Drone Surveillance System

Project Report

Group 3

Aalborg University
Electronics and IT

Electronics and IT
Aalborg University
http://www.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Drone Surveillance System

Theme:
Drones, IoT and cloud

Project Period:
Fall Semester 2025

Project Group:
3

Participant(s):
Casper Bramm
Abdullah Al Mehedi Hira
Vasco Teixeira Afonso
Sebastian Roberto Ovelar Anderson

Supervisor(s):
Sokol Kosta

Copies: 1

Page Numbers: 62

Date of Completion:
December 22, 2025

Abstract:

This project focuses on the development and implementation of a surveillance system utilizing drones and cameras. The main challenges addressed and worked on includes remote configuration of camera video streams and handling as well as remote control of a drone from a cloud interface. The final product is a containerized framework for handling drone and camera surveillance with integrated object detection model for automated alerts. The system aims to be flexible, allowing users to utilize commercially available hardware. Additionally the system supports end-to-end encryption and role-based access control to ensure secure communication and restrict unauthorized access to sensitive operations. Finally the project was concluded with some reflections regarding the process and final system.

Contents

Chapter 1

Introduction

In recent years, the use of drones and camera-based surveillance systems has increased significantly in both public and private sectors. These systems are commonly used for tasks such as area monitoring, infrastructure inspection, security patrols, and emergency response. Compared to traditional fixed surveillance solutions, drones have high flexibility, can cover wider areas, and can be deployed fast, making them suitable for dynamic and large-scale environments.

1.1 Motivation

Traditional surveillance systems are often based on fixed cameras and require significant infrastructure, such as permanent installations, wired networks, and dedicated control centers. These limitations reduce their usability in temporary, remote, or fast changing environments. Drones and mobile cameras address some of these challenges by offering mobility and rapid deployment, but they introduce new technical challenges related to connectivity, system integration, and security.

Moreover, an open source plug-and-play drone surveillance system will be an innovative solution given the rise in availability of different drone brands and the advances in powerful computer vision models that can be used for real-time object detection. Integrating such models into a surveillance system allows automatic detection of predefined threats and reduces the need for continuous human monitoring.

1.2 Objectives

This project aims to explore how a modular and independent surveillance framework can be designed using commercially available drones and cameras. By combining real-time video streaming, centralized management, and AI-based detection, the system seeks to provide a practical and flexible solution for monitoring areas of interest. Rather than focusing on a single device or vendor, the project emphasizes interoperability, scalability, and security.

The outcome of this work is not a fully commercial product, but a functional prototype that demonstrates how such a system can be designed, implemented, and evaluated. The project also serves as an opportunity to apply concepts from computer systems engineering, security in IoT devices and the correct and secure use of cloud technologies.

Chapter 2

Problem Statement

In the Introduction, we discussed the potential upsides of having a surveillance system using drones that could serve the need of individual users, with flexibility for expansion, and that makes use of the latest technologies for video analysis. We now present the problems we aim to solve based on this.

2.1 Main Problem statement

> ***How to develop a system that provides a portable and independent framework for surveying areas using commercial drones and cameras?***

To further understand the problem statement, we formulated these sub problem statements to lead us in the right direction for solving the main goal of this project.

2.1.1 Subproblem statement 1

> ***How can we develop a framework for video streaming of devices to a single cloud backend that is able to expose these streams to a user?***

A key element of any surveillance system includes handling multiple incoming video streams from different cameras at the same time. The subproblem will focus on how to make this possible in our setup.

2.1.2 Subproblem statement 2

> ***How can we develop a system that allows for remote connection, video retrieval and control of a drone?***

A drone is not quite the same as a camera, and has some other configurations as well as flight mechanisms. We wanted to include a drone in our system and investigate how to make the drone send video and receive flight commands for remote control.

2.1.3 Subproblem statement 3

How can we set up autonomous threat detection in the system that alerts the user when a threat is detected?

As image recognition and AI are hot topics for automation in industries to eliminate manual labor, including an automatic alert system based on image recognition models seems like a natural choice. We will integrate an AI model for video detection, but we will not implement or train our own.

We will later in the report refer to what is being detected as a "threat". This "threat" is an arbitrary entity as it is not an important aspect of the project what is detected, but rather that when the object detection model detects something we define as a threat, we are able to catch it and forward the alert triggered.

2.1.4 Subproblem statement 4

How can we securely manage these connection and manage access control in the system to minimize risk of exploitation

Security is a core part of any reliable system as cyber crime is a constant global threat to all electronic systems. To prevent our system to be prone to these attacks and to learn what measurements are implemented we added this subproblem as our last subproblem.

Chapter 3

Methodology

In this section, we will describe all of the methods and strategies that have been used during the project, along with the reasons why we chose those. The objective is to provide an insight into our project research phase and how we achieve our target results.

We integrated the Agile Model, the iterative decision-making process, into our workflow, thus allowing us to recheck and refine any of our previous decisions. This approach enhanced the productivity and the quality of our decisions and allowed us to make continuous improvement until the final output.

We had weekly meetings to discuss the project and assign tasks to each group member. We used Jira to make the process of tracking tasks and assign them easier. These meetings gave a direction for each element to follow, increasing productivity.

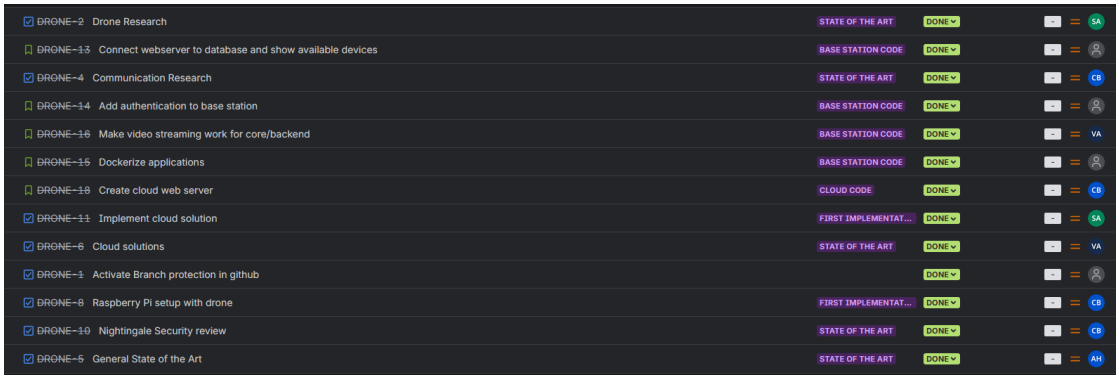

Figure 3.1: Jira Board

In addition, following the V-model process, we ensured a smooth connection between each development phase and its corresponding testing activity. We started the project with the identification of user and system requirements, from high-level to low-level designs. Each of

the development stages has progressed with verification and validation activity which helped us to have early test planning and quality assurance. Implementation started once the designs and other related parts were done. By following this method, the project maintained traceability, minimum defects, and ensured validation was done at each stage of progression which ensured a reliable and well-tested system.

In the state of the art, we have investigated relevant projects that have similar content to ours, and the main purpose of this is to gather relevant knowledge and inspiration from their content. And used those ideas and made scenarios in the requirements. In this section, all the researched information is based on up-to-date and detailed sources which were provided by the companies themselves on their websites and other reliable online platforms.

We derived scenarios based on the state-of-the-art, which portrays the base for the system description. Both functional and non-functional requirements are under the section final requirements. There, they are described individually, showing their origin and to which part of the system they apply.

In the system design, our system is further explained using system architecture diagrams which will provide a good overview of the system flow in this project and will open doors for the implementation.

Chapter 4

State of the Art

In this chapter, we review the current state of the art in drone surveillance systems and other relevant technologies to search for answer of our problem statement. The goal of this analysis is to understand how existing research and commercial solutions address the challenges of building a secure and scalable drone surveillance system.

The chapter is divided into three parts. First, we analyze recent research literature on drones in security and surveillance, focusing on system architectures and concrete security methods. Second, we review and compare existing commercial drone surveillance solutions to extract practical design insights. Finally, we explore some other technologies that are relevant to the system and hadn't been identified in the other sources.

4.1 Drones in Security and Surveillance

The use of drones for security and surveillance has increased significantly in recent years. Modern drone surveillance systems are not only limited to simple video capture, they also integrate real-time communication, automated threat detection, and distributed decision-making. As a result, the system design should be robust, ensuring a secure and reliable system.

The work of AL-Dosari et al. studies the use of drones in surveillance systems and proposes a prototype aimed at standardizing how such systems can be designed and deployed. They have tested their prototype with the Qatar Police using on-call drones to patrol specific areas. They describe two different prototypes. The first is a system where images captured by the drone are monitored by human operators. The second is a more advanced system that allows remote access and supports automated threat recognition, creating a hybrid solution that combines human supervision with automated analysis. This work showcases how drone systems can be designed, implemented and tested, which is great starting point for our development.[8]

Security is another key challenge in drone surveillance systems, especially when multiple drones and ground stations need to communicate with each other. Gilani et al. focus on this problem by proposing a secure communication architecture for an Internet of Drones (IoD) sys-

tem. Their approach uses the IOTA distributed ledger to improve trust and security between drones and the ground station. The proposed system includes cryptographic techniques such as elliptic-curve signatures (ECDSA), hashing (SHA-256), and secure session establishment to authenticate drones and ground stations before exchanging surveillance data. Using these methods, they achieved protection against common attacks such as spoofing, replay attacks, and botnets.[16]

The reviewed literature showcase the following points:

1. The need for a well designed system implementation.
2. The necessity of strong authentication and secure communication.
3. The suitability of modern cryptographic methods, such as public-key authentication and lightweight key exchange, for drone systems with limited computational resources.

These insights justify the focus of this project on design and secure communication mechanisms, including authenticated gRPC channels using Ed25519 keys. By grounding our design in established research, we ensure that the proposed system follows best practices in both functionality and security.

4.2 Drone security systems

On the marked already several Drone surveillance systems are commercially available. We choose to investigate some of these solutions to gain inspiration for the design of our solution.

These solutions include:

- Nightingale security
- Asylon Robotics
- Height Technologies
- Securiton AG
- Nando Drone

4.2.1 Asylon Robotics

Asylon is a leading U.S. provider of automated air and ground robotic perimeter security systems, combining advanced robotics with real-time human oversight. Its DroneCore platform integrates drones, robotic dogs, intelligent sensors, and AI-driven software to deliver autonomous patrols, rapid response, and robust situational awareness [7] [4].

At the center of the system are the Robotic Security Operations Center (RSOC) and DroneIQ software, which serve as the backend of the operations. They provide 24/7 monitoring, mission assurance, remote command from FAA-certified operators, and AI-powered automation

[7] [4]. “FAA-certified” meaning: approved by the Federal Aviation Administration, the U.S. government agency responsible for regulating aviation.

DroneIQ integrates with alarms, access control, sensors, and existing GSOC (Global Security Operations Center) infrastructure. This enables streamlined workflows and comprehensive data analytics. With over 85,000 past executed security missions and a proven dual-use model, Asylon delivers advanced robotic security solutions for both military and commercial environments [4].

This short blog [6] shows a demonstration of the overall system architecture of an Asylon robotics solution deployed system. The figure is as shown below 4.1.

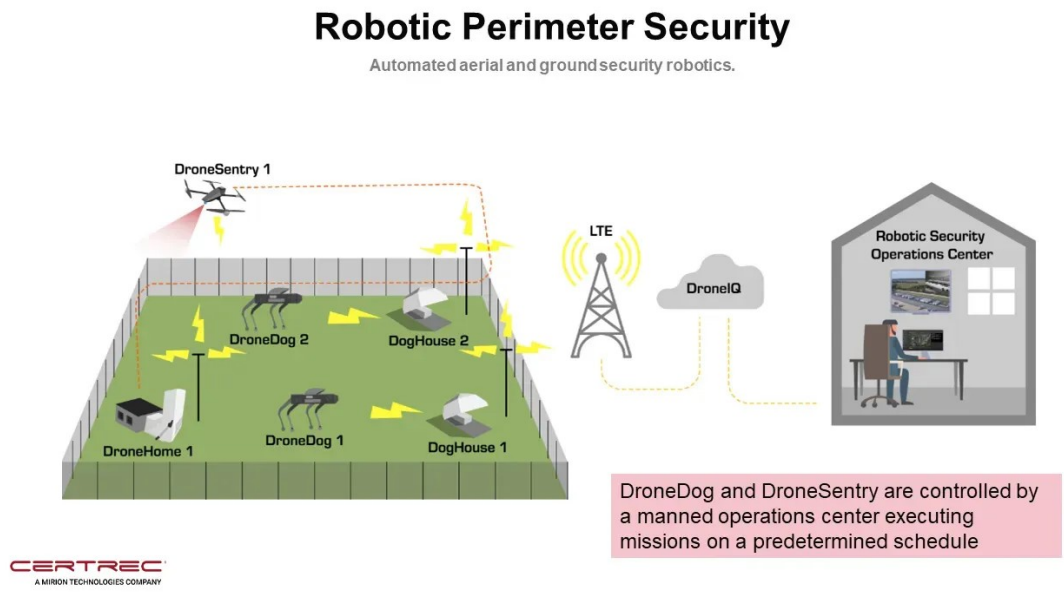

Figure 4.1: Asylon deployment example

The picture shows that the general system has a LTE antenna communicating to each drone. The drones "home" in this system’s only function is to charge the drone, as the drone communicate directly to the cloud through LTE. For this to be possible, the drone needs an antenna that can communicate using LTE technology. The Guardian model is the flying propellor drone seen in the picture. Even though a charging station is deployed nearby the drone coverage area the drone receives its instruction directly through the LTE communication, and uses AES-256 for encryption [5].

