



COMSATS University Islamabad (CUI)

**Software Design Description
(SDS DOCUMENT)**

for

Mihawk

Version 1.0

By

Hammad ur Rehman CIIT/FA21-BCS-055/ISB

Hozefa Rauf CIIT/FA21-BCS-057/ISB

Usman Malik CIIT/FA21-BCS-072/ISB

Supervisor

Mr. Qasim Malik

Bachelor of Science in Computer Science (2021-2025)

Table of Contents

Revision History	iii
Application Evaluation History	iv
1. Introduction.....	1
1.3 Modules	1
1.4 Overview	4
2. Design Methodology and Software Process Model	4
3. System Overview	5
3.1 Architectural Design	6
4. Design Models.....	9
4.1 Activity Diagram	9
4.2 Data Flow Diagram (DFD)	20
4.3 State Transition Diagram	23
5. Data Design.....	26
5.1 Data Dictionary	26
5.2 JSON Schema.....	27
6. Human Interface Design	28
6.1 Screen Images.....	29
6.2 Screen Objects and Actions.....	33
7. Implementation	34
7.1 Algorithm.....	34
7.2 External APIs/SDKs	41
7.3 User Interface	41
7.4 Deployment	46
8. Testing and Evaluation	46
8.1 Unit Testing.....	46
8.2 Functional Testing	50
8.3 Business Rules Testing	55
8.4 Integration Testing.....	56
9. Plagiarism Report	59

Revision History

Name	Date	Reason for changes	Version

Application Evaluation History

Comments (by committee) *include the ones given at scope time both in doc and presentation	Action Taken

Supervised by

Signature_____

1. Introduction

1.1 Purpose

The purpose of the Mihawk project is to address the limitations of traditional surveillance systems, which often suffer from restricted coverage, delayed threat detection, and increased false alarms. By introducing a manually driven drone-based surveillance system, we aim to enhance the efficiency and effectiveness of security operations. Additionally, the integration of a database ensures the integrity and security of the data collected, making it readily accessible for analysis. The ultimate goal is to provide law enforcement agencies, security personnel, and organizations with a tool that offers improved surveillance capabilities, leading to quicker response times, fewer false alarms, and better protection of assets and people.

1.2 Scope

The aim of the Mihawk project is to create a cutting-edge drone surveillance system that addresses the challenges of modern security operations. The system will consist of several integrated modules, each designed to enhance surveillance efficiency and reliability. The manual flight control module enables users to navigate drones along the defined routes for precise surveillance coverage. The real-time video streaming module provides continuous live surveillance, allowing users to monitor activities as they happen. The advanced threat detection module will utilize sophisticated algorithms to identify potential security risks and respond promptly. A database will be integrated for secure and reliable data storage, ensuring the integrity and confidentiality of flagged surveillance data. Additionally, user management features will allow administrators to control access and permissions, while customizable surveillance settings will provide flexibility in managing different monitoring scenarios. Tools for data analysis and reporting will enable users to gain insights from surveillance data and optimize security operations. This platform aims to offer a scalable, efficient, and comprehensive solution that enhances situational awareness and streamlines operational workflows in various security environments.

1.3 Modules

1.3.1 Module 1: Drone Flight System

FE-1: Real-time monitoring of drone status and location.

FE-2: Integrate weather data APIs to adjust flight plans based on weather conditions.

FE-3: Implement flight assist features to optimize drone navigation

FE-4: Provide controls for emergency procedures in case of unexpected situations during flight.

1.3.2 Module 2: Surveillance and Threat Detection

FE-1: Implement real-time video processing algorithms to analyze surveillance footage for potential threats or anomalies.

FE-2: Develop object detection algorithms to identify and classify objects of interest in the surveillance feed.

FE-3: Design threat identification algorithms to detect and categorize security threats based on detected objects or behaviors.

FE-4: Develop algorithms for crowd behavior analysis to detect suspicious activities in crowded areas.

FE-5: Customizable threat detection settings for specific security requirements and environments.

1.3.3 Module 3: Alert Management

FE-1: Real-time alerts and notifications for security incidents or unusual activities.

FE-2: Priority-based alert categorization and escalation procedures for timely response.

FE-3: Integration with existing security systems for seamless alert management and coordination.

1.3.4 Module 4: User Management

FE-1: User authentication and authorization mechanisms for secure access to the system.

FE-2: Role-based access control to restrict functionalities based on user roles and permissions.

FE-3: Enable users to edit profile information (password).

FE-4: Implement a secure password reset functionality where user can retrieve forgotten passwords through email verification or security questions.

FE-5: Audit trail functionality to track user activities and changes made to the system.

FE-6: Implement 2-Factor Authentication for better security.

1.3.5 Module 5: Surveillance Monitoring Interface

FE-1: Design a user-friendly web interface for operators to monitor and control the surveillance system.

FE-2: Develop features to display live surveillance feed and playback recorded footage on the web application.

FE-3: Implement real-time status indicators for the network, battery levels, and connection status.

1.3.6 Module 6: Interactive Mapping and Location Visualization

FE-1: Interactive maps and visualization tools for displaying drone locations and surveillance data.

FE-2: Provide search functionality for users to selectively view specific types of surveillance data or events.

FE-3: Implement zoom and pan functionality for detailed exploration of surveillance data and maps.

FE-4: Integrate real-time weather data overlays to visualize weather conditions and their impact on surveillance operations.

FE-5: Implement dynamic data filtering options to enable users to filter surveillance data by criteria like time, location, or activity type for focused analysis and monitoring.

1.3.7 Module 7: Data Handling

FE-1: Develop mechanisms to efficiently store captured images or videos on the database.

FE-2: Implement functionality to retrieve stored surveillance data from the database for analysis or playback.

FE-3: Implement data compression techniques to reduce storage requirements without compromising quality.

FE-4: Integrate data lifecycle management policies to automatically archive or delete outdated surveillance data.

1.3.8 Module 8: Reporting and Analysis

FE-1: Reporting functionalities to generate detailed reports on surveillance activities and incidents.

FE-2: Advanced analytics tools for deep dive analysis of surveillance data and trends.

FE-3: Export functionalities to save reports and analytics data in various formats for sharing and archival purposes.

1.3.9 Module 9: Admin Dashboard

FE-1: Design a user-friendly interface for administrators to monitor and manage the surveillance system efficiently.

FE-2: Enable administrators to manage user accounts, including creating, editing, and deleting user profiles, as well as assigning roles and permissions.

FE-3: Provide a comprehensive overview of system status, including drone fleet health, surveillance coverage, and alert summaries.

FE-4: Allow administrators to configure system settings, such as alert thresholds, and data retention policies.

FE-5: Design reporting tools to generate insights on system performance, alert trends, and operational metrics for analysis.

1.3.10 Module 10: Raspberry Pi Integration

FE-1: Configure Raspberry Pi to stream real-time video from the drone's camera using the Real-Time Streaming Protocol (RTSP). This allows for live monitoring of surveillance feeds over the network.

FE-2: Enable integration of various camera modules and sensors with the Raspberry Pi, providing flexibility for different surveillance and data collection needs.

FE-3: Implement data compression techniques on the Raspberry Pi to ensure efficient transmission of high-quality video streams with minimal bandwidth usage.

FE-4: Integrate power management features to monitor and optimize the Raspberry Pi's energy consumption, ensuring it operates efficiently during drone flights.

FE-5: Use Raspberry Pi for real-time processing of surveillance data (e.g., object detection) to reduce latency before data is sent to the central system, improving threat detection capabilities.

1.4 Overview

The remainder of SDS document provides a detailed specification of Mihawk, outlining its functional and non-functional requirements. The document is organized into distinct modules, each focusing on a specific aspect of the software's functionality. This is followed by a section on non-functional requirements, encompassing aspects such as performance, security, usability, and maintainability. By following this modular structure, the document ensures clarity, organization, and ease of understanding for all stakeholders involved in the development process

2. Design Methodology and Software Process Model

2.1 Design Methodology

The design methodology we will be using is the **Procedural Approach**. Procedural programming is ideal for Mihawk as it promotes simplicity, clarity, and efficiency, which are essential for a system that integrates various modules such as Drone Control, Threat Detection, and User Management. This methodology allows the team to break down the system into a series of step-by-step procedures (functions) that perform specific tasks, making it easier to understand and implement the system's functionalities. Additionally, procedural programming enables straightforward collaboration among team members, with clear and easy-to-follow function calls and data management. We will utilize technologies such as ReactJS for the front-end and NodeJS for the back-end, where we will focus on modular functions that handle the core operations of each component. Furthermore, the design will be communicated through flowcharts and function call diagrams, ensuring clarity and precision during the development phase.

2.2 Software Process Model

The software process methodology we will use is the Incremental Process Model. The Mihawk

project is composed of multiple well-defined modules, and most of the system requirements have been identified at the start of the project. This process model allows us to develop and deliver the application in incremental builds, focusing on delivering functional modules such as Real-Time Video Streaming and User Management in early iterations. Any unforeseen changes or enhancements can be accommodated in subsequent increments without affecting the previously developed components. The incremental process model ensures that we can achieve early system functionality, incorporate user feedback iteratively, and maintain flexibility during development, making it the best-suited model for our project.

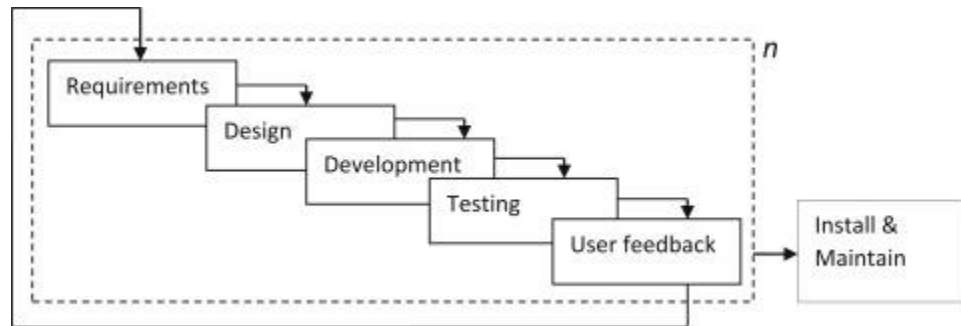


Figure Modified Incremental Model

3. System Overview

The **Mihawk** system consists of 10 modules, focused on enhancing drone surveillance with features like manual flight control, real-time video streaming, and threat detection. It integrates user feedback and alerts to improve system performance and security.



