *International Research Journal of Modernization in Engineering Technology and Science, 5*(7), 1–6. https://doi.org/10.56726/IRJMETS47001

Temitope, O. (2024). *Development of a web-based integrated surveillance system* (Undergraduate thesis). University of Ilorin.

Zeng, L., Xu, L., Shi, Z., Wang, M., & Wu, W. (2006). Techniques, process, and enterprise solutions of business intelligence. *Proceedings of the 2006 IEEE*

International Conference on Systems, Man and Cybernetics, 6, 4722–4726. https://doi.org/10.1109/ICSMC.2006.384918

Tariku, M. (2022). *Design and implementation of smart home security system using IoT*. [Undergraduate thesis, Jimma University]. Jimma University Repository. https://repository.ju.edu.et/handle/123456789/7012

Erken, B. A. (n.d.). *Visualizing and monitoring of environmental data using various sensors*. International Journal of Scientific & Engineering Research, 10(5), 1036–1040https://www.ijser.org/researchpaper/Visualizing-and-Monitoring-of-Environmental-Data-Using-Various-Sensors.pdf

SMXInfo Whitepaper (Matched with Source #34)

Real Time Logic. (n.d.). Barracuda Application Server for Device Control Applications. SMXInfo. http://www.smxinfo.com/rtos/tcpip/barracuda/wp/DeviceControl.pdf

APPENDICES

APPENDIX A: Bill of Engineering and Materials Evaluation (BEME)

S/N	Component	Quantity	Unit cost (₦)	Total cost (₦)
1	ESP32-CAM Module	1	15,000	15,000
2	OV5640 Camera Module	1	5,600	5,600
2	PIR Sensor.	1	2,500	2,500
3	3.7V Li-ion Battery	1	3000	3,000
4	MT3608 Boost Converter	2	3,200	6,400
5	TP4056 Charging Module with BMS	2	3,000	6,000
6	KCD-101 Rocker Switch	1	500	500
7	Vero Board	1	500	500
8	2-Terminal PCB Flex Connectors	3	200	600
9	3D-Printed Enclosure	-	-	42,000
10	Jumper Wires	-	-	1,000
11	Soldering Lead (100g)	-	-	4,000
12	Copper Wires	-	-	2,000
	TOTAL			89,100

APPENDIX B: ESP32-CAM Firmware code

```
#include "esp_camera.h"
#include <WiFi.h>
#include <HTTPClient.h>
#include "base64.h"

#define PIR_PIN 13  // PIR sensor pin

#define CAMERA_MODEL_AI_THINKER
#include "camera_pins.h"

// WiFi credentials
const char *ssid = "******";
const char *password = "******";
// Server URL
const char *serverUrl = "https://smart-surveillance-
etdk.onrender.com/api/events/upload";

// Connect to Wi-Fi with timeout
bool connectToWiFi(const char *ssid, const char *password, unsigned long
timeoutMs = 15000) {
  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi");
  unsigned long startAttemptTime = millis();
```

```
  while (WiFi.status() != WL_CONNECTED && millis() - startAttemptTime <
timeoutMs) {
    Serial.print(".");
    delay(500);
  }
  Serial.println();
  return WiFi.status() == WL_CONNECTED;
}
// Capture image and return Base64 string
String captureAndEncodeImage() {
  camera_fb_t *fb = esp_camera_fb_get();
  if (!fb) {
    Serial.println("Camera capture failed");
    return "";
  }

  String base64Image = base64::encode(fb->buf, fb->len);
  base64Image.replace("\\", "\\\\");
  base64Image.replace("\"", "\\\"");
  base64Image.replace("\r", "");
  base64Image.replace("\n", "");

  Serial.printf("Base64 length: %d\n", base64Image.length());
  esp_camera_fb_return(fb);
  return base64Image;
```

```
}

// Upload image to server
void uploadImage(const String &base64Image) {
  HTTPClient http;
  http.setTimeout(10000);
  http.begin(serverUrl);
  http.addHeader("Content-Type", "application/json");

  String jsonBody = "{\"image\":\"" + base64Image + "\"}";
  int httpResponseCode = http.POST(jsonBody);

  if (httpResponseCode > 0) {
    Serial.printf("HTTP Response code: %d\n", httpResponseCode);
    Serial.println(http.getString());
  } else {
    Serial.printf("HTTP POST failed: %s\n",
http.errorToString(httpResponseCode).c_str());
  }

  http.end();
}

// Setup camera
bool setupCamera() {
```

```
camera_config_t config;
config.ledc_channel = LEDC_CHANNEL_0;
config.ledc_timer = LEDC_TIMER_0;
config.pin_d0 = Y2_GPIO_NUM;
config.pin_d1 = Y3_GPIO_NUM;
config.pin_d2 = Y4_GPIO_NUM;
config.pin_d3 = Y5_GPIO_NUM;
config.pin_d4 = Y6_GPIO_NUM;
config.pin_d5 = Y7_GPIO_NUM;
config.pin_d6 = Y8_GPIO_NUM;
config.pin_d7 = Y9_GPIO_NUM;
config.pin_xclk = XCLK_GPIO_NUM;
config.pin_pclk = PCLK_GPIO_NUM;
config.pin_vsync = VSYNC_GPIO_NUM;
config.pin_href = HREF_GPIO_NUM;
config.pin_sccb_sda = SIOD_GPIO_NUM;
config.pin_sccb_scl = SIOC_GPIO_NUM;
config.pin_pwdn = PWDN_GPIO_NUM;
config.pin_reset = RESET_GPIO_NUM;
config.xclk_freq_hz = 20000000;
config.frame_size = FRAMESIZE_SXGA;
config.pixel_format = PIXFORMAT_JPEG;
config.fb_location = CAMERA_FB_IN_PSRAM;
config.jpeg_quality = 12;
config.fb_count = psramFound() ? 2 : 1;
```

```
  config.grab_mode = psramFound() ? CAMERA_GRAB_LATEST :
CAMERA_GRAB_WHEN_EMPTY;

  esp_err_t err = esp_camera_init(&config);

  sensor_t *s = esp_camera_sensor_get();
  s->set_vflip(s, 1);    // Flip image vertically
  s->set_hmirror(s, 1);  // Optional: flip horizontally if needed

  if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x\n", err);
    return false;
  }

  if (s->id.PID == OV3660_PID) {
    s->set_vflip(s, 1);
    s->set_brightness(s, 1);
    s->set_saturation(s, -2);
  }

  return true;
}

void setup() {
  Serial.begin(115200);
```

```
  Serial.println("\nBooting...");

  pinMode(PIR_PIN, INPUT);  // PIR sensor input

  if (!setupCamera()) {
    Serial.println("Camera setup failed");
    return;
  }

  if (!connectToWiFi(ssid, password)) {
    Serial.println("Failed to connect to WiFi");
    return;
  }

  Serial.println("WiFi connected");
}

void loop() {
  if (digitalRead(PIR_PIN) == HIGH) {
    Serial.println("Motion detected!");
    String base64Image = captureAndEncodeImage();
    if (base64Image.length() > 0) {
      uploadImage(base64Image);
    } else {
      Serial.println("Image capture failed");
```

```
    }
    delay(5000);  // Prevent spamming on constant motion
  }

  delay(200);  // PIR debounce delay
}
```

APPENDIX C: Repository Links

Server API Configuration: https://github.com/theayodeji/smart-surveillance/tree/main/backend

Web Application Interface: https://github.com/theayodeji/smart-surveillance/tree/main/frontend