The LTE transport of data is supplied by a cloud backend running their DroneIQ service. This service enables the controller of the drone to send instructions or execute scheduled missions to the drone [7] [4] [5]. From this picture of the overview we can tell that the general system architecture that the system has 4 main modules.

- Drones deployed on site
- Drone home for recharging drones
- Backend service for monitoring and managing drone missions
- User interface for using the cloud backend services

4.2.2 Nightingale security

Nightingale Security is an American tech company that offers real-time drone surveillance solutions that support fast emergency alerts to local authorities if intrusion is detected [11].

This service is a deployable system that companies can buy as a service. Nightingale Security can then deploy their system at the desired location of the customer. Each deployment requires a edge/sink IoT device that comes with powerful components [12].

These include:

- CPU: Intel i7 10700TE [12]
- RAM: 16GB DDR4 [12]
- GPU: NVIDIA RTX 3060 [12]
- Storage: 512GB SSD, 10TB HD, 4TB HD (expandable) [12]

Additional system features include: Robotic AI Intrusion Detection (RAID): Drones automatically patrol facility perimeters and send alerts only when people or vehicles are detected, reducing false alarms and improving efficiency [12].

Scheduled Autonomous Patrols: Plan patrols by day, time, route, altitude, hover time, and camera angle to obtain consistent and repeatable coverage [12].

Autonomous Threat Response: When an alarm goes off, a drone launches automatically, flies to the area, and sends live video to the security team [12].

Manual Surveillance Missions that can launch a drone quickly for live monitoring during an incident [12].

Always On Duty: With drones, base stations, and mission software running around the clock, the system is always ready for use [12].

System architecture

Nightingale has 3 models for their deployment architectures. These are called:

- Flexible model [13]
- Security model [13]

- Restrictive model [13]

Each model has its own use case for deployment, depending on the need for accessibility versus security. The similarities between the deployment models reveal the core aspect of the operational features of the system.

The core aspects are the following.

- Drone [13]
- Base station [13]
- Drone control server [13]

The architectures follows a layered communication structure. The Drone acts as the operational endpoint responsible for flight execution and data capture. The Base Station functions as an intermediary node, managing mission data, telemetry, and control signals. The Drone Control Server (DCS) serves as the central coordination layer, handling authentication, mission planning, and data management. Together, these components form a distributed system designed for both resilience and adaptability [13].

4.3 Comparison of Commercial Drone Surveillance Solutions

To compare the solutions we have summarized the key features of representative drone-based surveillance systems (Asylon Guardian, Height Technologies SAMS, Securiton SecuriDrone, Nando Autonomous Drone, and Nightingale Security) in the table below.

Most of these systems, such as those from Asylon, Height Technologies[1], Nando[2], and Nightingale Security, include a fixed docking or base station that allows the drone to take off, land, and recharge automatically. In contrast, Securiton's system[3] is vehicle-mounted, which means it can be moved easily but still needs an operator to control the drone and handle charging.

All of the compared systems provide real-time video and some form of remote access. Most also use secure communication methods like AES encryption to protect data transmission. The main difference between them lies in their level of autonomy, flight endurance, and the type of sensors used. For example, high-end systems can stay in the air for up to 70 minutes and use both normal and thermal cameras, while others focus on shorter automated patrols.

Feature	Asylon Guardian	Height SAMS	Securiton SecuriDrone	Nando Drone	Nightingale Security
Real-time video	Yes (day/night camera)	Yes (HD and thermal)	No (signal detection only)	Yes (HD + thermal)	Yes (4K + thermal)
Remote access	Cloud streaming and 24/7 monitoring	Remote operator monitoring	Web or app interface	Web/cloud interface	PC and mobile mission manager
Security (encryption)	AES-256	Not specified	RF detection and defense	Not specified	AES-128
Plug-and-play	Docking base station	Docking base station	Vehicle-mounted kit	Automatic charging dock	Fixed docking station
Autonomy	Automated patrols	Fully autonomous patrols	Autonomous detection system	Fully autonomous patrols	Fully autonomous patrols
Flight time	Around 30 min	Extended	N/A	Up to 70 min	30–50 min
Sensors	HD day/night camera	HD and thermal camera	RF sensors	HD and thermal camera	4K and thermal camera
Deployment	Fixed	Fixed or mobile dock	Vehicle-mounted	Fixed dock	Fixed
Base station	Yes	Yes	No	Yes(charging dock)	Yes

Table 4.1: Comparison of key features among commercial drone surveillance systems.

4.4 Onboarding of Internet of Things Devices

With the growth of the Internet of Things and the desire to connect more and more of these devices to the cloud, there is a need to adopt simple and safe onboarding mechanisms. Most of the mechanisms that currently exist rely on a three-box model composed of the Enrollee, the Configurator and the Access Point. [9]. In this model the configurator is assumed to have already access to the network and its main goal is to help the enrollee to obtain access to the network provided by the access point. [9] explains this process in an algorithm:

1. **Enrollee** powers on and sends configuration.
2. **Configurator** checks the **Enrollee** info and sends the **Access Point** Information
3. **Enrollee** Uses the information received to connect to the **Access Point**
4. **Access Point** allows connection of the **Enrollee** based on the information received

There are multiple implementations of this algorithm varying in how the Configurator and the Enrollee exchange information.

NFC Based

The Enrollee needs to have an NFC chip embedded in which the Configurator should tap into to read the information.

QR Code Based

The Qr Code approach is similar to the NFC one but instead of an NFC chip there should be a qrcode which the Configurator could read.

Ultrasound Based

Similar to the other two, but using ultrasound as the medium of transmission.

Soft-Ap based

In Soft-AP Based schemes the Enrollee typically creates a new temporary access point to which the Configurator connects in order to exchange the credentials of the final Access point. This method does not require special hardware (besides access point capabilities), in contrast with NFC and Ultrasound. However in order to establish connection with the Enrollee, the Configurator must tear-down its connection, resulting in an offline period.

4.5 Remote Control systems

The available drone surveillance system descriptions do not clearly specify how the data exchange occurs between the cloud and the fog nodes. Therefore we decided to investigate other types of solutions that have as one of the primary focus low latency streaming.

4.5.1 Cloud Gaming

The challenge of remote controlling a UAV over the network has analogous challenges to the ones present in remote desktop systems in the sense that both receive commands as inputs and output a video stream. One use case of this technology that closely relates to the remote control of drones is cloud gaming since latency between user instructions and video feedback greatly impact the Quality of Experience. The premise of cloud gaming is that of allowing users to experience the latest titles without the need to constantly upgrade their hardware setup [14]. In [14] is presented a framework for a cloud gaming platform, as seen in Figure 4.2

We will now analyze the implementation of one of the most used cloud gaming platforms.

GeForce Now

GeForce Now, developed by NVIDIA, is a cloud gaming service that allows users to play games remotely in powerful machines containing Nvidia GPUs with the goal of having a smooth gaming experience. It is important to understand in this context how data is exchanged between the client and the cloud to achieve the lowest possible latency. An analysis of the protocols present in this service can be found in [10] which identifies WebRTC as the primary technology for browser based connections. WebRTC is particularly significant because it

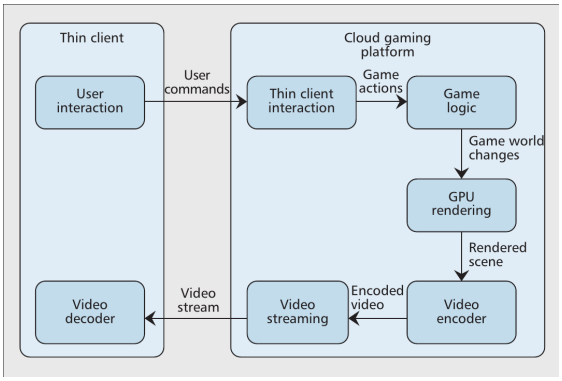

Figure 4.2: Cloud platform framework [14].

enables gameplay media and user input to be multiplexed over a single UDP stream, in contrast to the multiple parallel flows typically used by native or console based clients, therefore simplifying communication.

4.2.

4.6 Summary of state of the art

This chapter has examined the current state of drone surveillance systems through academic research, commercial solutions, and related technologies to establish a foundation for our system design.

The academic literature highlighted three key requirements: well-designed system architecture, strong authentication and secure communication protocols, and modern cryptographic methods suitable for resource-constrained drones. These findings validate our focus on secure communication mechanisms and authenticated channels.

The commercial solutions analysis revealed common architectural patterns across industry systems. Most employ a modular structure consisting of autonomous drones, base stations/drone hubs, backend services, and user interfaces. The comparison showed varying deployment approaches—from isolated restrictive models to flexible cloud-integrated architectures—demonstrating that design must adapt to specific security and operational requirements. Common features include automated patrols, real-time video streaming with thermal capabilities, and encrypted communication using AES.

The exploration of IoT onboarding mechanisms provided practical approaches for securely integrating drones into network infrastructure. The three-box model offers multiple implementation strategies varying in complexity and security. Additionally, cloud gaming technologies like GeForce NOW demonstrate proven techniques for low-latency video streaming and edge computing, directly applicable to real-time drone control and video transmission.

Chapter 5

Requirements

In this section, we will start outlining the requirements for the system. This will be done by exploring scenarios based on the research done on the state of the art and with the usage of use case diagrams. We will in the end of this section, outline the final requirements for this system.

5.1 Limitations

As we are students and have access to limited resources, we have set these limitations for our system. These limitations will play a big role in the system design and the requirements for the system.

Our main limitation for this project is the DJI Tello Drone, we are working with. This drone is small, commercially available, and affordable, but comes with severe limitations. These limitations include:

- 1. Drone is not on-board programmable and can only be connected to using its own WiFi hot spot.
- 2. Drone does not support remote activation.
- 3. Drone has to have battery manually changed and can't recharge itself.

Figure 5.1: Dji Tello Drone

These limitations means that we need to have the local access point near the drone connected to the drone's own WiFi hotspot, and have that access point act as a controller of the drone on site.

We will for the rest of the report refer to this local access point as the **Base Station**. This terminology is adopted from the Nightingale Security state-of-the-art study, where the drone-hosting edge device is referred to as a Base Station (ref: 4.2.2).

This heavily affects the architecture of the system so that Drones are completely reliant on having a Base Station as a remote control unit.

5.2 Scenarios study

In the state of the art we researched some other systems using drones and cameras for area surveillance. From our study of those we generated these scenarios and from those derived requirements for our own system.

Scenario 1:
When the camera detects any threats, the user will be alerted. The user can then launch the drone to monitor the area further and optionally track the threat using the drone control interface (ref: 4.2.2).

Takeaway from Scenario 1:

1. The stationary cameras should continuously monitor the area.
2. The system should be able to alert the user if a threat is detected.
3. The user should be able to launch the drone at will.
4. The user should be able to manually control the drone using a low-latency video stream that supports responsive user interaction.
5. The system should be able to detect a "threat" autonomously.

6. The system should support drone steaming with a delay of 200 ms to user interface to support reliable latency for drone control.
7. The stationary cameras should stream video when no threats are detected.
8. The system should provide a drone control interface that supports optional threat tracking.

Scenario 2:
The user wants to monitor the area using the drone. The user can then manually activate the drone and control it to monitor the area, even though he did not receive an alert by the system (ref: 4.2.1, 4.2.2).