Figure : System Overview

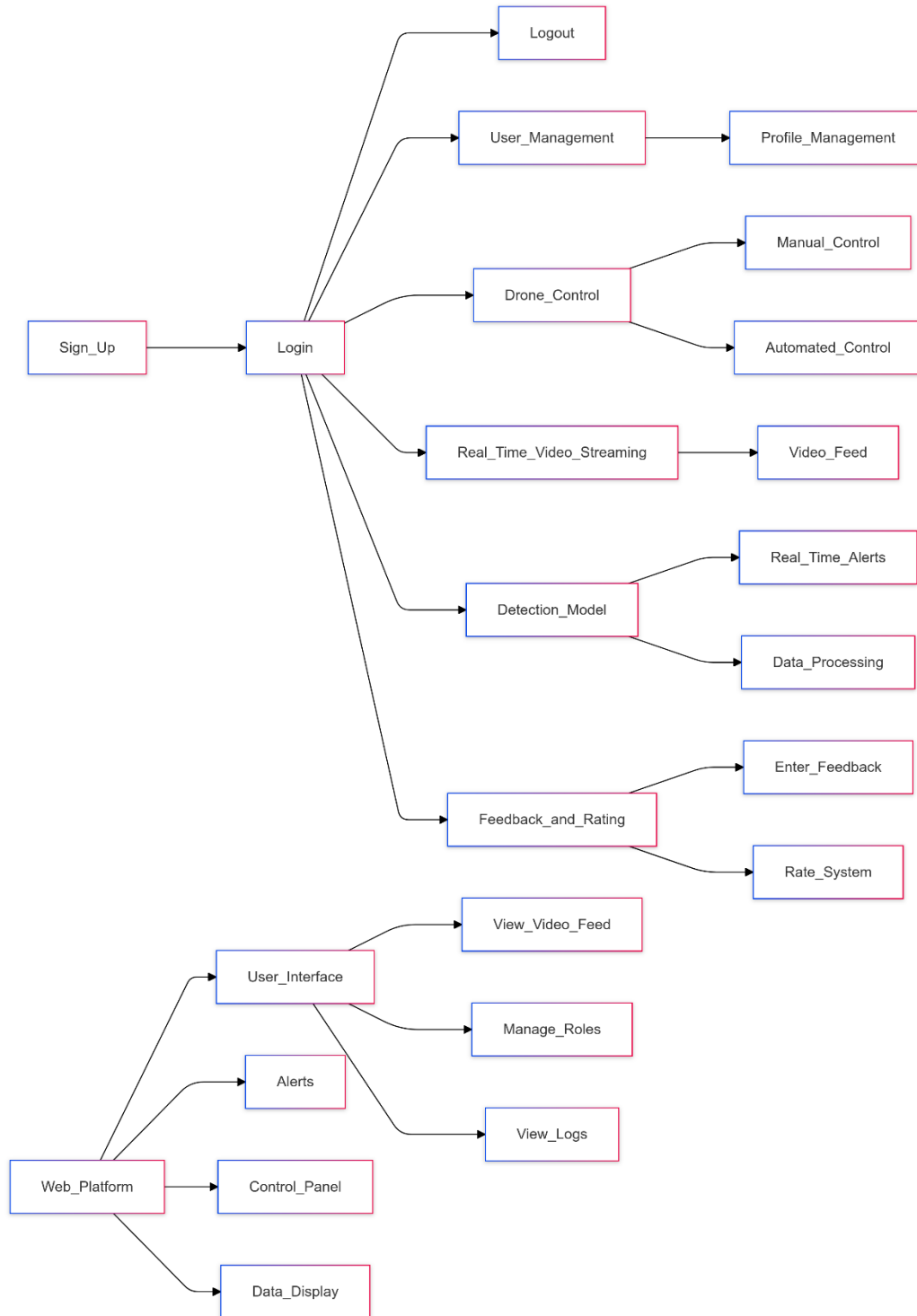
3.1 Architectural Design

The Mihawk system comprises 10 modules, each implemented as modular, object-oriented components. The **User** module forms the core, with other modules such as Drone Control, Threat Detection, and Alert Management tightly integrated for seamless operation.

The **Admin Dashboard** manages user roles and system configurations, while the **Real-Time Video Streaming** and **Surveillance Monitoring Interface** enable live monitoring and control. The **Interactive Mapping and Location Visualization** module provides situational awareness through dynamic mapping tools.

The **Threat Detection** module employs machine learning to identify security risks, with insights and reports generated by the **Reporting and Analysis** module. Surveillance data is securely managed in the **Data Handling** module using a robust database.

This architecture ensures scalability, efficiency, and adaptability to meet evolving security needs.

**Figure : Block Line Diagram for Mihawk**

4. Design Models

Create design models as are applicable to your system. Provide detailed descriptions with each of the models that you add. Also ensure visibility of all diagrams.

4.1 Activity Diagram

Module 1: Drone Flight System

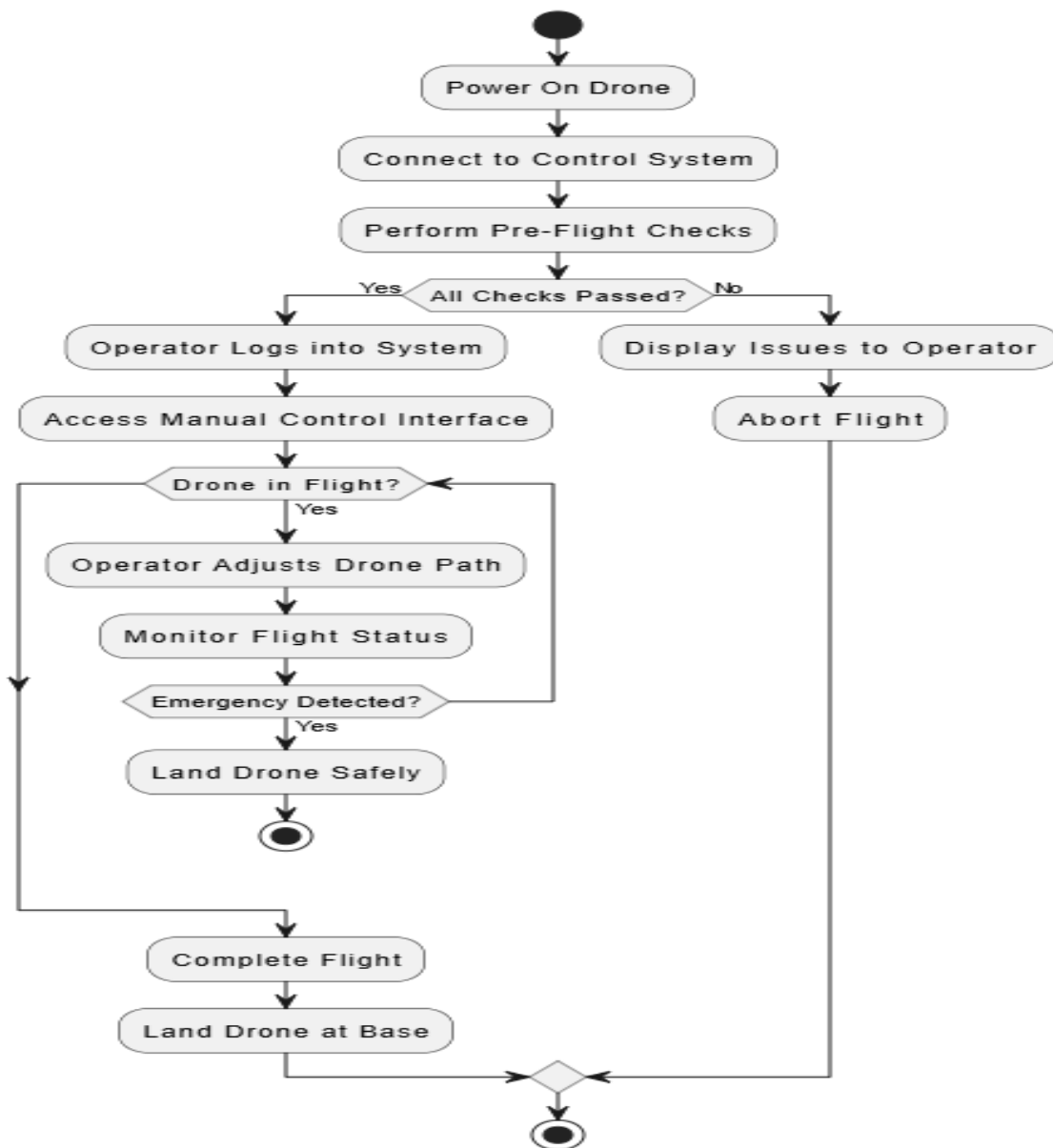


Figure : Drone Flight System

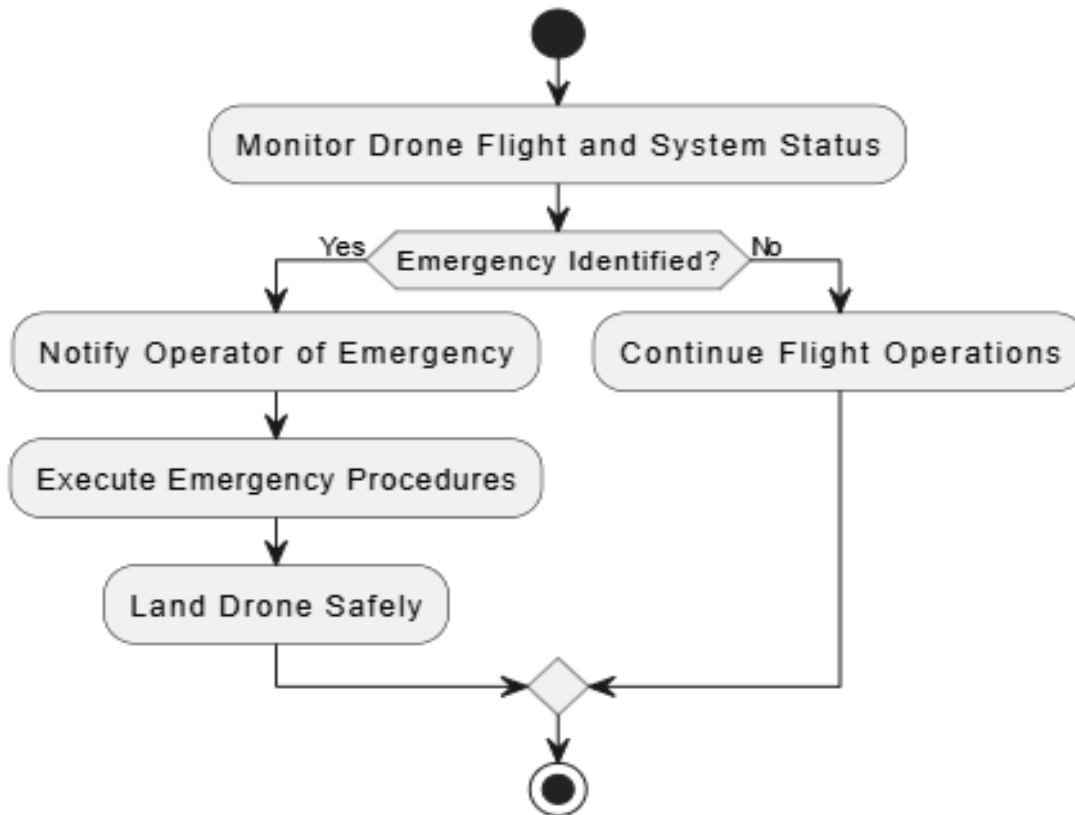
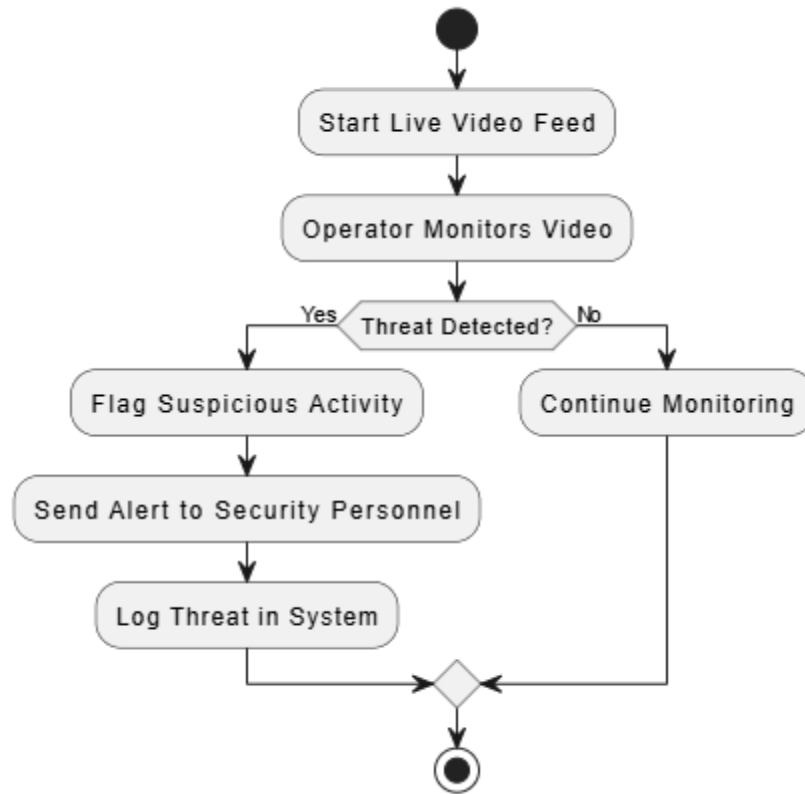
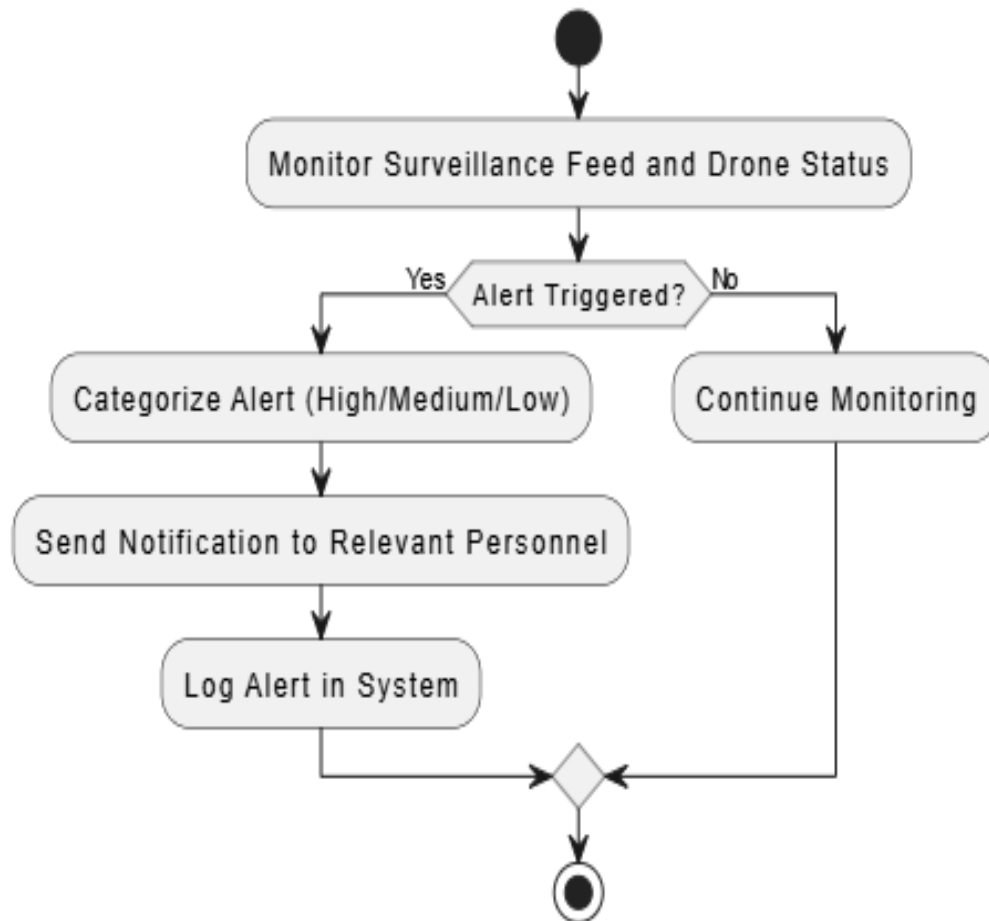
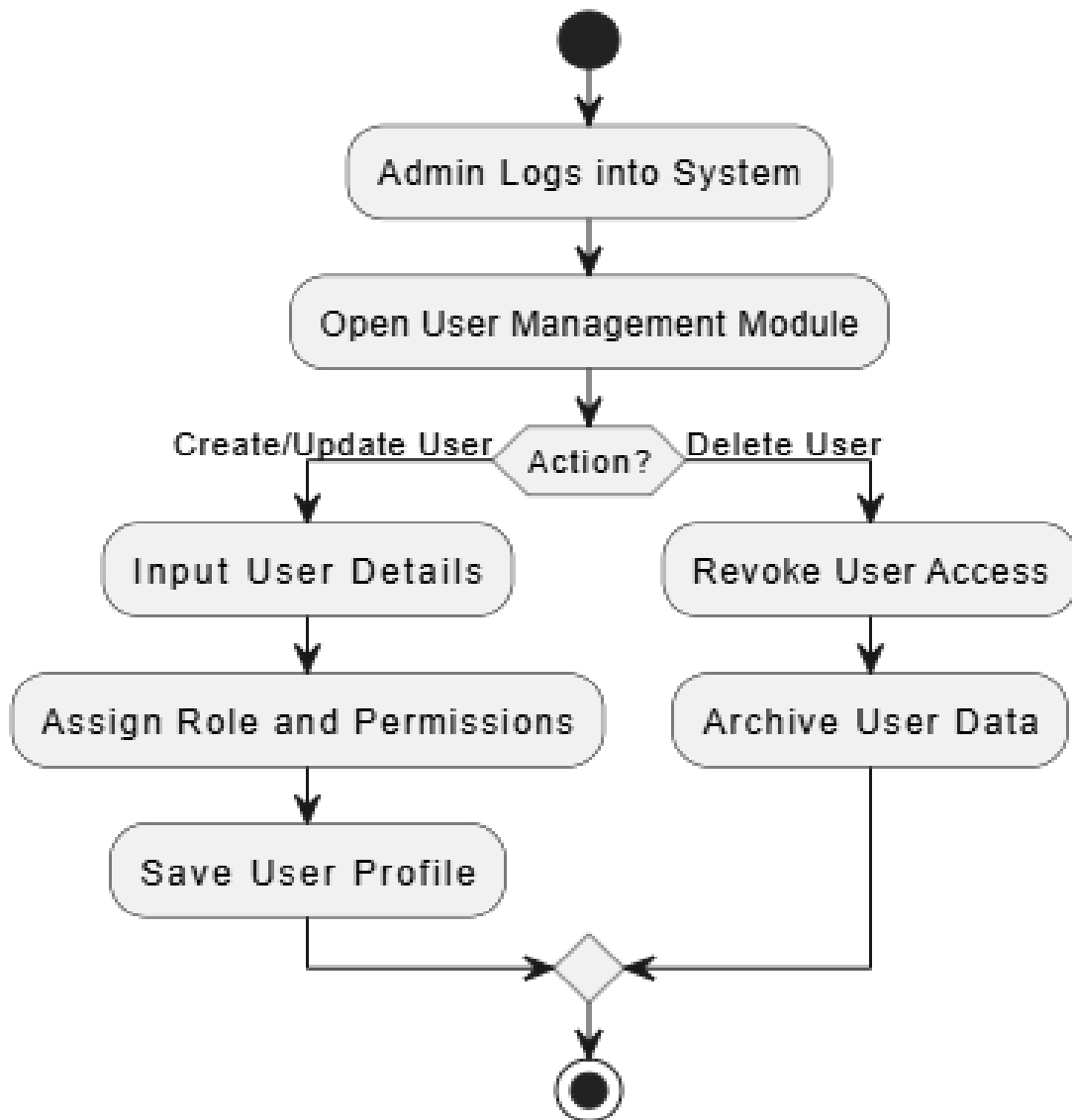
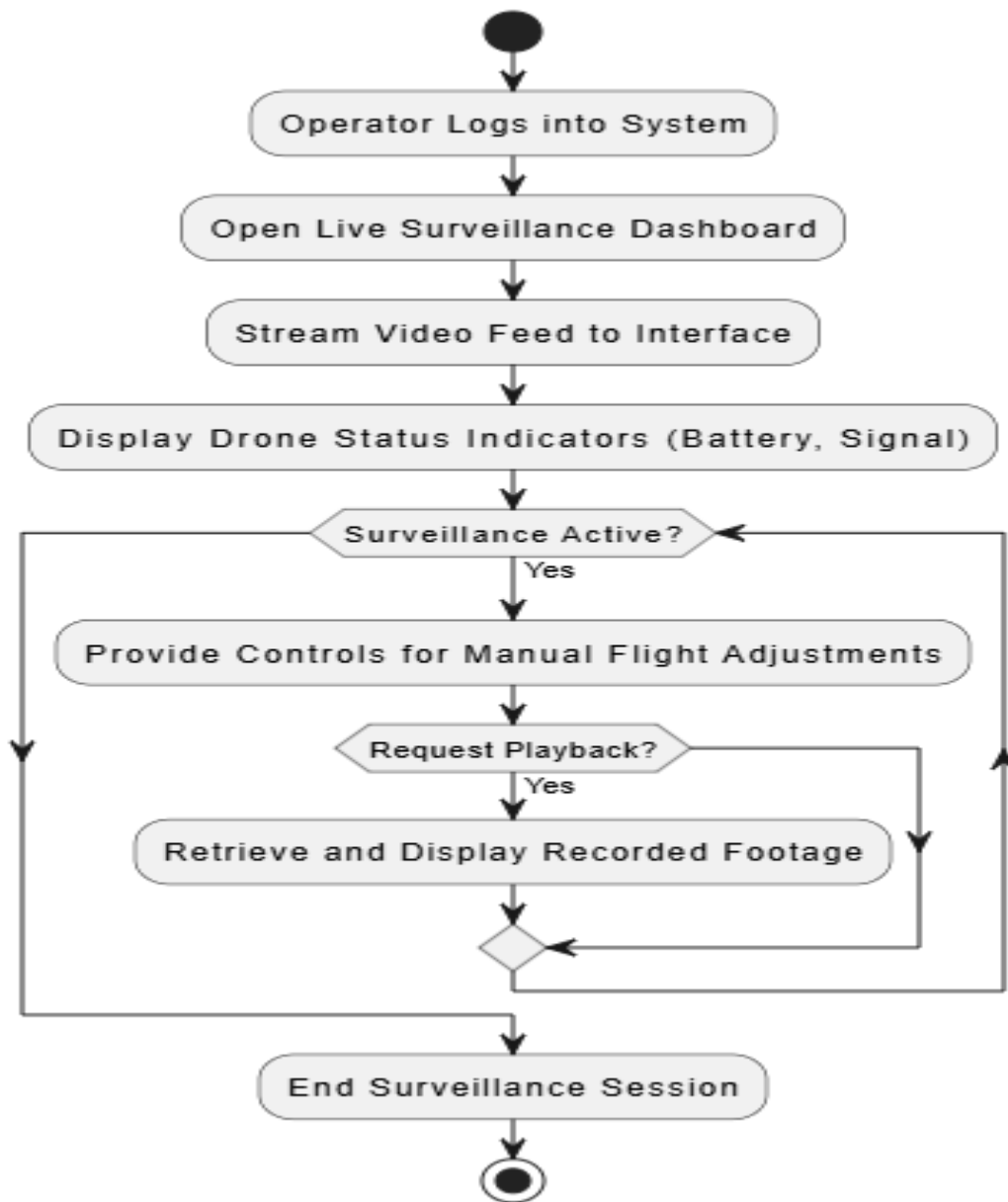


Figure : Emergency landing

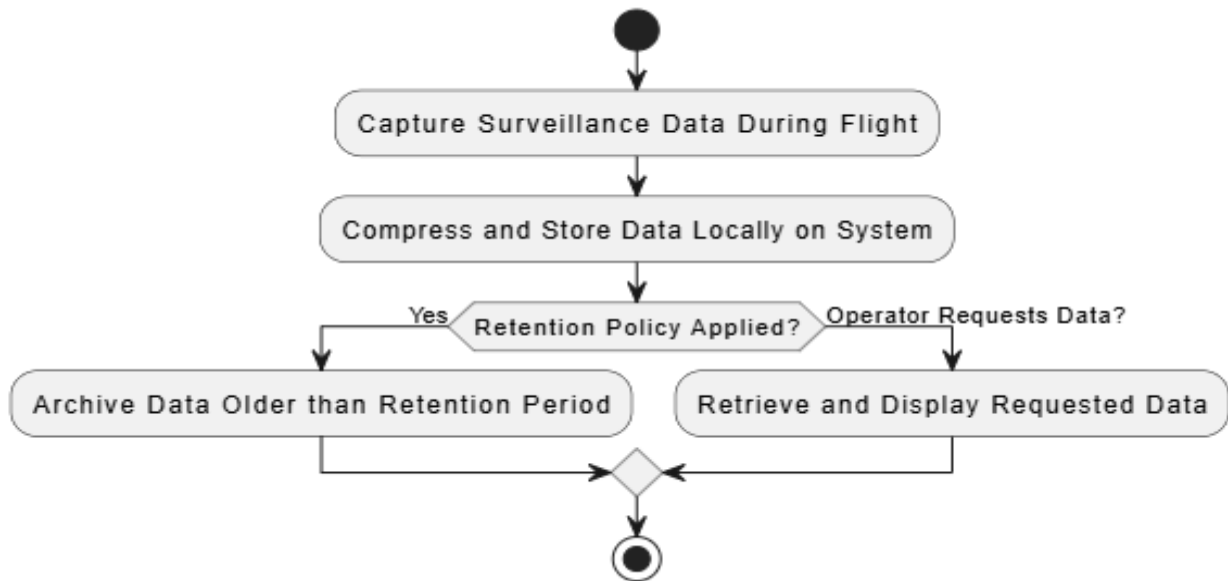
Module 2: Surveillance and Threat Detection**Figure : Surveillance and Threat Detection**