Takeaway from Scenario 2:

1. The user should be able to use the drone for monitoring even when no threat is detected.

Scenario 3:
The user wants to review any previous clip that was captured through the drone or the camera for a given moment in time. (Standard scenario for surveillance systems)

Takeaway from Scenario 3:

1. The system should store recordings from both the drone and stationary cameras.
2. The user should be able to access the web interface to replay previous clips.
3. The user should with ease be able to navigate to specific timestamps where alerts were triggered.
4. The system should log alert timestamps for later review.
5. The system should provide a searchable list of past alert events.
6. The system should maintain a time-indexed database of recordings.

Scenario 4:
A new customer wants to apply the system at a desired location. The Base Station must be manually deployed and configured with the cloud (ref: 4.4, 4.2.2).

Takeaway from Scenario 4:

1. The system should support manual deployment and configuration of a base station at any location with power and Wi-Fi.
2. The base station should be able to connect to the cloud for registration and synchronization.
3. The cloud should associate each base station with the correct user interface instance.
4. The base station should support connection to a local network for communication.
5. The system should verify successful cloud linkage before enabling user access.
6. The system should allow configuration of the base station through an on boarding interface.
7. The system should verify network connectivity during setup.
8. The cloud must uniquely identify each base station.
9. Communication between cloud and Base Station should be Authenticated.

Scenario 5:

A new customer wants to deploy a new drone or camera to the base station. The user should have an interface to manage devices connected to the base station (ref: 4.4).

Takeaway from Scenario 5:

1. The user should be able to add and remove devices.
2. The user should be able to select from available protocols when registering a new device.
3. The system should automatically detect compatible devices during onboarding.
4. The user should be able to manage device configurations through an interface.
5. The system should validate compatibility of newly added devices.
6. The system should maintain an updated inventory of connected devices.

5.3 Use case diagram

A use case provides an overview of the interactions the different actors have with the system. Through investigations of the scenarios we made this use case diagram 5.2.

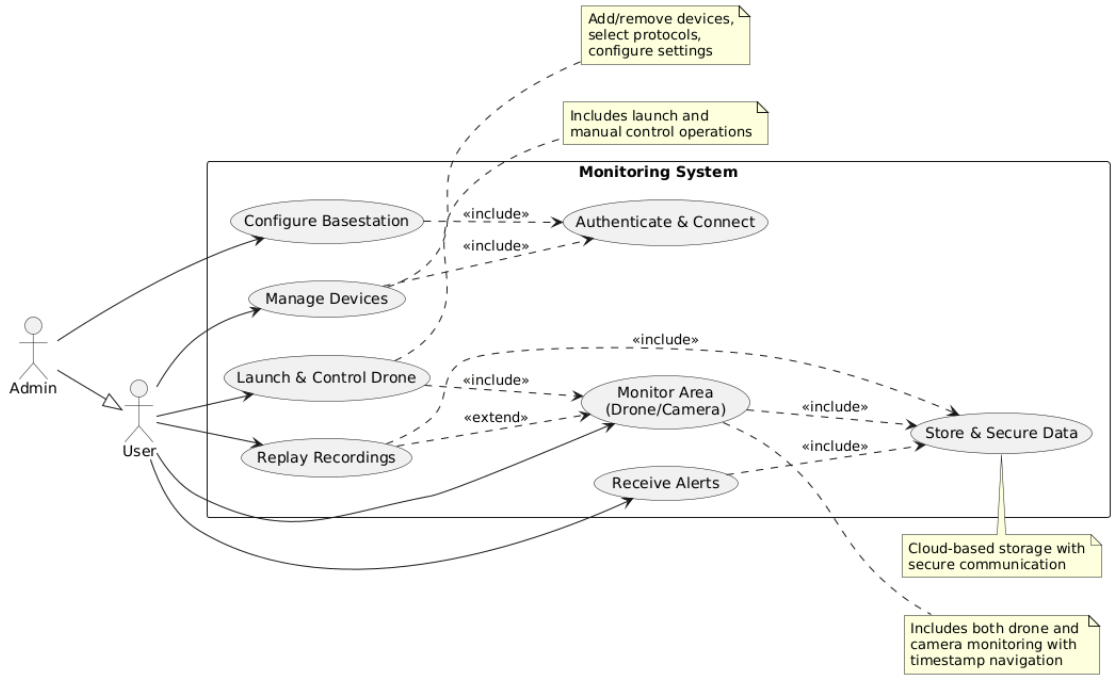

Figure 5.2: Use case diagram 1

The User and the admin can at all times either:

- 1. Manage devices

- 2. Launch and control a drone
- 3. Replay recordings
- 4. Manually monitor the area live

The user will also be able to receive alerts; however, this is not triggered by the user and will instead be triggered by the system when a "threat" is detected within the video frame of the cameras.

The admin will be able to do the same, but they can also configure the base station. This include adding/removing users for that base station, as well as register the base station in the initial setup.

All video from the cameras are then both streamed live if the user wants a live feed and is also stored and can be watched back later on demand.

5.4 Final requirements

In this section we will list our final requirements for the system. These will be divided into functional and non-functional for the sake of testing them in the final implementation.

5.5 Testing Approach

The system will be tested using these functional and non-functional testing methods:

- **Functional testing**
  - Unit testing
  - Integration testing
  - System testing
  - User acceptance testing
- **Non-functional testing**
  - Load testing
  - Security testing
  - Usability testing
  - Reliability testing
  - Scalability testing
  - Compatibility testing

These testing types is well documented of IBM for further details [15].

5.5.1 Functional Requirements

Nr.	Requirement	System	Origin	Test Type
1.	The stationary cameras should continuously monitor the area.	Overall System	Scenario 1 – Takeaway 1	System testing
2.	The system should be able to alert the user if a threat is detected.	Overall System	Scenario 1 – Takeaway 2	Integration testing
3.	The user should be able to launch the drone at will.	Overall System	Scenario 1 – Takeaway 3	User acceptance testing
4.	The user should be able to control the drone, manually flying it while viewing the video stream in real time.	Overall System	Scenario 1 – Takeaway 4	System testing

5.	The system should be able to detect a "threat" autonomously.	Overall System / Cloud	Scenario 1 – Takeaway 5	Integration testing
6.	The system should support drone steaming with a delay of 200 ms to user interface to support reliable latency for drone control.	Overall System	Scenario 1 – Takeaway 6	System testing
7.	The stationary cameras should stream video when no threats are detected.	Overall System	Scenario 1 – Takeaway 7	System testing
8.	The system should provide a drone control interface that supports optional threat tracking.	Overall System	Scenario 1 – Takeaway 8	System testing
9.	The user should be able to use the drone for monitoring even when no threat is detected.	Overall System	Scenario 2 – Takeaway 1	User acceptance testing
10.	The system should store recordings from both the drone and stationary cameras.	Cloud / Base Station	Scenario 3 – Takeaway 1	Integration testing
11.	The user should be able to access the web interface to replay previous clips.	Cloud / UI	Scenario 3 – Takeaway 2	System testing
12.	The user should be able to navigate to specific timestamps where alerts were triggered.	Cloud / UI	Scenario 3 – Takeaway 3	User acceptance testing
13.	The system should log alert timestamps for later review.	Cloud	Scenario 3 – Takeaway 4	Unit testing
14.	The system should provide a searchable list of past alert events.	Cloud / UI	Scenario 3 – Takeaway 5	System testing
15.	The system should maintain a time-indexed database of recordings.	Cloud	Scenario 3 – Takeaway 6	Integration testing
16.	The system should support manual deployment and configuration of a Base Station at any location with power and Wi-Fi.	Base Station	Scenario 4 – Takeaway 1	User acceptance testing
17.	The base station should be able to connect to the cloud for registration and synchronization.	Base Station / Cloud	Scenario 4 – Takeaway 2	Integration testing
18.	The cloud should associate each base station with the correct user interface instance.	Cloud	Scenario 4 – Takeaway 3	System testing
19.	The base station should support connection to a local network for communication.	Base Station	Scenario 4 – Takeaway 4	System testing

20.	The system should verify successful cloud linkage before enabling user access.	Base Station / Cloud	Scenario 4 – Takeaway 5	Integration testing
21.	The system should allow configuration of the base station through an on boarding interface.	Base Station / UI	Scenario 4 – Takeaway 6	User acceptance testing
22.	The system should verify network connectivity during setup.	Base Station	Scenario 4 – Takeaway 7	System testing
23.	The cloud must uniquely identify each base station.	Cloud	Scenario 4 – Takeaway 8	Unit testing
24.	Communication between cloud and Base Station should be Authenticated.	Cloud / Base Station	Scenario 4 – Takeaway 9	Integration testing
25.	The user should be able to add and remove devices.	Base Station	Scenario 5 – Takeaway 1	User acceptance testing
26.	The user should be able to select from available protocols when registering a new device.	Base Station	Scenario 5 – Takeaway 2	System testing
27.	The system should automatically detect compatible devices during onboarding.	Base Station	Scenario 5 – Takeaway 3	Integration testing
28.	The user should be able to manage device configurations through an interface.	Base Station / UI	Scenario 5 – Takeaway 4	User acceptance testing
29.	The system should validate compatibility of newly added devices.	Base Station	Scenario 5 – Takeaway 5	Unit testing
30.	The system should maintain an updated inventory of connected devices.	Base Station	Scenario 5 – Takeaway 6	System testing

5.5.2 Non-Functional Requirements

Nr.	Requirement	System	Origin	Test Type
1.	The system should support drone steaming with a delay of 200 ms to user interface to support reliable latency for drone control.	Overall System	Scenario 1 – Takeaway 6	Load testing
2.	The system should automatically detect compatible devices during onboarding.	Base Station	Scenario 5 – Takeaway 3	Compatibility testing
3.	The cloud should support multiple base stations and users simultaneously.	Cloud	Availability	Scalability testing
4.	The system should support expansion with additional cameras and drones.	Overall System	Scalability	Scalability testing
5.	The base station should operate reliably under continuous load.	Base Station	Availability and reliability	Reliability testing
6.	The user should be able to manually control the drone using a low-latency video stream that supports responsive user interaction.	UI	Scenario 1 – Takeaway 4	Usability testing
7.	The user should with ease be able to navigate to specific timestamps where alerts were triggered.	UI	Scenario 3 – Takeaway 3	Usability testing
8.	Communication between cloud and Base Station should be Authenticated.	Base Station / Cloud	Scenario 5 – Takeaway 4	Security testing
9.	Communication between system components should be encrypted.	Overall System	Security	Security testing
10.	The user should be able to access the web interface to replay previous clips.	Cloud / UI	Scenario 3 – Takeaway 1 & 2	Security testing

Chapter 6

System design

In this section, we will start designing our system, using different diagrams to give an overview of the different elements that play a part in our overall system. Finally, an overview of the entire architecture will be presented.

Taking inspiration from the research made prior and following the steps of other systems, we started with the main goal of building a system that could interact with multiple cameras and drones. We decided that there should be a Base Station that would act as a fog node where the cameras and drones could connect to and that would then establish connection with the cloud where the heavy lifting would be done. We assumed that the Base Station could be any computer from small device like a raspberry pi to a full desktop with the latter performing much better. With this in mind we assigned the functionalities of our system assuming that the Base Station could not be a very powerful machine and for this reason it will act just as a proxy for sending video and receiving commands to control the drone.

The cloud is then responsible for processing the video and evaluating if any threats are present, to record the video from all the cameras so it can be accessible later and to alert the users in case of a positive detection.

With the responsibilities of each node clearly separated we can now see them as distinct entities that can be worked on independently as long as we define the medium of communication between them. In the spirit of proceeding with this separation we choose to stream the video from the Base Station to the Cloud using WebRTC, in order to have the lower possible latency. This was also an important decision being that our ambition was to remote control the drone through the browser and WebRTC offers the best latency since it establishes a direct connection from the user to the server (in this case the Base Station). On top of that, WebRTC encrypts the video that is sent which fits our use case perfectly.

Now that we have the video streaming protocol and we need to decide on how the WebRTC connection could be established as well as general connection from the Base Station to the Cloud. For this, gRPC presents the best option since it is a great solution for the master-slave architecture (the Cloud being the master and the multiple Base Stations being the slaves) we were envisioning. The cloud would be running a gRPC server to which the Base Stations could make requests to perform their multiple functions (connection, disconnection, setup WebRTC

connection). Another positive part of this technology is the fact that it is language agnostic meaning the Base Station code would not be bound to a single programming language, leaving it open to refactors in the future without the need for changes on the Cloud or vice-versa. It also allows for channel encryption in order to make sure all data exchanged between them is secure.