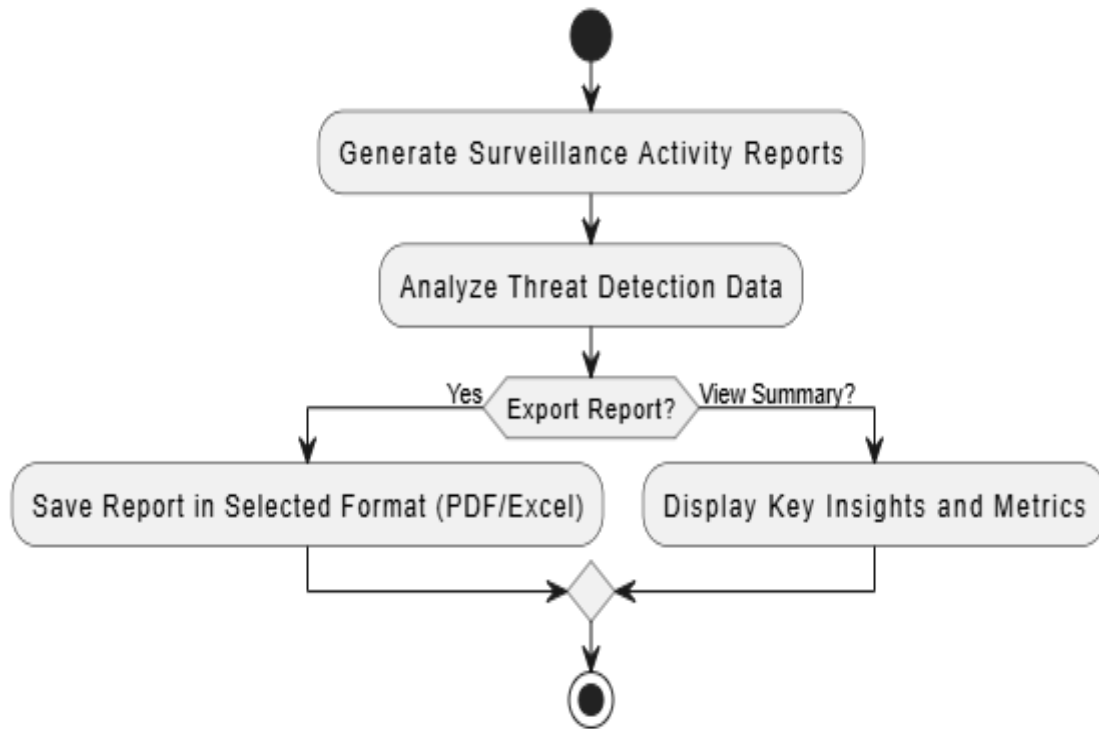
Module 3: Alert Management**Figure : Alert Management**

Module 4: User Management**Figure : User Management**

Module 5: Surveillance Monitoring Interface**Figure : Surveillance Monitoring Interface**

Module 6: Interactive Mapping and Location Visualization**Figure : Interactive Mapping and Location Visualization**

Module 7: Data Handling**Figure : Data Handling**

Module 8: Reporting and Analysis**Figure : Reporting and Analysis**

Module 9: Admin Dashboard

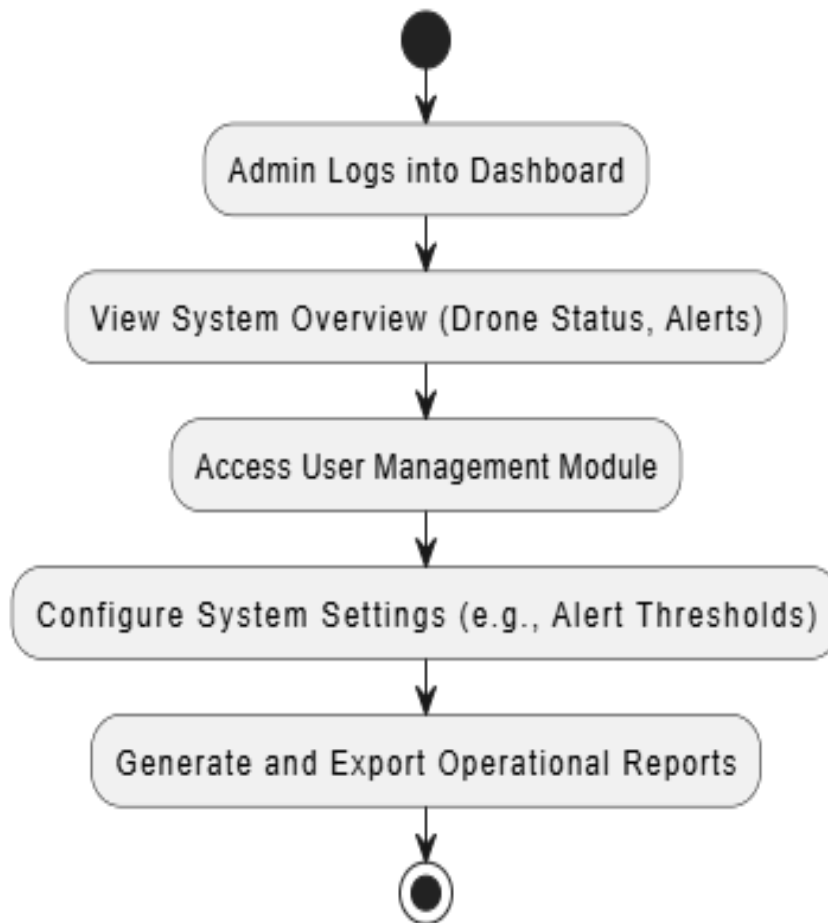
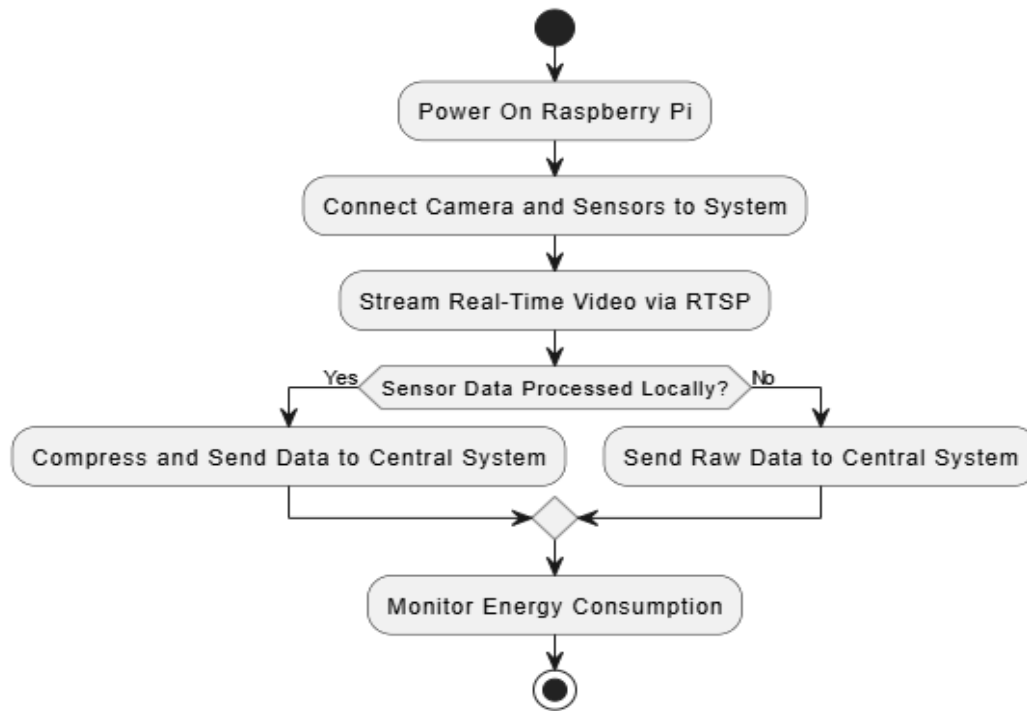