In summary, the main functional responsibilities of the system should be:
Basestation:

- Connect To Cloud
- Disconnect from Cloud
- Allow user to register new cameras and drones
- Gather video from registered cameras
- Send video to cloud
- Act as proxy for drone remote control

Cloud:

- Manage basestations
- Receive video
- Evaluate, record and show video from cameras
- Allow the user to establish connection to the drone

With this defined we could now focus on each part individually.

6.1 Context diagram

To get a better overview of the terminators of the system we used a context diagram.

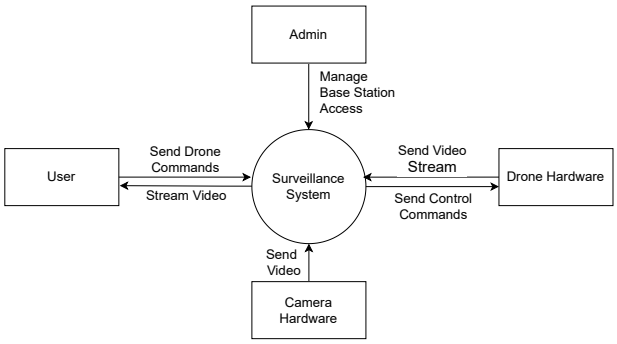

Figure 6.1: Context diagram

Terminators identified are:

- User
- Admin
- Drone Hardware
- Camera Hardware

The context diagram shows how the actors interact with the system.

The user should be able to control the drone manually using the system's interface, and the user should be able to view real-time video streams from the cameras or consult old recordings and be notified/alerted when a potential threat is detected.

The admins only role is to manage access control on the Base Station for all Users. The admin inherits all functions of the user and will therefore also be able to view video and control drones.

The drone will be able to receive control commands from the system, as well as sending its video feed to the system.

The cameras will be providing the system with their video stream.

6.2 Base Station

We decided that to make the Base Station fulfill its purpose it had to have three main components. A user interface where the person setting up the system could manage all the cameras and drones that belonged to that area. The types of devices should be already registered and available in the system so they can be selected; A database where information about the active devices and their configurations could be stored; and a core module where the video would be captured and redirected, and where the communication with the exterior would happen.

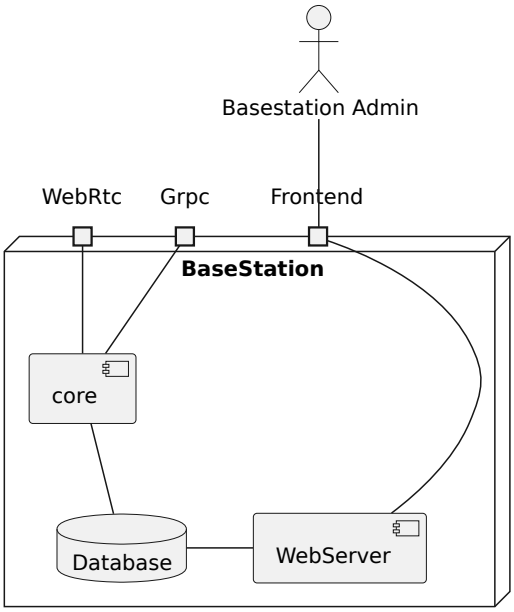

Figure 6.2: Base-station's Component Diagram

6.2.1 Database

For the database, we decided to go with *postgresql*, since it presents itself as an easy to use, open-source relational database that offers atomicity in its operations, ideal for the way we want it to communicate with both the Core module 6.2.3 and the Web Server 6.2.2. Along with that, the support for JSON columns is a great feature for our support of many types of devices. The database has two tables one for Cameras and one for Drones, and stores for each an id, a name, the type of device (so the system can have knowledge of how to behave in each case), a json with the configuration that is exclusive to each type of device and is managed by the web server, and timestamps for creation and update.

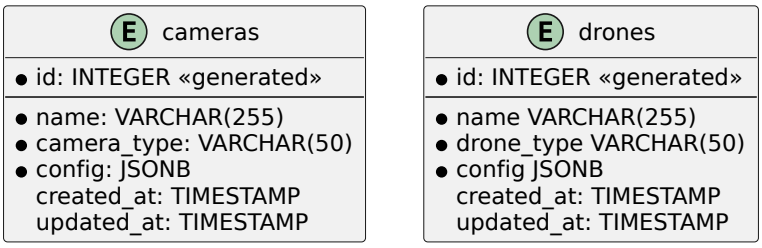

Figure 6.3: Base-station's Entity Diagram

6.2.2 Web Server

In the Base Station, the web server is a simple module that creates a local user interface so that the owner can add and remove cameras, as well as see which of these are available. We can see in 6.4 how the flow of the frontend should look when trying to add a new device to the system. First, the user should choose the device to add (camera or drone). Then choose the type of device to add (in the case of camera could be IP Camera, Usb Camera, etc...), and depending on this it will appear the fields that need to be filled so that the system can communicate with this device, based on the configuration that is coded into the application. The web server makes sure that the type of the device is correct in relation to its configuration and after verification it writes it to the database.

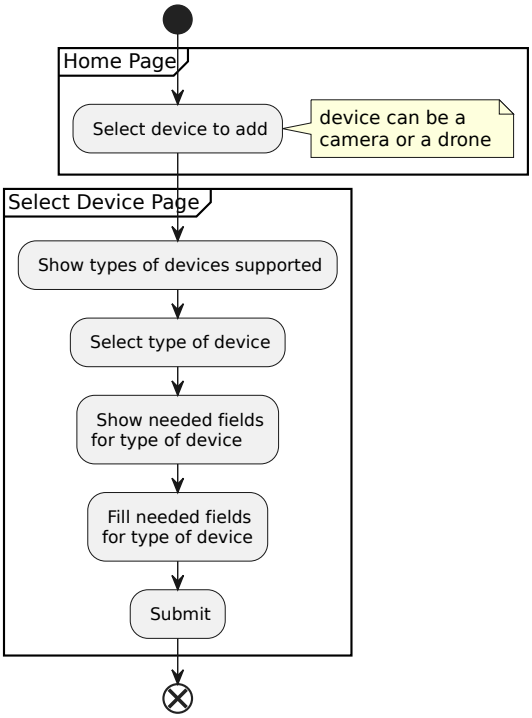

Figure 6.4: Activity Diagram of the User Interface

6.2.3 Core

The core module of the Base Station is where the most important functions inside this machine occur. Inside run three separates components simultaneously: Keep Alive, Drone Manager, and Camera Manager.

Keep Alive

This component runs cyclically every thirty seconds sending a Ping Request through gRPC and expects to receive a response. This signals the remote server that the current Base Station is still alive and in case of a response being received that the remote server is also active.

Camera Manager

The Camera Manager component is responsible for checking which cameras are active by polling the Database and stream their video to the remote server. It does this by polling the database every 10 seconds. If a new camera is detected the component initiates the WebRTC connection sequence with an instance of the desired camera. The WebRtc connection is established by exchanging messages through gRPC with the remote server and can be observed in 6.5. When the Manager detects a camera has been removed it should stop transmitting video.

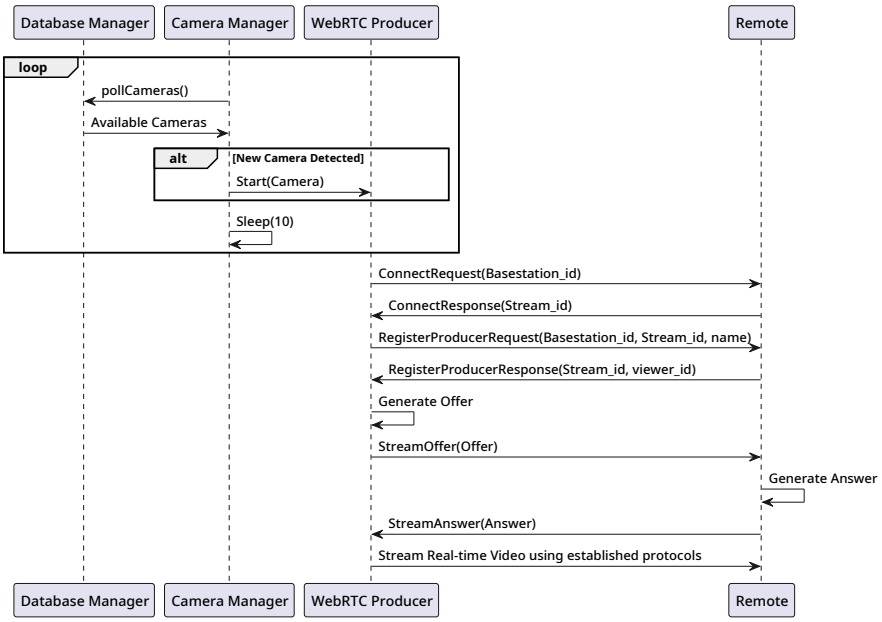

Figure 6.5: Sequence diagram showing the establishment of a WebRTC video stream

In order to support multiple camera types we define that the system should accept any camera class that inherits from the class *VideoStreamTrack* that is provided by the *aiortc* python library (library responsible for the WebRtc connection) and implements the recv method that should return a Video Frame when called. This way it should be possible to implement any type of cameras needed. 6.7

Drone Manager

The Drone Manager is quite similar to the Camera Manager in most of its genesis except for the fact that while in the cameras case, we want to continuously stream their video to the remote server, with the drones, we only want to establish a WebRTC connection when the user requests access to the drone, so that the connection is made directly with the user's browser, reducing latency. For this, the drone process that is running, once a drone has been added, connects to the remote sever signaling that a drone from that Base Station is now active and can be accessed. Then it proceeds to poll the remote server every half a second to check if connection to that drone has been requested. In case the response returns true, the WebRtc producer starts to exchange information with the remote server in order to establish connection. 6.6

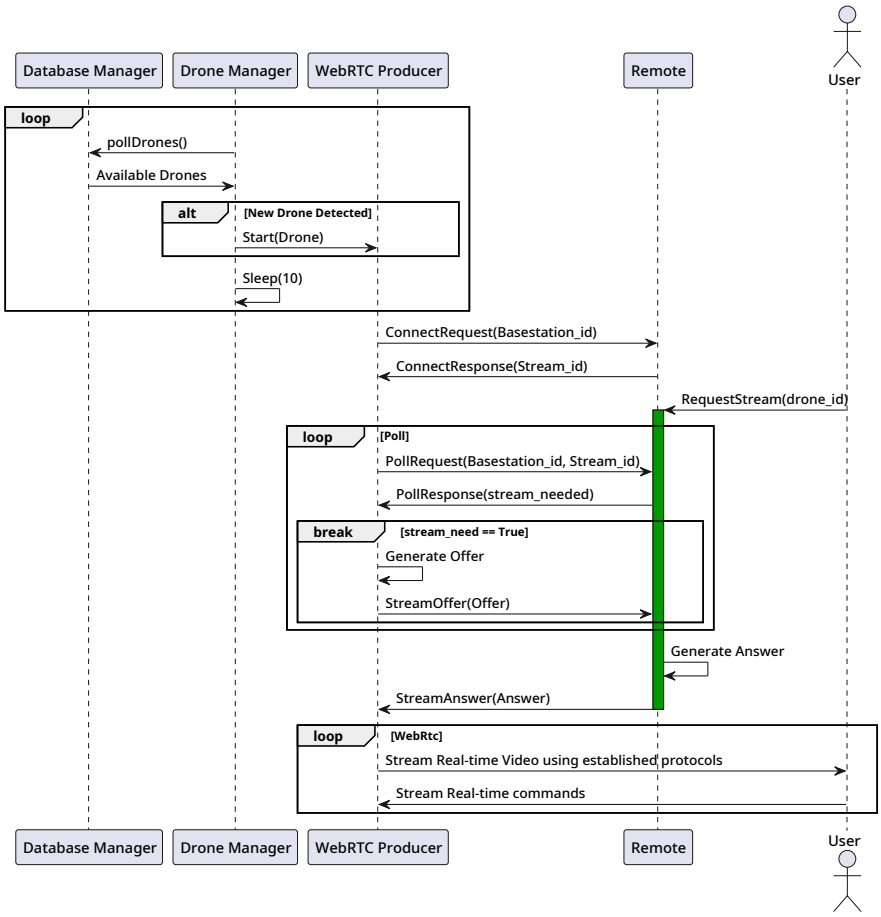

Figure 6.6: Sequence diagram showing the establishment of a WebRTC drone stream

In order to support multiple drone types, we define that the system should accept any drone class that implements the *IDrone* class and must have an associated class that inherits

VideoStreamTrack so the video can be received. The abstract class contains the method "decode" that receives input from the user, parses it, and executes the corresponding method associated with the implementation of its type of drone 6.7. For the Dji Tello Drone we are using, we created a support **Movement** class that holds the values of where the drone should move for every direction at any given moment. These values are continuously sent to the drone.