Figure : Admin Dashboard

Module 10: RaspberryPI Integration**Figure : Raspberry Pi Integration**

4.2 Data Flow Diagram (DFD)

4.3.1 DFD Level 0:

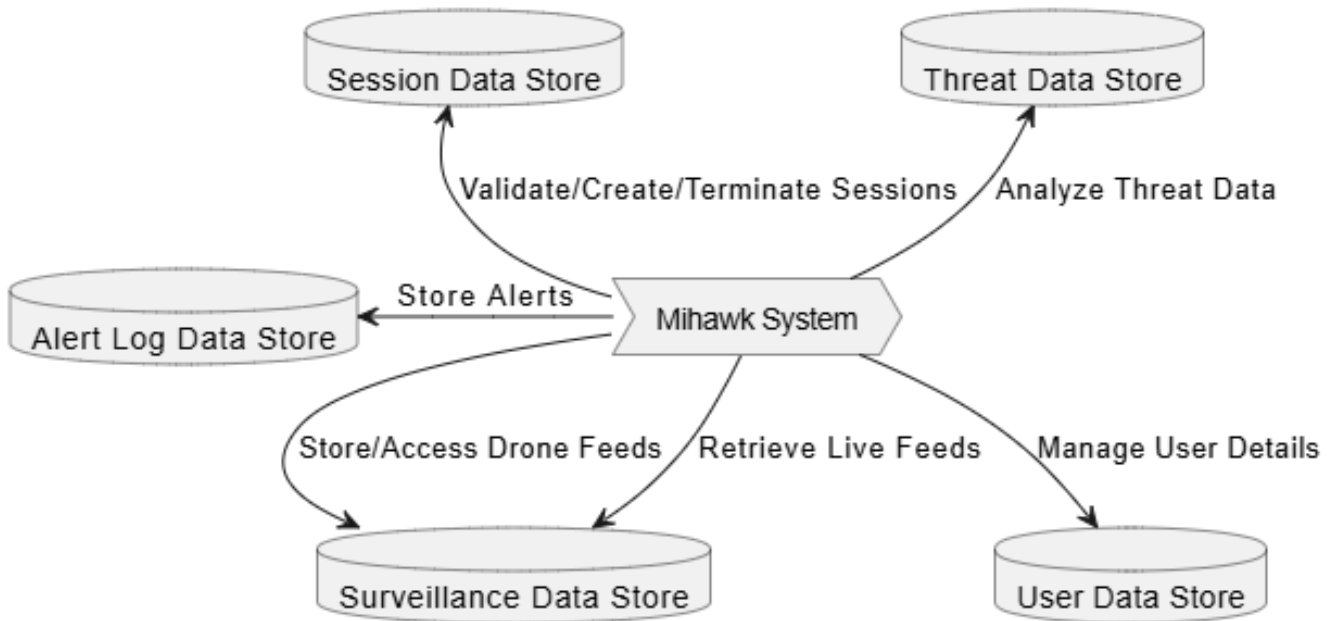


Figure : Context Diagram

4.3.2 DFD Level 1:

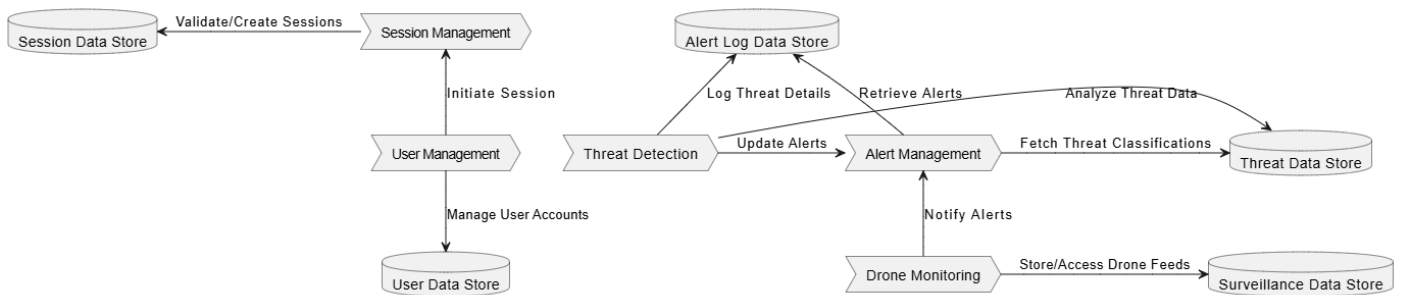


Figure : System Processes

4.3.3 DFD Level 2:

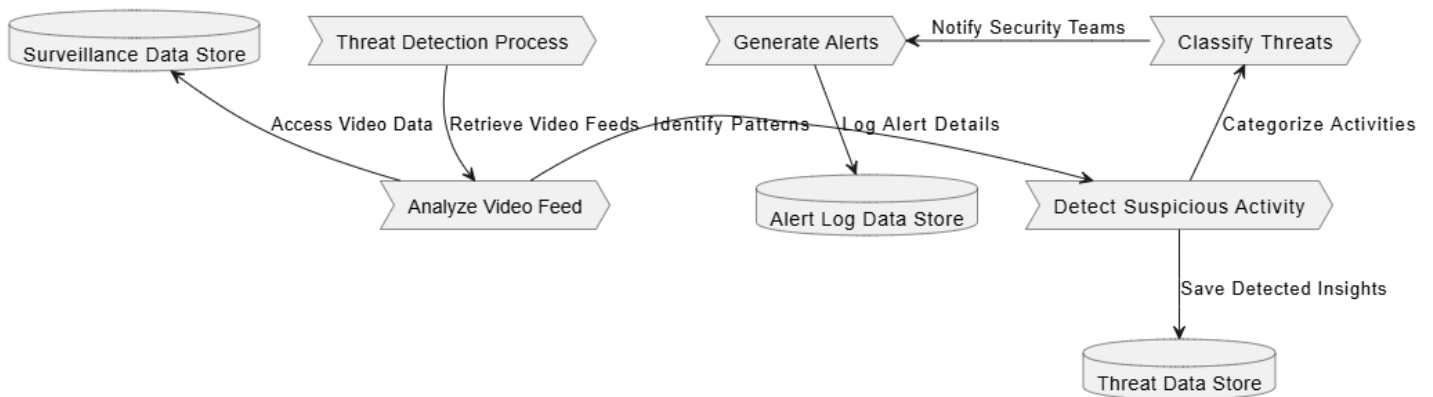


Figure : DFD Threat Detection Process

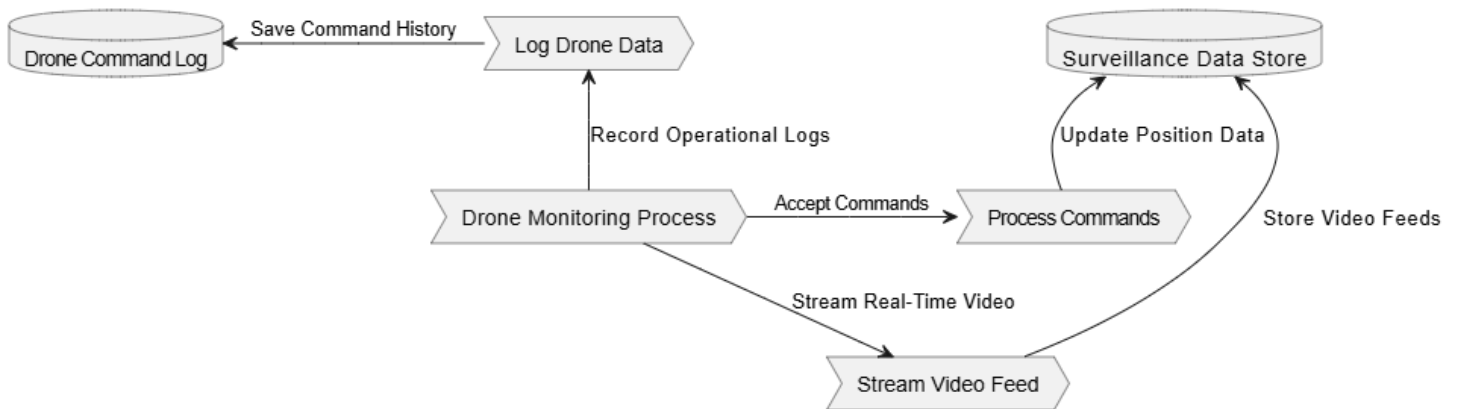


Figure : DFD Drone Monitoring Process

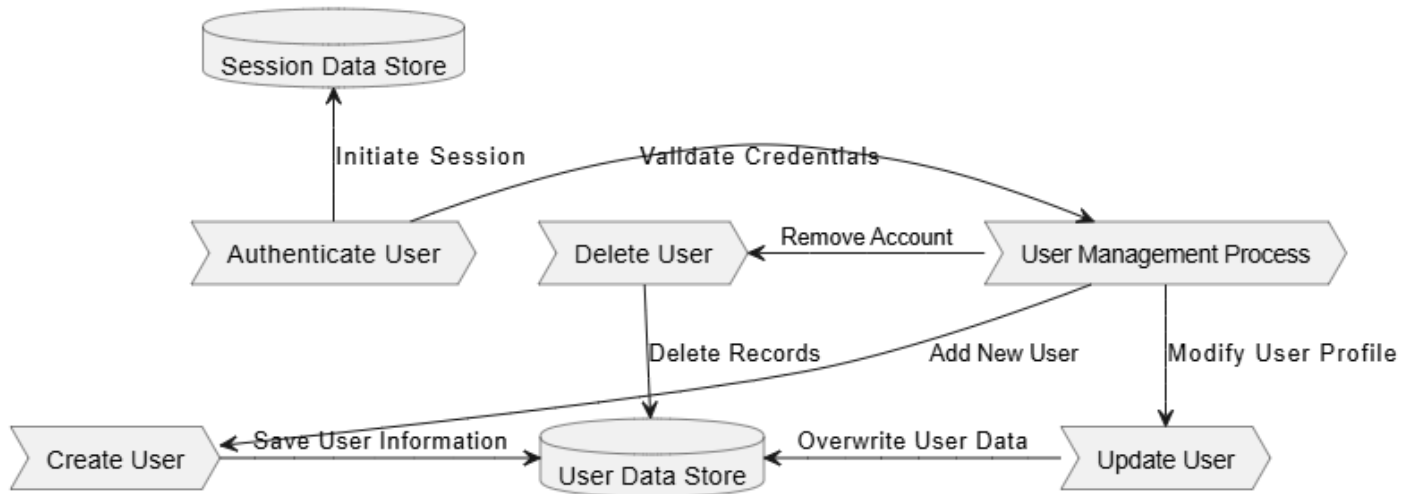


Figure : Detailed User Management Process

4.3 State Transition Diagram

4.3.1 Drone State:

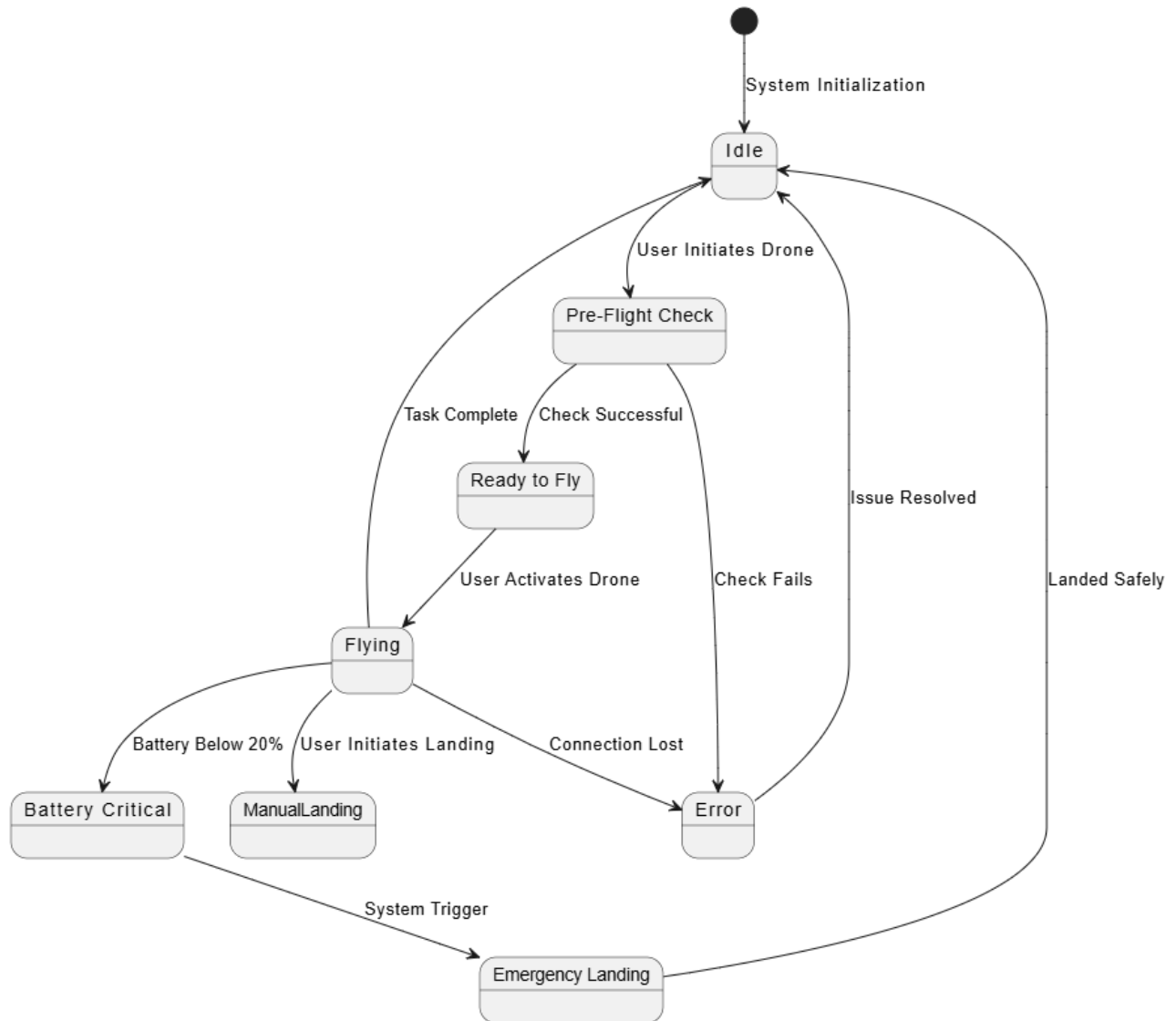


Figure : Modified Incremental Model

4.3.2 Alert Lifecycle

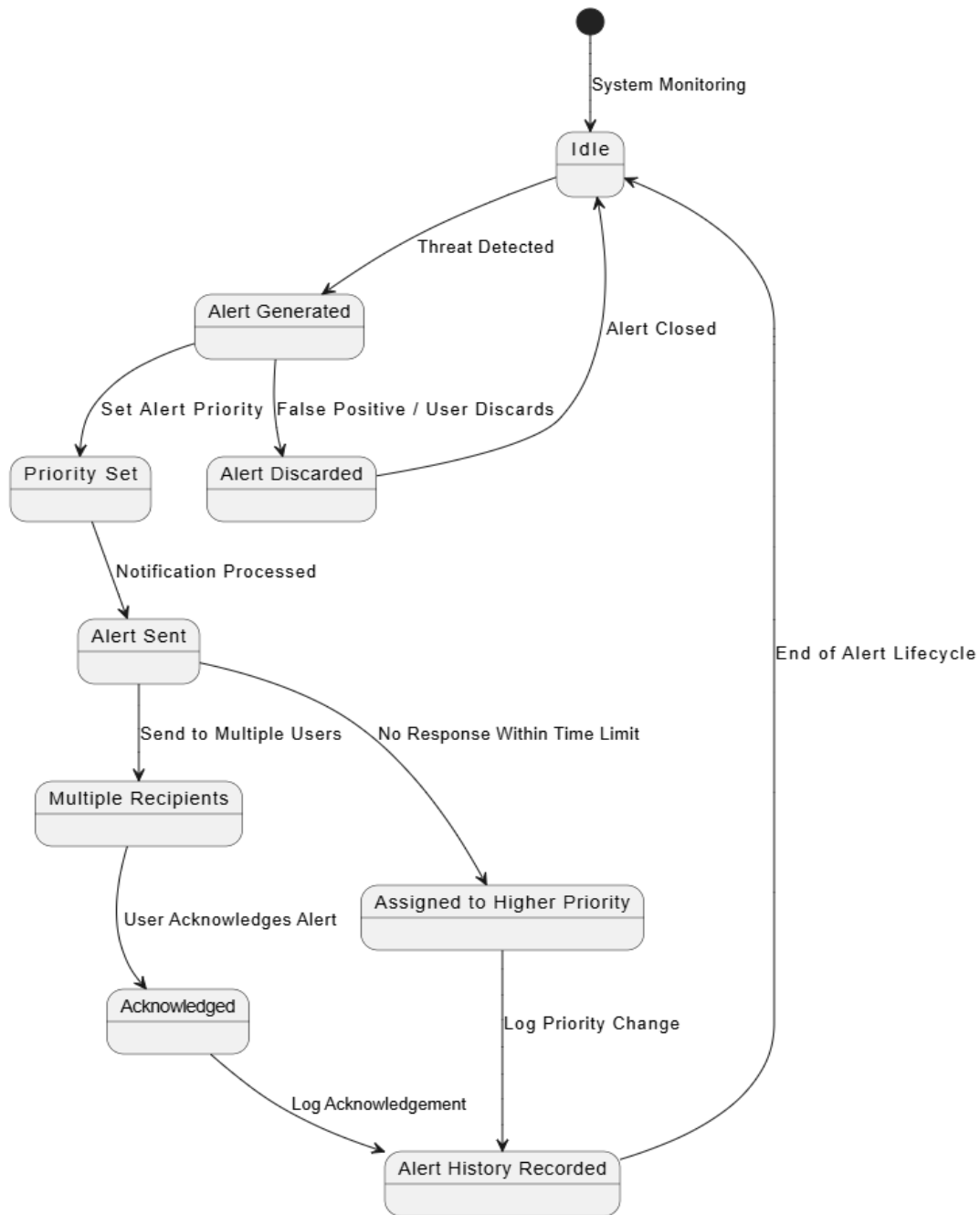


Figure : Modified Incremental Model

4.3.3 User States

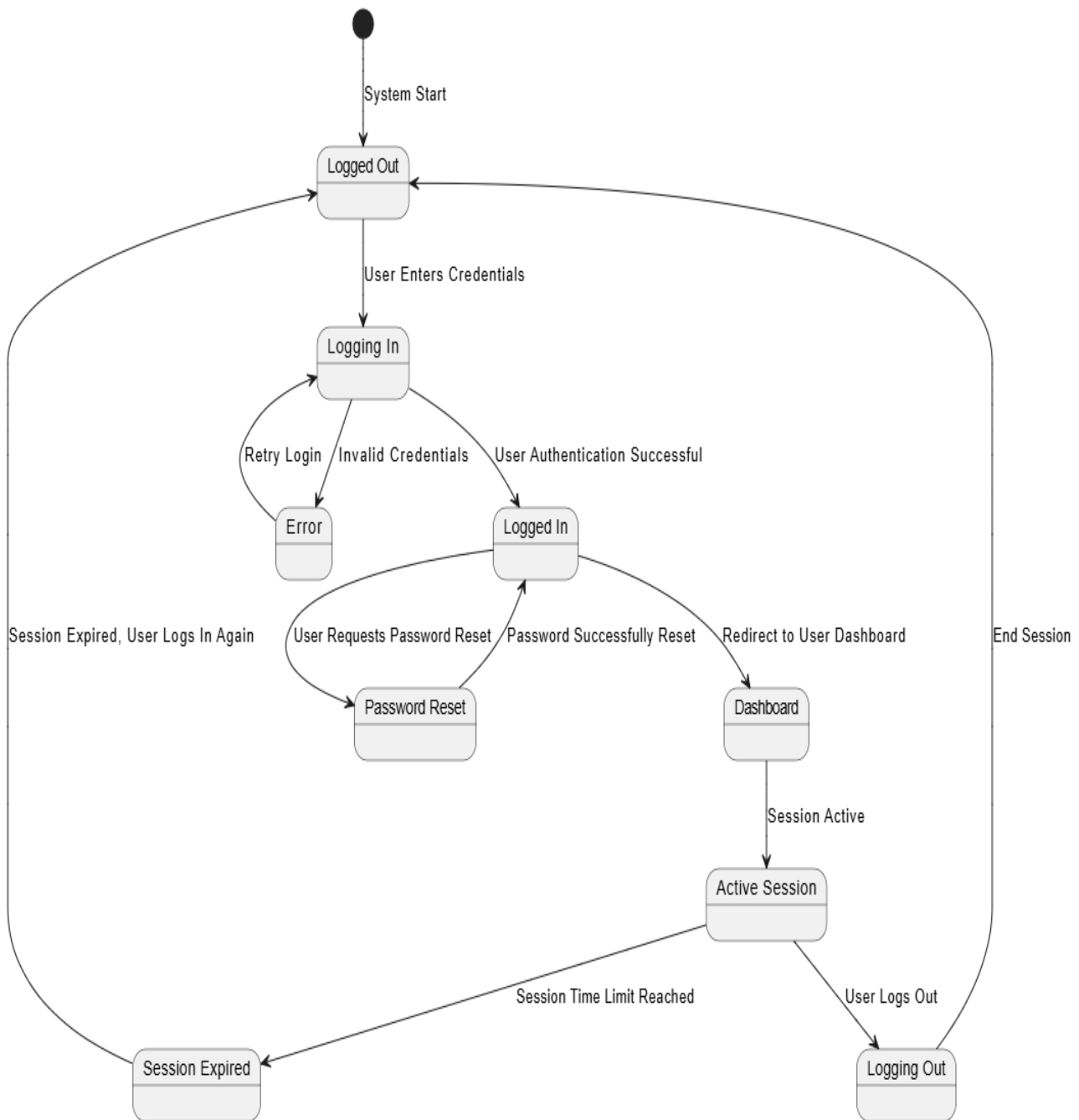


Figure : Modified Incremental Model

5. Data Design

The information domain for the Mihawk system is transformed into structured data models using MongoDB and the Mongoose library. The major data entities are stored in schemas that define the structure, constraints, and relationships of the data. The system ensures modularity and scalability by organizing the data into distinct collections.

Data Storage Items

1. **User Data Store:** Stores information about system users, including their roles and permissions.
2. **Drone Data Store:** Stores drone-related data such as logs, video feeds, and commands.
3. **Alert Data Store:** Logs alerts generated by the system during threat detection.
4. **Threat Data Store:** Stores analyzed threat data and suspicious activity details.
5. **Session Data Store:** Manages user session information for authentication and validation.

The data is processed and organized through a combination of:

- **Validation:** Ensuring constraints like uniqueness, required fields, and enums.
- **Middleware:** Automating operations such as timestamps and updates.
- **Relationships:** Linking related entities, such as users to permissions or roles.