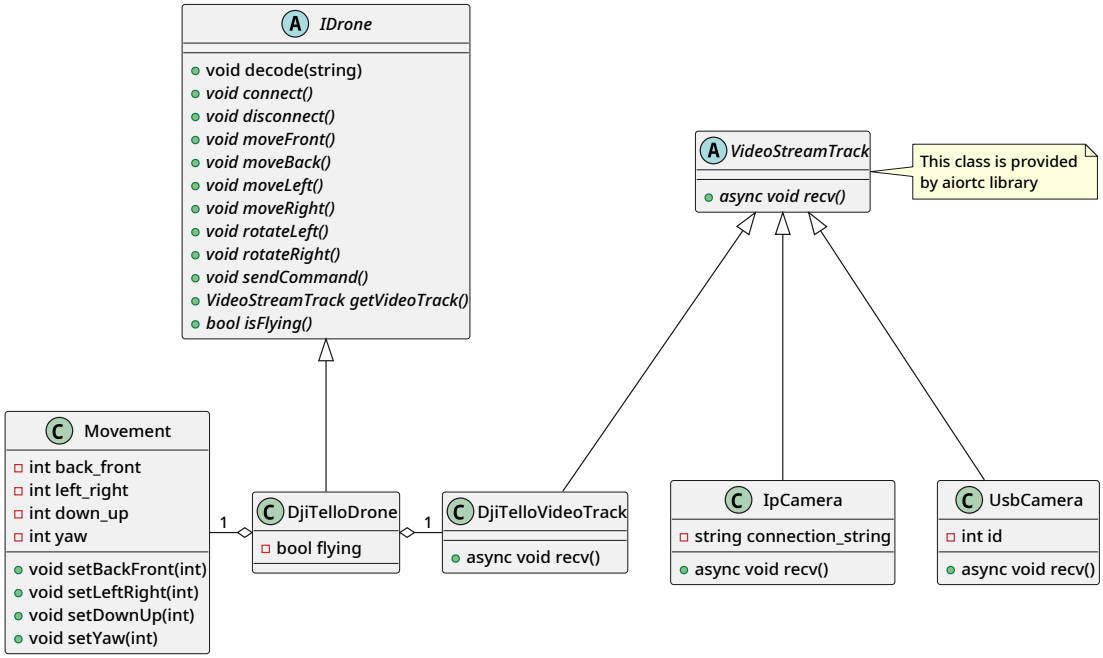

Figure 6.7: Camera and Drone Class Diagram

6.3 Remote Server / Cloud

In the Remote Server (Cloud) is where most of the functions of the system are achieved. We decided to divide this node into distinct parts: Proxy, Database, Video Storage, Threat Detector, Notifier and Web Server.

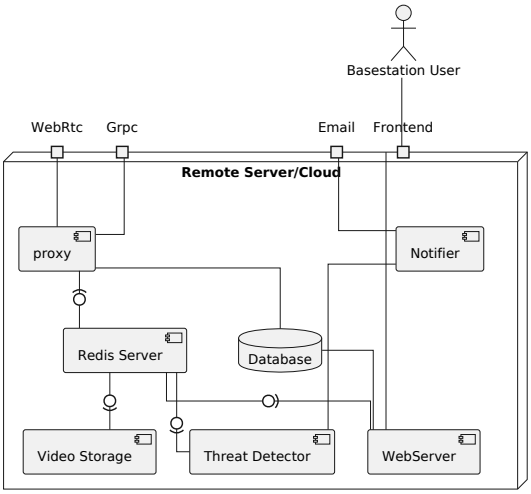

Figure 6.8: Remote Server Component Diagram

6.3.1 Database

For the Database we decided to use a relational database and for sake of consistency we kept the postgresql that we also used for the Basestation. The database is responsible for storing all the information related with Basestations, users and user permissions. It is composed by a Basestations' table that keeps track of identifiers, names, passwords and status. A users' table storing username and passwords, and finally a relation table that assigns users to Basestations. This table has the column owner, denoting if the corresponding user is the owner of that Basestation, giving them more permissions. The Entity relationship diagram can be seen in 6.9.

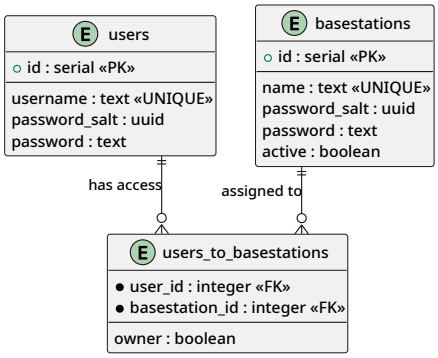

Figure 6.9: Database's Entity Relationship Diagram

6.3.2 Proxy

The Proxy is the unit of communication between the remote server and the Basestation, all messages pass by this component. It contains the gRPC server where it can receive requests from multiple locations. It receives connection requests from Basestations and registers them in the database. Then listens for Ping Requests of the available Basestations and if any of them do not respond for **2 minutes** its status is changed to inactive. It is also responsible for receiving the video streams and publishing them to the Redis server, allowing the rest of the services to consume them. It acts as a medium for signaling the WebRTC communication between the user and Basestation when trying to control the drone.

6.3.3 Redis Server

The Redis server supplies us with a publish/subscribe service that we can use to relay the video streams inside the cloud. The proxy pushes video frames, and an heartbeat signal that can be read by the consumers to identify which streams are active and can be shown.

6.3.4 Threat Detector

The Threat Detector consumes video frames from the Redis Server and evaluates them every **200 ms**, for threats. If a threat is detected the notifier service is called with a frame of the detected threat.

6.3.5 Notifier

The notifier receives a frame and the Base Station Id and emails it to the users associated with that Base Station in the database.

6.3.6 Video Storage

The Video Storage consumes video frames from the Redis Server and stores them in **10 second** MP4 videos on the disk.

6.3.7 Web Server

The Web Server is the sub-part that allows the user to interact with the system itself. We use flask with templates to manage the flow of the website. First, a user needs to register with a new username and can then login. The users are then faced with the home page where they should see the Base Stations assigned to them and the option to add a new Base Station. In the add Base Station view two inputs are required: id and password. The password is set up in the Base Station configuration while the id is assigned when the Base Station starts and could be seen in its console. The Base Station is then added to the user requesting it, if the credentials are correct and no other user has already registered it. Inside the Base Station view (when a Base Station is selected) the user has three different options plus two more if he is the owner: Views surveillance cameras, view recordings from the surveillance cameras and views

available drones. Additionally in case of being an admin, can also add and remove users from the Base Station using their username.

In the camera view, the web server consumes the Redis server streams showing only the ones that have an active heartbeat message (less than **thirty seconds** have passed since last message) allowing the user to watch all the video cameras in a single window.

In the drone view, a list of available drones are listed, this list is requested from the proxy server that keeps record of the active drones by receiving connections from the Base Stations. Upon clicking a drone, the server provides the user with the web page that along with some Javascript will establish a WebRTC connection to the Base Station and capture user input in the page, that will be sent through a WebRTC data channel.

6.4 Final architecture design

We used a deployment diagram to expand our vision of the system and the sub-parts of it.

As the diagram shows (Figure : 6.10) our initial vision as a single cloud service as the core entity of the system, pulling the heavy weight in terms of supplying the user with video footage, user access control and drone control interface. The Edge Server will then be running on the Base Sttation deployed on site and act as a gateway for the end devices, which are the cameras and drones.

There will be a single cloud entity supervising multiple base stations. And each base station will be able to support multiple cameras and drones.

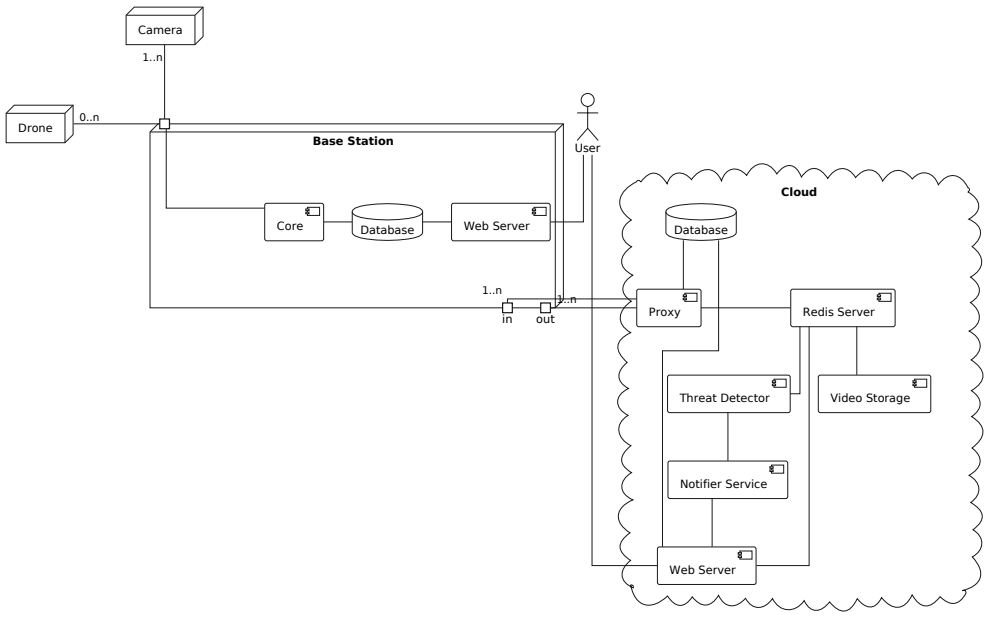

Figure 6.10: Deployment Diagram

Chapter 7

Implementation

In this chapter we discuss how we implemented the decisions we took in the previous chapter, elaborating on technical aspects and providing code snippets to better explain our work. The file structure of the project closely follows the architecture as can be seen in 7.1

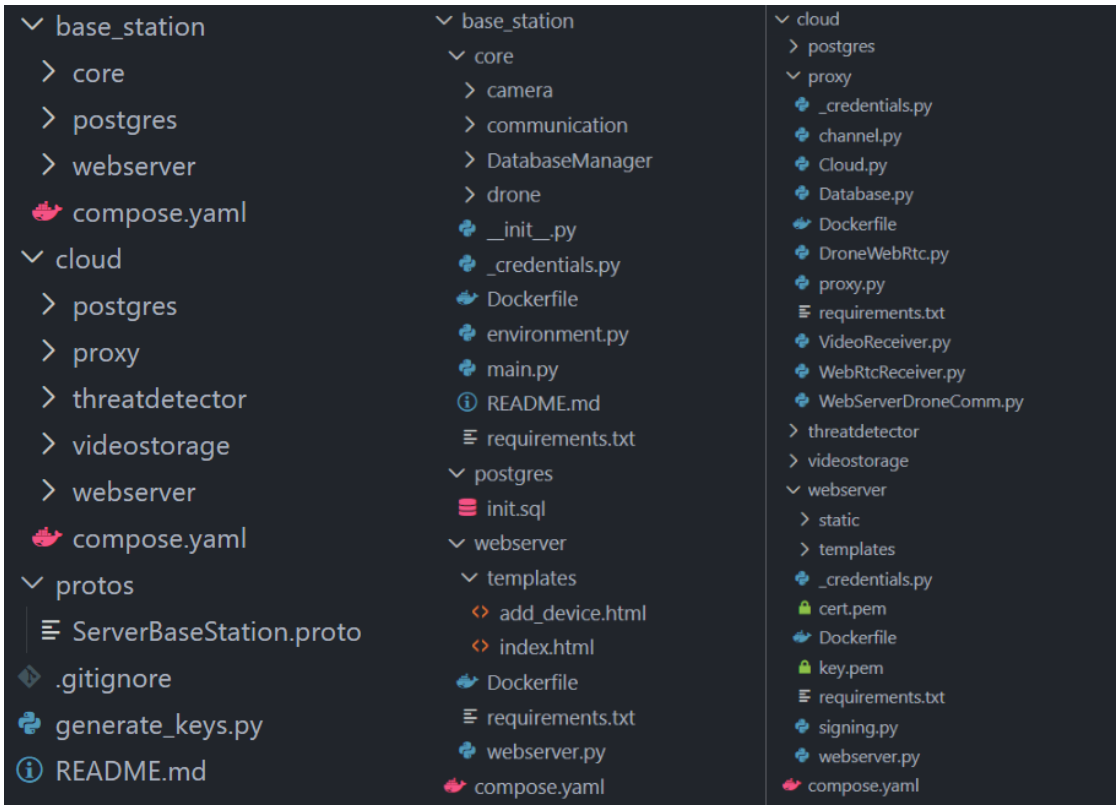

Figure 7.1: Overall Structure of the Project

We describe the implementation in the order in which a user would configure the system, with the aim of improving comprehensibility.

7.1 Key Generation

To enable secure communication among the multiple nodes in the system, credentials for gRPC channel encryption must be generated. To this end, we provide a Python script that is executed at the beginning of the setup process. This script generates RSA key pairs using the OpenSSL utility, which is available on most Unix-based systems.

The generated client keys are copied into the core directory of the Base Station and into the web server directory in the cloud, as both components act as clients of the gRPC server. The server keys are placed in the proxy directory, where the gRPC server can access them. Those credentials are then read by a python script placed in those locations.

In addition, public certificates are generated for use by the Flask web application hosted on the web server in order to establish a secure HTTPS connection.

7.2 Base Station

As mentioned in the design chapter, the Base Station contains three services. Two of them, the Web Server and the Database are grouped into a docker compose, the Core module ideally would run inside this compose making it very easy to deploy, however if we want to use usb cameras/built-in cameras from the Base Station device, it needs to run bare metal to be able to access them. Both the compose and the core module require a configuration file to be attached with Base Station and local database information, as follows.

```
GRPC_REMOTE_IP="" # IP of the Grpc Server
GRPC_REMOTE_PORT= # Port of the Grpc Server
BASESTATION_NAME="" # Name of the Basestation (must be unique among basestations)
BASESTATION_PASSWORD="" # Password of the Basestation (to be used for registration)
# Local database configuration
DB_HOST=""
DB_PORT=""
DB_USER=""
DB_PASSWORD=""
DB_NAME=""
```

Listing 7.1: Configuration File Template

7.2.1 Core

Inside the core main function, we start by Connecting to the Cloud using gRPC with the credentials previously generated. If the connection is successful the remote server returns the

assigned Base Station Id that can than be used to call the other functions executed by this module.

ServerBasestation_pb2_grpc and *ServerBasestation_pb2* are generated files from the gRPC protobuf where the python code for the gRPC communication resides.

```
async def main():
    creds = grpc.ssl_channel_credentials(_credentials.ROOT_CERTIFICATE)
    dbMan = DatabaseManager()

    identifier = -1
    name = CONFIG.get("BASESTATION_NAME")
    password = CONFIG.get("BASESTATION_PASSWORD")
    address = f"{CONFIG.get('GRPC_REMOTE_IP')}:{CONFIG.get('GRPC_REMOTE_PORT')}"
    while identifier == -1:
        sleep(1)
        async with grpc.aio.secure_channel(address, creds) as channel:
            stub = ServerBaseStation_pb2_grpc.CloudStub(channel)
            response =  await stub.Connect(ServerBaseStation_pb2.ConnectToCloudRequest(name=name, password=password))
            identifier = response.id

    print("Got Id:", identifier)
    camManager = CameraManager(identifier, name, dbMan)
    droneManager = DroneManager(identifier, name, dbMan)
    print("Started")
    await asyncio.gather(camManager.run(), droneManager.run(), stayalive(address, identifier, creds))
```

Listing 7.2: Configuration File Template

Camera Manager

The Camera Manager task calls the pollCameras method inside Database Manager that returns a list of available camera ids.

```
def pollCameras(self):
    self.cursor.execute("SELECT id FROM cameras")
    rows = self.cursor.fetchall()
    return [row[0] for row in rows]
```

Listing 7.3: pollCameras Database Manager method

If a new row in the cameras database is detected then a new process is started and stored along with a signal in association with the camera id. The signal is activated once a camera has been removed and warns the task that it should stop.

```
def __registerCamera(self, cam: int, event_loop):
    camera = self.dbMan.getCameraById(cam)
    cam_name = self.dbMan.GetCameraNameById(cam)
    producer = WebRTCProducer(basestation_id=self.identifier,
   source=camera, stream_name=cam_name)
    signal = asyncio.Event()
    print("Starting process...")
    task = event_loop.create_task(producer.start(signal))
    return {cam: (task, signal)}
```

Listing 7.4: Method from camera registration

The producer start function that was initiated in a new process, proceeds to establish a WebRTC connection, via gRPC following the sequence diagram we defined previously 6.5.

```
async def __on_start_stream(self):
    ice_server = RTCIceServer(urls='stun:stun.l.google.com:19302')
    self.pc = RTCPeerConnection(RTCConfiguration(iceServers=[
   ice_server]))
    self.pc.addTrack(self.source)
    offer = await self.pc.createOffer()
    await self.pc.setLocalDescription(offer)

async def start(self, q):
    print(f"Starting producer: {self.stream_name}")
    self.q = q
    address = f"{CONFIG.get('GRPC_REMOTE_IP')}:{CONFIG.get('
   GRPC_REMOTE_PORT')}"
    self.channel = channel = grpc.aio.secure_channel(address,
   creds)
    self.stub = ServerBaseStation_pb2_grpc.WebRtcStub(channel)
    response = await self.stub.Connect(ServerBaseStation_pb2.
   ConnectRequest(basestation_id=self.basestation_id, name=self.
   stream_name))
    self.stream_id = response.stream_id
    self.name = self.stream_name
    response = await self.stub.Register(
                    ServerBaseStation_pb2.RegisterProducerRequest(
                        basestation_id=self.basestation_id,
                        stream_id=self.stream_id,
                        name=self.name
                    ))
```

```
    await self.__on_start_stream()
    response = await self.stub.Stream(
        ServerBaseStation_pb2.StreamOffer(
            basestation_id=self.basestation_id,
            stream_id=self.stream_id,
            offer=ServerBaseStation_pb2.StreamDesc(
                type=self.pc.localDescription.type,
                sdp=self.pc.localDescription.sdp)))

    await self.pc.setRemoteDescription(
        RTCSessionDescription(sdp=response.answer.sdp, type=
   response.answer.type)
    )
    while not self.q.is_set():
        await asyncio.sleep(1)
    await self.__cleanup()
```

Listing 7.5: Method from camera registration

Drone Manager

The Drone Manager behavior and its WebRTC Producer are similar to that of the Camera Manager, differing only in Poll Requests sent by the drone producer and by the data channel need for remote control of the drone. The data channel is defined as such, calling the decode method from the IDrone class when a new message is received on that channel:

```
channel = self.pc.createDataChannel("commands")
@channel.on("message")
async def on_message(message):
    self.source.decode(message)
```

Listing 7.6: Channel definition in Drone WebRTC Producer

7.2.2 Web Server and Database

The Web Server runs a Flask app and uses psycog python library to interact with the database. Here is where the Database is initialized and the configuration of the supported cameras and drones are.

```
CAMERA_TYPES = {
    'ip_camera': {
        'name': 'IP Camera',
        'fields': [
            {'name': 'ip_address', 'label': 'IP Address', 'type':
   'text', 'required': True},
```

```
            {'name': 'port', 'label': 'Port', 'type': 'number', 'required': True},
            {'name': 'username', 'label': 'Username', 'type': 'text', 'required': False},
            {'name': 'password', 'label': 'Password', 'type': 'password', 'required': False}
        ]
    },
    'usb_camera': {
        'name': 'USB Camera',
        'fields': [
            {'name': 'device_id', 'label': 'Device ID', 'type': 'number', 'required': True},
        ]
    },
}
```

Listing 7.7: Example of supported cameras' fields

This is used in the HTML templates ending up with this view for the Ip Camera:

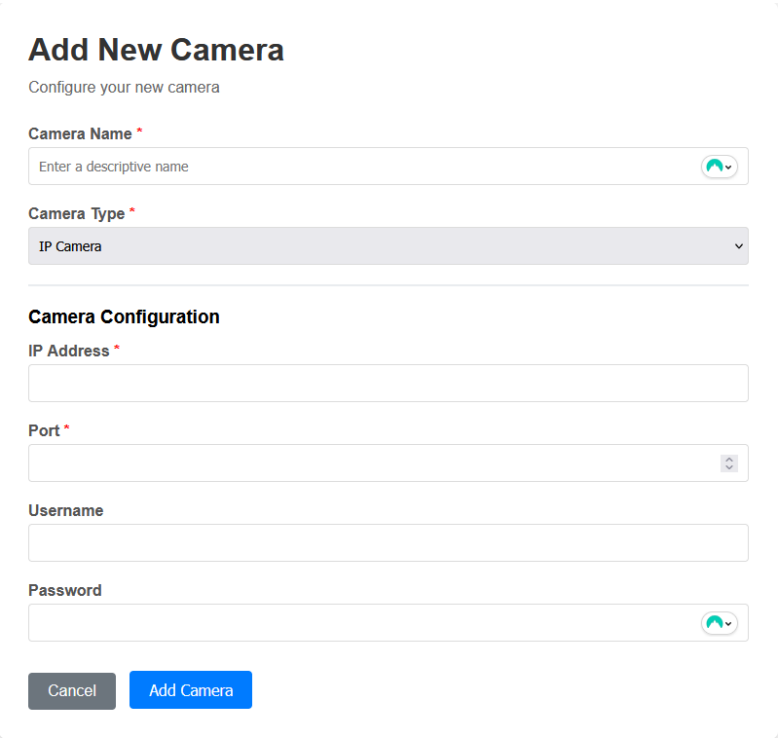

Figure 7.2: Add Camera Page

7.3 Remote Server/Cloud

The Remote Server services are all grouped inside one docker compose, making it extremely simple to deploy, with the only steps needed before hand being, generate keys and generate gRPC files. All of these services are developed in python using similar libraries as in the Base Station, the main ones being: psycogpg for database access, aiortc for WebRTC streaming, Flask for the Web Server and Yolo for the Threat detection in the video.

7.3.1 Web Server

The Web Server, built with Flask, uses the flask_login library to handle authentication and session management, making it simpler to interact with the user permissions. The Web Server uses a url hierarchy to navigate between pages, for example if a user would like to access the page of a Base Station with id 1 the url would be (on localhost) **https://127.0.0.1/basestation/1**, if then the same user would like to access the cameras view inside this Base Station, the server would redirect to **https://127.0.0.1/basestation/1/cameras**. To access these pages however, the user must be authenticated and have permissions to access this Base Station, to ensure that no unauthorized personnel could view the cameras (or access any other functionality) from any Base Station. Each of these sub domains have HTML templates associated with them to define the structure of the page.

Here an example can be seen of the Base Station view and its template:

Figure 7.3: Base Station View Page

```
@webserver.route("/basestation/<int:basestation_id>")
def basestation(basestation_id):
    if not user_has_access(basestation_id):
        return "Unauthorized"
    basestations = pd.read_sql("select id, name from basestations where id = %s::integer", conn, params=[basestation_id],)
    basestation_info = basestations.to_dict("records")[0]
    return render_template("basestationview.html", basestation=basestation_info, admin=user_has_access(basestation_id, True))
```

Listing 7.8: Endpoint for Basestation in the Flask app

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Home</title>
    <link rel="stylesheet" href="../static/style.css">
</head>
<body>
    <div class="header">
        <h1>Base Station: {{ basestation.name }}</h1>
        <div class="user-info">
            <span class="username">{{ username }}</span>
            <a href="{{ url_for('logout') }}" class="logout-btn">Logout</a>
            <a href="{{ url_for('home') }}" class="home-btn">Home</a>
        </div>
    </div>
    <li>
        <a href="/basestation/{{basestation.id}}/drones">Drone view</a>
    </li>
    <li>
        <a href="/basestation/{{basestation.id}}/cameras">Camera view</a>
    </li>
    <li>
        <a href="/basestation/{{basestation.id}}/recordings">Recordings</a>
    </li>
    {% if admin %}
    <li>
        <a href="/basestation/{{basestation.id}}/adduser">Add User
```

```
    to Basestation</a>
    </li>
    <li>
        <a href="/basestation/{{basestation.id}}/removeuser">
   Remove User from Basestation</a>
    </li>
    {% endif %}
    <ul>
    </ul>
</body>
</html>
```

Listing 7.9: Template basestationview.html

In the template, it can also be seen all the other endpoints we can access from the main Basestation view.

When trying to access the drone control page, the web server supplies the user with javascript to establish connection through WebRTC, show its video, and to capture keyboard input that will be sent to the Base Station and later decoded into drone commands. The connection is established by calling the endpoints **/request**, to signal that a stream is being requested and expecting to receive a stream offer, and **/answer** with the answer generated from the offer.

7.3.2 Proxy

The Proxy component implements the classes associated with the gRPC Server.

Cloud

Handles connections and pings from Basestations.

WebRTCReceiver and WebRTCDroneReceiver

Handle communication to the WebRTC Producers in the Base Stations. Both are similar but the drone receiver also listens for polls replying with a boolean value representing if a connection to that drone has been requested by the User.

WebserverDroneCommuncationDetails

Receives requests from the Web Server, mainly to request available drones and establish WebRTC connection with the Drone pilot's browser.

7.3.3 Redis Server

The Redis Server runs inside docker using a public redis image: redis:6.2-alpine. It exposes port 6379 to the others containers to allow them to access it.

7.3.4 Threat Detector

The Threat detector consumes frames from the the Redis Server as long as they have a valid Heartbeat. Then, starts a new process for every active stream channel. It uses the yolo v8 nano model in python, since in our tests it seem to be able to identify people, quite well with low overhead. We specified a 0.5 confidence threshold to avoid false positives. After a detection is made we register the time, and only evaluate the image again after **20 seconds**, so as not to spam the email receivers.