5.1 Data Dictionary

5.3.1 Models:

Type	Member	Data Type	Description
User	name	String	Full name of the user.
	email	String	Unique identifier for the user; email address used for login.
	password	String	Encrypted password for authentication.
	role	String	Role of the user in the system, constrained to predefined roles (admin, operator, observer).
	permissions	Array	List of specific permissions assigned to the user.
	createdAt	Date	Timestamp indicating when the user account was created.
	updatedAt	Date	Timestamp indicating the last update made to the user account.

5.3.2 Functions:

Function	Function List	Parameters	Description
User Operations	createUser	name, email, password, role, permissions	Adds a new user with defined role and permissions to the system.
	updateUser	userId, updatedData	Updates the details of an existing user.
	deleteUser	userId	Removes a user from the system.
	getUser	userId	Retrieves details of a specific user.
	listUsers	filters	Returns a list of users based on filters such as role or permissions.
	authenticateUser	email, password	Validates the credentials of a user attempting to log in.

5.2 JSON Schema

5.3.1 User

```
const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
    trim: true,
  },
  email: {
    type: String,
    required: true,
    unique: true,
    lowercase: true,
    trim: true,
  },
  password: {
    type: String,
    required: true,
  },
  role: {
    type: String,
    enum: ['admin', 'operator', 'observer'], // Allowed roles
    required: true,
```

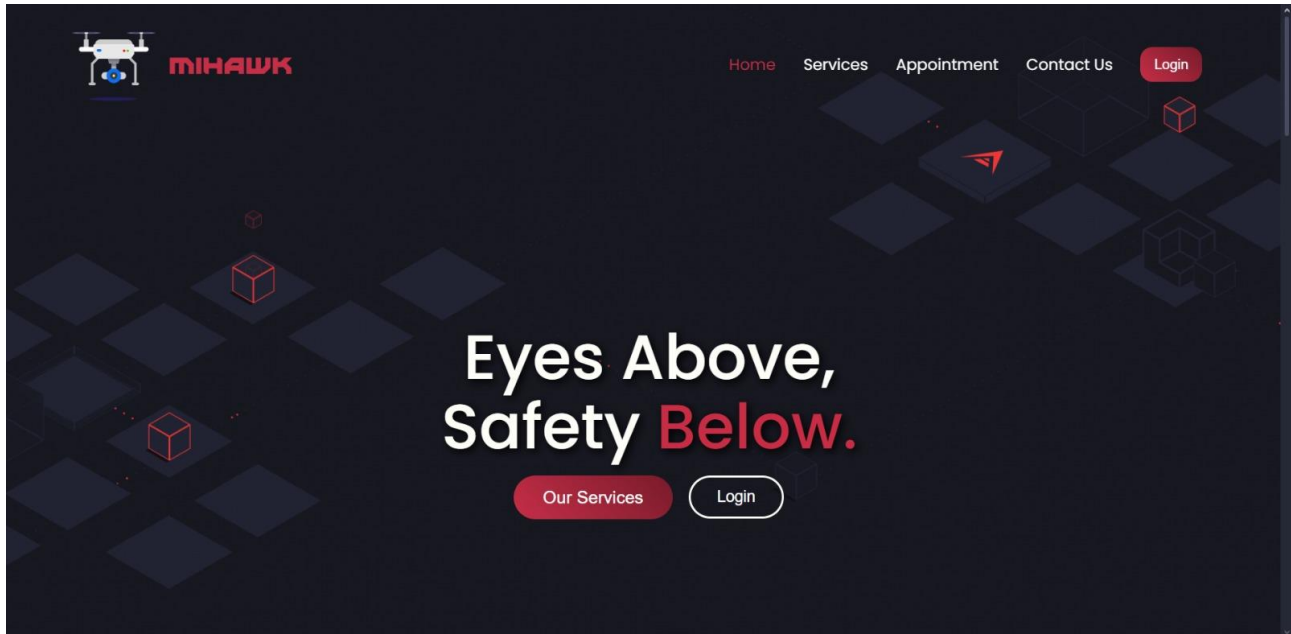
```
    },  
    permissions: {  
      type: Array, // Array of specific permissions if needed  
      default: [],  
    },  
    createdAt: {  
      type: Date,  
      default: Date.now,  
    },  
    updatedAt: {  
      type: Date,  
      default: Date.now,  
    },  
  });  
  
  userSchema.pre('save', function (next) {  
    this.updatedAt = Date.now();  
    next();  
  });  
  
  const User = mongoose.model('User', userSchema);  
  
  module.exports = User;
```