```
async def process_stream(channel, r):
    """Runs YOLO on the latest frame of this stream every N
   milliseconds."""
    print(f"[DETECTOR] Watching stream: {channel}")
    while True:
        frame_key = f"latest_frame_{channel}"
        heartbeat_key = f"heartbeat_{channel}"
        # confirm active stream
        hb = await r.get(heartbeat_key)
        if hb is None:
            await asyncio.sleep(1)
            continue
        # read frame
        b64 = await r.get(frame_key)
        if not b64:
            await asyncio.sleep(0.1)
            continue
        try:
            frame = decode_jpg(b64)
        except Exception:
            await asyncio.sleep(0.1)
            continue
        # cooldown
        last = cooldowns.get(channel, 0)
        if (time.time() - last) < COOLDOWN_SECONDS:
            await asyncio.sleep(PROCESS_INTERVAL)
            continue
        # run YOLO
        results = model.predict(frame, verbose=False)
        threats = detect_threat(results)
        if threats:
            print(f"[{channel}] THREAT DETECTED:", threats)
            cooldowns[channel] = time.time()
            # Get annotated image with bounding boxes
            annotated_frame = results[0].plot()  # Returns image
   with boxes drawn
```

```
            # send email
            send_alert_email(channel, threats, annotated_frame)
        await asyncio.sleep(PROCESS_INTERVAL)
```

Listing 7.10: Snippet of Threat Detector

7.3.5 Notifier

The Notifier was initially intended to alert users of threats by email. However, this proved a bit difficult due to email provider restrictions. We kept the component as it receives a frame of the detected threat and can then be extended to have any notifier behavior we need. Currently, it simply displays the captured frame in the machine executing it.

7.3.6 Video Storage

The Video Storage consumes frames from every active channel in the Redis Server like the Threat Detector and uses an FFmpeg process to concatenate the jpegs received into 10 second mp4 videos.

These videos are saved to shared volume in docker that can be accessed by the Web Server to show them in the Recordings Page of each Base Station.

```
# Check if 10 seconds have passed
current_time = time.time()
if proc is not None and (current_time - last_save_time) >= 10:
    print(f'[{channel}] New recording segment saved')
    proc.stdin.close()
    proc.wait()
    with self.global_lock:
        self.processes.remove(proc)
    proc = None
    file_counter += 1
    last_save_time = current_time

# Create new ffmpeg process if needed
if proc is None:
    timestamp = datetime.utcfromtimestamp(current_time).strftime('
    %Y%m%d_%H%M%S')
    video_name = f"videos/{stream_info['basestation_id']}_{
    stream_info['display_name']}_time_{timestamp}.mp4"
    proc = subprocess.Popen([
        'ffmpeg', '-y',
        '-f', 'rawvideo',
        '-vcodec', 'rawvideo',
        '-pix_fmt', 'bgr24',
        '-s', f'{width}x{height}',
```

```
        '-r', '30',
        '-i', '-',
        '-c:v', 'libx264',
        '-pix_fmt', 'yuv420p',
        '-preset', 'ultrafast',
        video_name
    ], stdin=subprocess.PIPE, stderr=subprocess.DEVNULL)
    with self.global_lock:
        self.processes.append(proc)
    last_save_time = current_time
    print(f"[{channel}] Started new video segment: {video_name}")