6. Human Interface Design

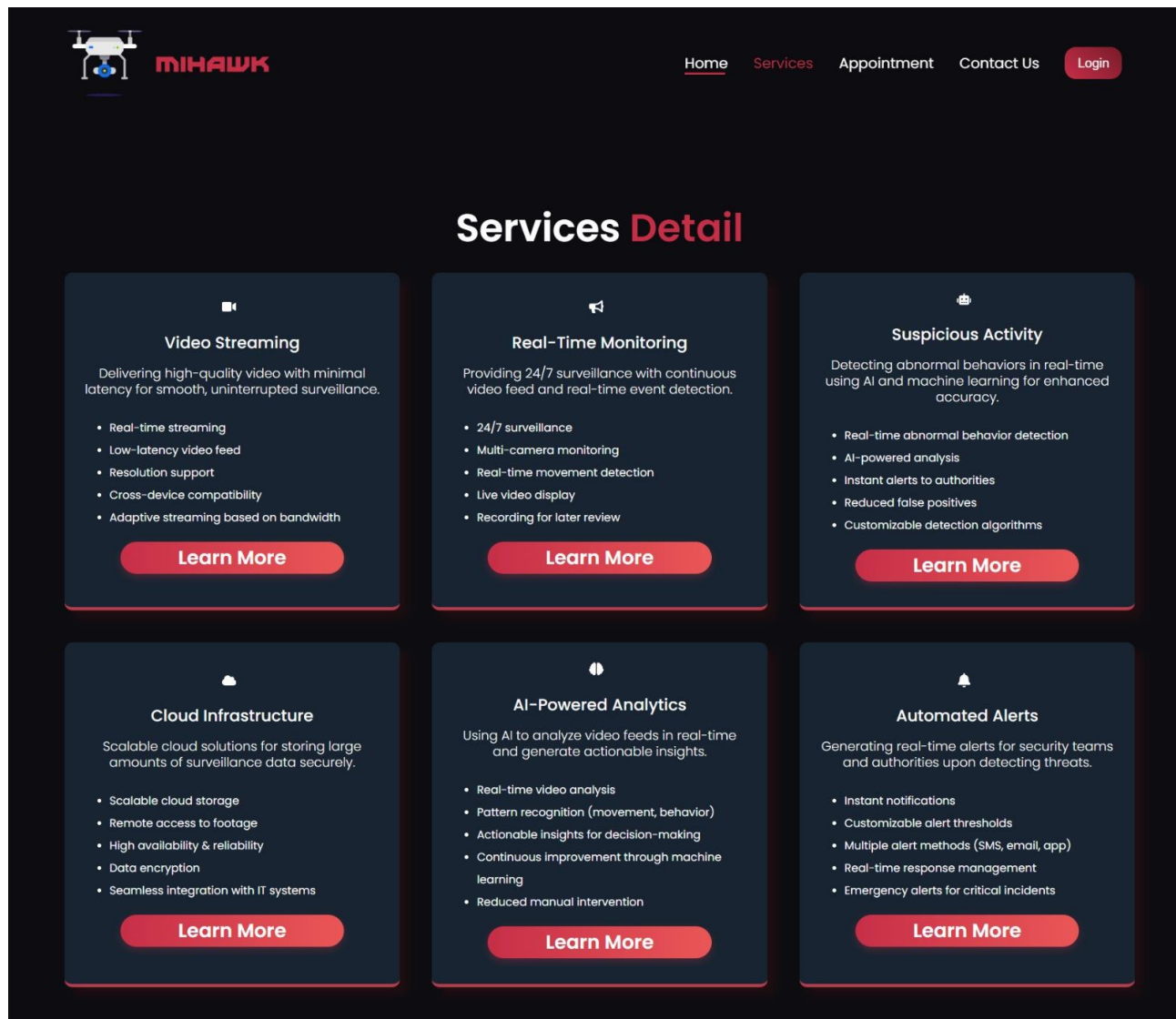
The Mihawk system allows users to monitor real-time video feeds from drones, track alerts for suspicious activities, and access data on surveillance operations. Users can view live video, receive AI-powered threat detection alerts, and review flagged footage. The system provides feedback through notifications, informing users of successful actions, errors, or issues, ensuring smooth operation. Security features, including data encryption, are in place to protect user information and video content, while the intuitive interface ensures ease of use for all system functions.

6.1 Screen Images


7.3.1 Landing page



7.3.1 Services page



7.3.1 Appointment Page


MIHAWK

[Home](#)
[Services](#)
[Appointment](#)
[Contact Us](#)
[Login](#)


Book An Appointment

December 2024

Su	Mo	Tu	We	Th	Fr	Sa
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

Select Time

Submit






Eyes Above, Safety Below


Quick Links

[Home](#)
[Services](#)
[Book An Appointment](#)
[Contact Us](#)
[Login](#)

Follow Us

 contact@mihawk.com
 +92 321 5211814
 Mihawk

7.3.1 Contact page





MIHAWK

[Home](#)
[Services](#)
[Appointment](#)
[Contact Us](#)
[Login](#)

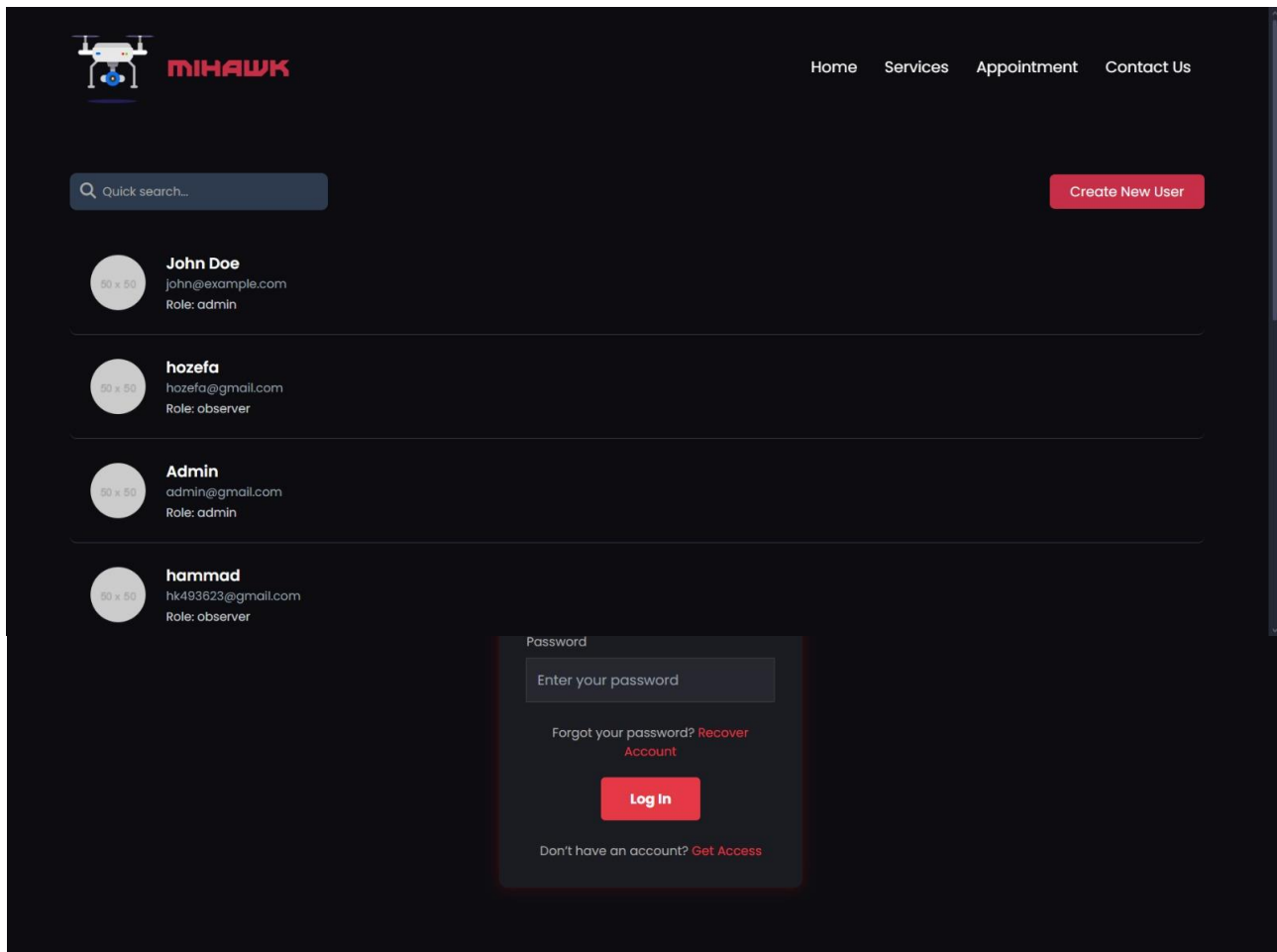
Contact Us

Let's get in touch!

Feel free to ask.

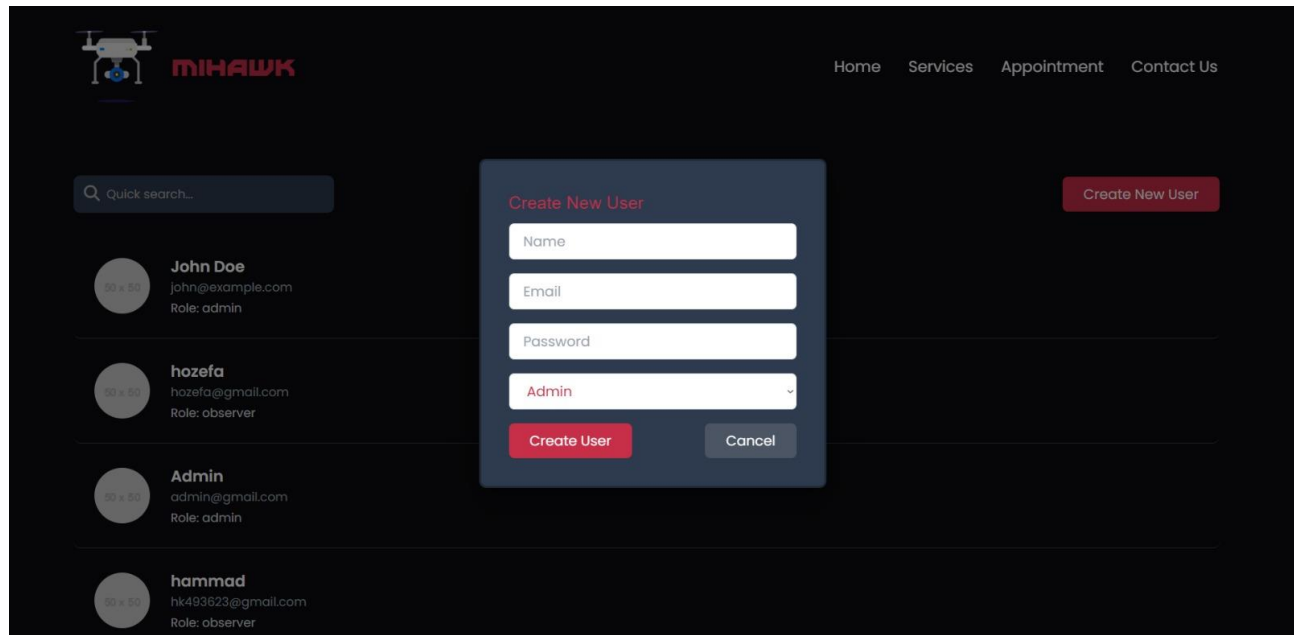
 contact@buzz-sol.com
 +92 321 5211814
 Mihawk

Submit



7.3.1 Login page

7.3.1 Create User Page



6.2 Screen Objects and Actions

Object	Action
Buttons	Buttons like "Our Services", "Login", "Submit", "Create User", etc., are used for various purposes like navigating between pages or submitting forms.
Form	Forms such as login, contact, appointment booking, and user creation allow users to enter data.
Input Fields	Used for data input such as Email, Password, Full Name, Phone, Service Selection, etc.
Navbar	Navbar provides navigation links for the user to move between sections of the application

	such as Home, Services, Appointment, Contact Us, and Login.
Icons	Icons are used for various purposes like visual representation of services, roles (admin/observer), and other actions (search, menu, etc.).
Images	Display information related to services, user details, or gallery (such as images of the drones, service details, etc.).
Modals	Pop-up modals like "Create New User" allow admins to perform actions like adding a new user or confirming an action.
Links	Links in buttons like "Learn More" or "Get Access" are used to provide additional information or navigate to a different section.