proc.stdin.write(frame.tobytes())
```

Listing 7.11: Snippet of Video Writing

Chapter 8

Testing

After building and testing the system we look back at our requirements we initially made after our study and will see how many of these the system has fulfilled.

We have marked for each requirement a status with either a "✓", "/" or a "X" if the requirement are:

- ✓: Fulfilled
- /: Partially fulfilled
- X: Not Fulfilled

We will also for the functional and non-functional requirements address the incomplete or partially complete requirements and share our status on those and why we did not complete those.

Before addressing the requirement verification we want to go through our verification processes for our system, as mentioned in chapter 5 our testing methods include:

- **Functional testing**
  - Unit testing
  - Integration testing
  - System testing
  - User acceptance testing
- **Non-functional testing**
  - Load testing
  - Security testing
  - Usability testing

- Reliability testing
- Scalability testing
- Compatibility testing

8.0.1 Testing procedure examples

As an example of our testing procedure, we will illustrate how we tested some of the requirements of the system using the methods.

Functional requirement test examples

- Requirement 1: The stationary cameras should continuously monitor the area. : System testing

  Since this test involves the interaction of multiple system components—specifically the Base Station, Cloud, and the camera connected to the Base Station—conducting a system test was the appropriate approach to verify their functionality.

  Our test involved initialization of the Base Station and a Camera. After that we used the web interface to test if we could access the live video stream being streamed from the camera by accessing the Base Station view. As this was possible, the test was concluded successful.

- Requirements 2: The system should be able to alert the user if a threat is detected.

  This requirement falls into integration testing because it involves verifying the interaction and communication between multiple system components working together.

  As stated earlier, we are not working in this project with the implementation and training of the AI model. We are therefore not testing the model itself and if its correctly identifying threats. We assume that the model is working correctly and testing if we are able to use the model to trigger an alert and handle that alert in the system and present it to the user.

  The test therefore involved applying the model on the incoming streams and transporting the alert trigger to the user.

  The test involved running the system and testing if the user would be alerted by the system itself when the Object detection model would trigger the alert mechanism.

  This requirement we marked as partially fulfilled. This is because our system was internally able to parse the incoming video footage from the live feed to the object detection model and the system was able to tell when the model detected something it defined as a "threat". However, in the current version of the system we do not have a working functionality to alert the user. We wanted to implement an email service to alert the users but we did not have time to complete this.

8.0.2 Functional Requirements Verification

Nr.	Requirement	System	Test Type	Status
1.	The stationary cameras should continuously monitor the area.	Overall System	System testing	✓
2.	The system should be able to alert the user if a threat is detected.	Overall System	Integration testing	/
3.	The user should be able to launch the drone at will.	Overall System	User acceptance testing	✓
4.	The user should be able to control the drone, manually flying it while viewing the video stream in real time.	Overall System	System testing	✓
5.	The system should be able to detect a "threat" autonomously.	Overall System / Cloud	Integration testing	✓
6.	The system should support drone steaming with a delay of 200 ms to user interface to support reliable latency for drone control.	Overall System	System testing	✓
7.	The stationary cameras should stream video when no threats are detected.	Overall System	System testing	✓
8.	The system should provide a drone control interface that supports optional threat tracking.	Overall System	System testing	/
9.	The user should be able to use the drone for monitoring even when no threat is detected.	Overall System	User acceptance testing	✓
10.	The system should store recordings from both the drone and stationary cameras.	Cloud / Base Station	Integration testing	/
11.	The user should be able to access the web interface to replay previous clips.	Cloud / UI	System testing	✓
12.	The user should be able to navigate to specific timestamps where alerts were triggered.	Cloud / UI	User acceptance testing	X
13.	The system should log alert timestamps for later review.	Cloud	Unit testing	X
14.	The system should provide a searchable list of past alert events.	Cloud / UI	System testing	X
15.	The system should maintain a time-indexed database of recordings.	Cloud	Integration testing	✓

16.	The system should support manual deployment and configuration of a base station at any location with power and Wi-Fi.	Base Station	User acceptance testing	✓
17.	The base station should be able to connect to the cloud for registration and synchronization.	Base Station / Cloud	Integration testing	✓
18.	The cloud should associate each base station with the correct user interface instance.	Cloud	System testing	✓
19.	The base station should support connection to a local network for communication.	Base Station	System testing	✓
20.	The system should verify successful cloud linkage before enabling user access.	Base Station / Cloud	Integration testing	✓
21.	The system should allow configuration of the base station through an on boarding interface.	Base Station / UI	User acceptance testing	✓
22.	The system should verify network connectivity during setup.	Base Station	System testing	X
23.	The cloud must uniquely identify each base station.	Cloud	Unit testing	✓
24.	Communication between cloud and Base Station should be Authenticated.	Cloud / Base Station	Integration testing	✓
25.	The user should be able to add and remove devices.	Base Station	User acceptance testing	✓
26.	The user should be able to select from available protocols when registering a new device.	Base Station	System testing	✓
27.	The system should automatically detect compatible devices during onboarding.	Base Station	Integration testing	X
28.	The user should be able to manage device configurations through an interface.	Base Station / UI	User acceptance testing	✓
29.	The system should validate compatibility of newly added devices.	Base Station	Unit testing	X
30.	The system should maintain an updated inventory of connected devices.	Base Station	System testing	✓

8.0.3 Incomplete functional requirements

- 2. The system should be able to alert the user if a threat is detected. : /
  Disclosed above (8.0.1)
- 8. The system should provide a drone control interface that supports optional threat tracking. : X
  This was not implemented as drone latency was already quite high since the drone was not that powerful. However it could be added quite easily with more time.
- 10. The system should store recordings from both the drone and stationary cameras. : /
  Video recordings are saved from the cameras but not for the drones. With more time we could have implemented this as well.
- 12 The user should be able to navigate to specific timestamps where alerts were triggered. : X
- 13 The system should log alert timestamps for later review. : X
- 14 The system should provide a searchable list of past alert events. : X

  Requirements 12, 13 and 14 was not fulfilled as we did not have time to implement this function of saving timestamps of threat detection, with more time it could be done.
- 22 The system should verify network connectivity during setup. : X

  The system does not verify network connectivity, but if the cloud is not reachable the system will naturally not work.
- 27 The system should automatically detect compatible devices during onboarding: X

  The system assumes that the correct protocol and device type is selected, if there is a mismatch the system wont work as expected.
- 29 The system should validate compatibility of newly added devices: X

  The system can not do this. This was never a priority due to other main focus points of this project.

8.0.4 Non-Functional Requirements Verification

Nr.	Requirement	System	Test Type	Status
1.	The system should support drone steaming with a delay of 200 ms to user interface to support reliable latency for drone control.	Overall System	Load testing	/
2.	The system should automatically detect compatible devices during onboarding.	Base Station	Compatibility testing	X

3.	The cloud should support multiple base stations and users simultaneously.	Cloud	Scalability testing	✓
4.	The system should support expansion with additional cameras and drones.	Overall System	Scalability testing	✓
5.	The base station should operate reliably under continuous load.	Base Station	Reliability testing	✓
6.	The user should be able to manually control the drone using a low-latency video stream that supports responsive user interaction.	UI	Usability testing	/
7.	The user should with ease be able to navigate to specific timestamps where alerts were triggered.	UI	Usability testing	X
8.	Communication between cloud and Base Station should be Authenticated.	Base Station / Cloud	Security testing	✓
9.	Communication between system components should be encrypted.	Overall System	Security testing	/
10.	The user should be able to access the web interface to replay previous clips.	Cloud / UI	Security testing	✓

8.0.5 Incomplete non-functional requirements

- 1. The system should support drone steaming with a delay of 200 ms to user interface to support reliable latency for drone control. : /

  The system can experience lags and delays due to a not so powerful drone. But at times the system runs within the expected latency

- 2. The system should automatically detect compatible devices during onboarding.. : X

  The system supports easy **manual** onboarding of known devices

- 6. The user should be able to manually control the drone using a low-latency video stream that supports responsive user interaction.

  At times as mentioned in the req. 1 comment. The system can experience delays. These delays can affect user experience while flying the drone, temporarily making the user flying the drone blind to the drone view.

- 7. The user should with ease be able to navigate to specific timestamps where alerts were triggered. : X

  The system does not support this feature in its current state and cant therefore test this requirement, if we had more time we could have implemented it.

- 9. Communication between system components should be encrypted. : /

Due to the drone not supporting on-board programmability and encryption we cannot communicate with it with in a securely manner from the Base Station. However all other parts of the system support encrypted communication.

Chapter 9

Discussion

After implementation and verification of requirements, we would like to review some core elements of our process throughout this project and discuss potential improvements if we were to do it again.

In this chapter we will go over these 3 core phases of the project as well as some final remarks regarding the security:

- Research phase
- Design phase
- Implementation phase
- Security concerns

9.1 Research phase

Our research phase was not the best, but we still managed to find relevant pieces of work that could support our fundamental understanding of how we wanted to create the system to solve our problem statement.

In the state of the art we had a lot of focus on commercial drone surveillance systems and their top level architecture. This gave us a good insight into how our initial system architecture should look like, but did not give us much depth in terms of how the underlying information exchanges would function.

It is worth noting that much of the underlying software tech are build by the companies themselves, and therefore company privacy and secrecy hides some parts of their systems to keep them competitive on the marked. Since we lacked this knowledge from the initial research we relied more on open source solutions and our own ideas for our early progress in the implementation.

Researching relevant papers or other sources on multi-video streaming and management would probably have given us more insight earlier in the architecture and could potentially have saved us some time in the implementation phase.

9.2 Design phase

The design phase took root in the scenarios and requirements made based on the research. Our goal was to design a system that would offer those required features stated in the requirements (Chapter 5).

Our initial architecture was also greatly inspired by the other commercial drone setup. Our integration of a Base Station as a drone controller and manager was not only inspired from the Nightingale solution from the drone security systems (section 4.2.2), but was also a necessary component as our drone relies on a WiFi connection to a control entity because it is not on board programmable.

Other larger scale companies that offer a solution like ours also have greater economic capabilities. Enough to be able to design their own drone hardware. This gives them a great edge in their system design in making the system flexible, fast and reliable. Some of these features for the drone include LTE communication and on board programmability. A capability we did not have during this project. This led to some solutions around the drone needing to be custom made to fit the drone at our disposal and could have possibly been made in a better way if we had more influence on the drone hardware capabilities.

Example: If our drone supported LTE and onboard programmability, it could potentially replace the base station with a communication module, with all data handling performed directly on the drone.

In this architecture, the communication module replaces the base station, but a communication module could serve far more drones since the increased range of the communication protocol instead of WiFi to a Base Station. Additionally, the drones themselves would handle the understanding of the incoming control commands and video streaming to the communication module, making the system scale better with more drones as they do not rely on another entity to process their controls.

9.3 Implementation phase

In the implementation phase, we learned a lot about video streaming and the internet protocol webrtc. We had big hopes for the final product, but ultimately due to having a small group working on the project we had to settle for what we have achieved so far.

Our system ended up fulfilling most of the requirements set in the design phase. The only requirements it falls short of are either not possible with the current setup and hardware, or were not prioritized since we have limited time and manpower.

When we began development, we had only a general sense of the technologies we might need. Many of the tools and frameworks we ultimately adopted were selected as the project evolved, based on experimentation and what proved to work best in practice.

Publish/Subscribe Technology

At one point in the development we tried using Kafka for the video stream between nodes. This was done by capturing the individual frames, converting them to jpegs, and then publishing them on a topic for consumption elsewhere.

The idea behind this was that if the video streams were directly posted on Kafka topics, it would be easier for other services to consume them (Evaluation,recordings and Camera Streams). However, this was not feasible due to Kafka under performing for this use case, when two or more streams were active at the same time, publishing to the same Kafka server. We suspect that Kafka has limited throughput or that some specific configuration was needed, but we did not investigate further, focusing our efforts on more meaningful tasks.

We later returned to the idea of having video streams using publish/subscribe distributing the streams from the Proxy Service to their consumers. For this purpose Redis was used. When we came back to this idea the Video Camera logic to stream video to the Cloud had already been implemented and we concluded it was preferable to keep it as it was, instead of adding further entropy. It, however, could simplify the implementation to replace the WebRTC stream from the Video Cameras with Redis channels, since then the WebRTC module of the Security Cameras could be removed altogether.

Cloud

Currently, all the services are run by a single docker compose, making it easily deployable, but at the same time, they all must run on a single machine which violates the principles of the micro-services architecture. We would like to have each service executing on their own Virtual Machine, which could be achieved by replacing the current Docker compose setup to an Infrastructure as Code one, where a full specification of Cloud formation would be added. This does not simply break the Micro-Services principles, but also adds a bottleneck in the system. In our case trying to run the Cloud Module in a t3-micro instance in AWS Cloud (Instance available in the Free Tier) resulted in poor performance, as one could barely watch a fluid video stream, as these instances only possess 2GB of RAM.

The Threat Detector is where the biggest overhead of our system is located. Currently, each stream is evaluated using YOLO in a different process, but they all must run on the same machine. The system should allow for a refactor of this component in the future, so it uses Kubernetes for load balancing. The idea would be to have the control plane observe which streams are active and for every active stream start a new pod where the detection would be done. This would improve performance and scalability of this module.

9.4 Security Concerns

Drone

The primary security concern in our system involves the drone setup. Initially, we believed the drone was capable of reconfiguration to connect to a different access point. In this case, there was only a need to connect to the insecure drone network once and change its access point. However, we later discovered our drone version did not support this functionality. On the positive side, it is possible to configure the drone network to require authentication, adding a layer of risk mitigation to the process.

Base Station

Another possible security issue we identified ties itself with the fact that the Cloud should identify if a connecting Base Station has previously been connected. In this case, the configuration already stored should be used. The system checks this by using the Base Station name, since it is the only thing (together with the password) that it sends for connection. If a Base Station with that name already exists, then the password is compared with the stored one. This in theory means that if a Base Station went offline, then a bad actor could hijack its connection causing a Denial of Service.

We evaluated this risk as low. It is a major issue for a Base Station to suffer a DoS attack, since the whole surveillance system of an area would be compromised. However, for it to be possible, an attacker would have not only to guess the name and password of the Base Station, but also to get access to the generated certificates, which we deemed very unlikely.

Chapter 10

Conclusion

In the Conclusion, we present our final remarks about the project and the product we developed. As the final stage of our development process, we verify the system by revisiting our initial problem formulation and reflecting on how our results address it.

We begin by addressing each sub-problem using the experience and insights gained throughout this project. These answers then inform our overall evaluation of how effectively we satisfied the main problem statement.

Subproblem 1:

> ***How can we develop a framework for video streaming of devices to a single cloud backend that is able to expose these streams to a user?***

Our solution provides a containerized application able to serve the user with a live video feed, as well as store past recordings and present them to the user. This is done using a combination of technologies. This is handled with these technologies:

- gRPC
  Used for exchanging configuration information about the webRTC video stream (offer, answer)
- WebRTC
  Used as the main protocol for transporting the video from Base Station to Cloud
- Redis
  Used for handling incoming video streams and making them available for other services on the clouds reach.

Subproblem 2:

> ***How can we develop a system that allows for remote connection, video retrieval and control of a drone?***

Our solution provides a way to take control of a DJI Tello drone connected via a Base Station. The Base Station acts as an external controller to the drone and handles the connection and video stream from the drone to the cloud. As mentioned in the video streaming above, we use gRPC for the WebRTC connection exchange including offer and answer, and with that establish a WebRTC connection from the Base Station to the users front-end.

Users can send control commands to the drone using a WebRTC data channel in the same connection as the video stream. The Drone controller application running on the Base Station is able to fetch video from the drone and stream it to the user as well as convert movement commands and send them to the drone, achieving remote control.

Subproblem 3:

> ***How can we set up autonomous threat detection in the system that alerts the user when a threat is detected?***

Our solution does not fully show how to achieve this. However, the system shows how to apply an object detection model on an incoming video stream and from that catch if the object detection model detects something that we would define as a "threat". The object detection model consumes this incoming stream from the redis, the same way the video recorder and live feed does.

The remainder of this problem lies in how to alert the user. Many options are available but we believe that most of the work has been done, and only smaller portion of this problem is still not realized.

Subproblem 4:

> ***How can we securely manage these connection and manage access control in the system to minimize risk of exploitation***

Our solution implements end-to-end encryption and role based user access throughout the system. This is done using RSA keys in the cloud when accessing the Web App and when the Flask app calls the proxy servers' gRPC functions. This ensures that all data exchanged is encrypted and not prone to man-in-the-middle attacks. Our role based user access ensures that users are presented with exactly the information and limited resources only they are allowed to consume. The security of the system has one pitfall which is that communication from Base

Station to drone is not encrypted. If a malicious user was close and could access the drone hotspot he would be able to intercept and possibly hijack the drone. As discussed in the limitations (5.1), the DJI tello drone does not support encryption nor on board programmability, and we are therefore unable to realize this feature in this project. Had we had another drone that supported either of the mentioned features, we could have realized this as well.

Main problem formulation

> ***How to develop a system that provides a portable and independent framework for surveying areas using commercial drones and cameras?***

To finalize, our solution provides users with a system that supports the majority of the sub-problem statements we defined. We provide a portable system in that we have containerized most modules making most of the application platform independent. We have realized how to handle streaming of multiple cameras for live video feed, video recording and playback features, as well as a video evaluation in the form of an object detection model. We implemented a drone system that can be remotely controlled through the user's front-end application, providing live video feed.

Ultimately, we delivered a flexible system that can be extended to support a wide range of cameras and drones, providing users with a new practical option for surveillance using commercially available devices.

Bibliography

[1] . *Height Technologies SAMS*. Accessed: 2025-11-01. URL: https://heighttechnologies.com/autonomous-security-drones/.

[2] . *Nando Drone*. Accessed: 2025-11-01. URL: https://www.nando-drone.com/product/.

[3] . *Securiton SecuriDrone*. Accessed: 2025-11-01. URL: https://partner.securiton.de/drone-security.

[4] Asylon Robotics. *ASYLON; YOUR SECURITY'S BRAIN AND BACKBONE*. Accessed: 2025-11-10. 2025. URL: https://asylonrobotics.com/solutions/#:~:text=YOUR%20SECURITY%27S%20BRAIN%20AND%20BACKBONE,analytics%20to%20inform%20strategic%20decisions.

[5] Asylon Robotics. *GUARDIAN™ ABOVE THE THREAT. ALWAYS READY.* Accessed: 2025-11-10. 2025. URL: https://asylonrobotics.com/solutions/guardiandrone.

[6] Certrec. *Certrec's Alliance With Aslylon*. Accessed: 2025-11-10. 2025. URL: https://www.certrec.com/alliances/asylon.

[7] Dennis Crowley. *ASYLON Awarded* $12M + AFWERX STRATFI SBIR Contract for United States Air Force Global Strike C$ Accessed: 2025-11-10. 2025. URL: https://asylonrobotics.com/resources/blog/afwerx-stratfi-contract.

[8] Ziad Hunaiti Khalifa AL-Dosari and Wamadeva Balachandran. "Drone Safety and Security Surveillance System". In: *Applied Science* (2024).

[9] Byung Moo Lee et al. "An easy network onboarding scheme for internet of things networks". In: *IEEE Access* 7 (2019), 8763–8772. DOI: 10.1109/access.2018.2890072.

[10] Minzhao Lyu et al. "Network anatomy and real-time measurement of Nvidia GeForce now cloud gaming". In: *Lecture Notes in Computer Science* (2024), 61–91. DOI: 10.1007/978-3-031-56249-5_3.

[11] Nightingale Security. *About Nightingale Security*. Accessed: 2025-10-08. 2025. URL: https://www.nightingalesecurity.com.

[12] Nightingale Security. *About Nightingale Security specs*. Accessed: 2025-10-10. 2025. URL: https://www.nightingalesecurity.com/specs-faqs.

[13] Nightingale Security. *Nightingale Security Operations Manual*. Accessed: 2025-10-22. 2025. URL: https://www.casmarglobal.com/media/akeneo_connector/media_files/D/S/DS_BLACKBIRDSTARTER_KIT_e994.pdf.

[14] R. Shea et al. “Cloud gaming: Architecture and performance”. In: *IEEE Network* 27.4 (2013), 16–21. DOI: 10.1109/mnet.2013.6574660.

[15] Stephanie Susnjara, Ian Smalley. *Software testing, defined*. Accessed: 2025-11-01. URL: https://www.ibm.com/think/topics/software-testing#2045551722.

[16] Abid Khan Madiha Haider Syed Syed Atif Moqurrab Gautam Srivastava Syeda Mahnoor Gilani Adeel Anjum. “A robust Internet of Drones security surveillance communication network based on IOTA”. In: *ScienceDirect* (2024).