7. Implementation

7.1 Algorithm

Algorithm 1: ManualDroneFlightControl
Input: Current drone position, user inputs for movement (direction, speed)
Output: Drone reaches each waypoint in sequence (manual control)
<pre> 1: Initialize current_position ← start_position 2: Display current drone position on user interface 3: While user is controlling the drone: 4: Call HandleUserInput() to get user inputs for direction and speed 5: Call MoveDrone(direction, speed) to move the drone accordingly 6: Call ObstacleDetection() to check for obstacles in the drone's path 7: If obstacle is detected: 8: Stop drone and alert user 9: Wait for user to take corrective action 10: If the drone reaches a waypoint: 11: Call WaypointHandling(current_position, waypoint_position) to check if waypoint is reached 12: If waypoint is reached: 13: Ask user if they want to continue to the next waypoint 14: If user confirms, repeat steps 4–14 for the next waypoint </pre>

15: Else, stop the flight 16: End if 17: End while 18: Return success when user completes flight or terminates manually Algorithm: HandleUserInput	
Input: User input for direction and speed	
Output: Direction and speed values for drone movement	
1: Display available commands to the user 2: Wait for user to input direction (e.g., forward, backward, left, right) 3: Wait for user to input speed (increase, decrease, or maintain speed) 4: Validate user inputs for direction and speed (ensure inputs are within valid range) 5: Return the validated direction and speed to the main algorithm	
Algorithm: ObstacleDetection	Algorithm: MoveDrone
Input: Drone's current position, environment sensors	Input: Direction and speed
Output: Obstacle detected (True/False)	Output: Updated drone position
1: Use sensors (e.g., ultrasonic or camera-based) to detect obstacles in the drone's path 2: Calculate the distance from the nearest obstacle 3: If distance < safe_threshold: 4: Return True (obstacle detected) 5: Else: 6: Return False (no obstacle detected)	1: If direction is forward: 2: Move drone forward by the specified speed 3: Else If direction is backward: 4: Move drone backward by the specified speed 5: Else If direction is left: 6: Move drone left by the specified speed 7: Else If direction is right: 8: Move drone right by the specified speed 9: Update current drone position based on movement 10: Return updated drone position
Algorithm: WaypointHandling	
Input: Drone current position, waypoint position Output: Waypoint status (Reached/Not Reached) 1: Calculate the distance between the drone's current position and the waypoint 2: If distance < threshold: 3: Mark waypoint as reached 4: Prompt the user for the next action (continue or stop) 5: Else: 6: Continue moving towards the waypoint 7: Return waypoint status (Reached/Not Reached)	
Algorithm: VideoStreaming	Algorithm: SuspiciousActivityDetection
Input: Video feed, available bandwidth, video resolution	Input: Video feed, trained machine learning model
Output: Streamed video with adaptive resolution	Output: Alert if suspicious activity is detected
1: Initialize stream_quality ← high_resolution 2: Monitor available bandwidth 3: If bandwidth is high: 4: Continue streaming in high_resolution 5: Else If bandwidth is medium: 6: Switch stream_quality ← medium_resolution 7: Else If bandwidth is low: 8: Switch stream_quality ← low_resolution 9: Send video frames to server 10: If video frame transmission fails: 11: Retry transmission or log the failure	1: Initialize model ← pre-trained activity_detection_model 2: Capture video frame from video_feed 3: For each frame in video_feed: 4: Apply model to detect activities 5: If activity is detected: 6: If activity is suspicious: 7: Trigger alert 8: Log activity with timestamp and type 9: End if 10: End for 11: Continue processing next frame

12: Continue streaming until drone operation ends 13: Return success or error status	12: Return success or error status
---	------------------------------------

8

9

Algorithm 1: AlertGeneration
Input: Suspicious activity detected
Output: Alert to authorities
1: If suspicious activity is detected: 2: Generate alert with details of activity (time, location, type) 3: Send alert via email, SMS, or other communication channels 4: Log alert in the system for future reference 5: Notify security personnel or law enforcement with the alert details 6: Continue monitoring for additional activities 7: Return success or error status
Algorithm: CloudStorageManagement
Input: Video file, storage service
Output: Video file stored and retrievable
1: Initialize cloud_storage_service ← cloud_provider_service 2: For each video_file in video_feed: 3: Upload video_file to cloud_storage_service 4: If upload is successful: 5: Log success with timestamp 6: Else: 7: Retry upload or store video in a temporary buffer 8: End for 9: Retrieve video file from cloud storage when requested 10: If retrieval fails: 11: Attempt to retrieve from backup storage 12: Return success or error status16: elseif ((coauthor1Fragments[0][0] == coauthor2Fragments[0][0] and coauthor1Fragments[0][1] == coauthor2Fragments[0][1] and coauthor1Fragments[0][2] == coauthor2Fragments[0][2]) and (coauthor1Fragments[2] == coauthor1Fragments[2])) then 17: //both authors have same first three characters of first name and full last name 18: if ((coauthor1Fragments[1] == coauthor2Fragments[1]) or (coauthor1Fragments[1][0] == coauthor2Fragments[1][0])) then 19: //both authors have same middle full name or same first character of middle name 20: Count++ 21: endif 22: end elseif 23: elseif (len(coauthor1Fragments) > 3 and len(coauthor2Fragments) > 3) then //both have more than three name fragments 24: if ((coauthor1Fragments[0][0] == coauthor2Fragments[0][0] and coauthor1Fragments[0][1] == coauthor2Fragments[0][1] and coauthor1Fragments[0][2] == coauthor2Fragments[0][2]) and ((coauthor1Fragments[len(coauthor1Fragments)-1] == coauthor2Fragments[len(coauthor2Fragments)-1]) or (coauthor1Fragments[len(coauthor1Fragments)-1][0] == coauthor2Fragments[len(coauthor2Fragments)-1][0]))) then //both have same first three characters of first name and either full last name or first character of last name 25: if (coauthor1Fragments[1][0] == coauthor2Fragments[1][0]) then //both have same first character of their second name 26: count++ 27: end if 28: end if 29: end elseif 30: end foreach 31: end foreach 32: if (count ≥ 1) then //number of similar co-authors excluding author in question

33: Flag ← true 34: endif 35: return Flag	
Algorithm: UserAuthentication	Algorithm: RoleManagement
Input: Username, password	Input: User credentials, role requirements
Output: Authentication status	Output: Access granted or denied
1: Initialize user_database ← database_of_users 2: For each user in user_database: 3: If username matches user.username and password matches user.password: 4: Authentication successful 5: Return authenticated status 6: End if 7: End for 8: If no match is found: 9: Return authentication failed status	1: Initialize role_permissions ← pre- defined_role_permissions 2: If user has correct role: 3: Grant access to resources 4: Return access granted 5: Else: 6: Deny access and log unauthorized attempt 7: Return access denied
Algorithm: VideoFeedAnalysis	
Input: Video feed, AI model Output: Anomalous behavior detection 1: Initialize ai_model ← trained_anomaly_detection_model 2: For each frame in video_feed: 3: Process frame using ai_model 4: If anomaly detected: 5: Log anomaly with timestamp and location 6: Trigger alert if necessary 7: Continue processing next frame 8: Return success or error status	
Algorithm: ContinuousLearning	Algorithm: APIIntegration
Input: Detected activity, feedback	Input: API request from external system
Output: Improved model accuracy	Output: Response to external system
1: Initialize learning_model ← initial_model 2: For each detected anomaly: 3: Collect feedback on the detection accuracy 4: If feedback indicates error: 5: Retrain model with new data 6: Update model parameters 7: End for 8: Return success or updated model	1: Initialize api_client ← external_system_api_client 2: Send API request to external system 3: If response is successful: 4: Process response and send back required data 5: Else: 6: Retry API request or log failure 7: Return success or error status

Algorithm: RealTimeReporting	
Input: System data, report type	
Output: Generated report	
1: Initialize reporting_system \leftarrow report_generation_tool 2: Collect real-time data from system 3: Generate report based on collected data 4: Display updated report on dashboard 5: If report generation fails: 6: Retry or log failure 7: Continue generating reports periodically 8: Return success or error status	
Algorithm: VideoStreaming	
Input: Video feed, available bandwidth, video resolution	
Output: Streamed video with adaptive resolution	
1: Initialize stream_quality \leftarrow high_resolution 2: Monitor available bandwidth 3: If bandwidth is high: 4: Continue streaming in high_resolution 5: Else If bandwidth is medium: 6: Switch stream_quality \leftarrow medium_resolution 7: Else If bandwidth is low: 8: Switch stream_quality \leftarrow low_resolution 9: Send video frames to server 10: If video frame transmission fails: 11: Retry transmission or log the failure 12: Continue streaming until drone operation ends 13: Return success or error status	
Algorithm: MonitorBandwidth	Algorithm: SuspiciousActivityDetection
Input: Network status	Input: Video feed, trained machine learning model
Output: Available bandwidth	Output: Alert if suspicious activity is detected
1: Measure current bandwidth from the network 2: If bandwidth > high_threshold: 3: Return "high" bandwidth 4: Else If bandwidth > medium_threshold: 5: Return "medium" bandwidth 6: Else: 7: Return "low" bandwidth	1: Initialize model \leftarrow pre-trained activity_detection_model 2: Capture video frame from video_feed 3: For each frame in video_feed: 4: Apply model to detect activities 5: If activity is detected: 6: If activity is suspicious: 7: Trigger alert 8: Log activity with timestamp and type 9: End if 10: End for 11: Continue processing next frame 12: Return success or error status
Algorithm: ApplyActivityModel	
Input: Video frame, activity detection model	
Output: Detected activity	
1: Apply pre-trained model to the video frame 2: If activity is detected: 3: Return detected activity 4: Else:	

5: Return no activity detected	
Algorithm: AlertGeneration	Algorithm: CloudStorageManagement
Input: Suspicious activity detected	Input: Video file, storage service
Output: Alert to authorities	Output: Video file stored and retrievable
1: If suspicious activity is detected: 2: Generate alert with details of activity (time, location, type) 3: Send alert via email, SMS, or other communication channels 4: Log alert in the system for future reference 5: Notify security personnel or law enforcement with the alert details 6: Continue monitoring for additional activities 7: Return success or error status	1: Initialize cloud_storage_service ← cloud_provider_service 2: For each video_file in video_feed: 3: Upload video_file to cloud_storage_service 4: If upload is successful: 5: Log success with timestamp 6: Else: 7: Retry upload or store video in a temporary buffer 8: End for 9: Retrieve video file from cloud storage when requested 10: If retrieval fails: 11: Attempt to retrieve from backup storage 12: Return success or error status

Algorithm: UploadVideo	
Input: Video file, cloud storage service	
Output: Success or failure of video upload	
1: Upload video_file to cloud_storage_service 2: If upload is successful: 3: Return success 4: Else: 5: Retry upload or store video in backup 6: Return failure after retrying	
Algorithm: UserAuthentication	
Input: Username, password	
Output: Authentication status	
1: Initialize user_database ← database_of_users 2: For each user in user_database: 3: If username matches user.username and password matches user.password: 4: Authentication successful 5: Return authenticated status 6: End if 7: End for 8: If no match is found: 9: Return authentication failed status	
Algorithm: RoleManagement	
Input: User credentials, role requirements	
Output: Access granted or denied	

1: Initialize role_permissions ← pre-defined_role_permissions 2: If user has correct role: 3: Grant access to resources 4: Return access granted 5: Else: 6: Deny access and log unauthorized attempt 7: Return access denied	
Algorithm: ContinuousLearning	
Input: Detected activity, feedback	
Output: Improved model accuracy	
1: Initialize learning_model ← initial_model 2: For each detected anomaly: 3: Collect feedback on the detection accuracy 4: If feedback indicates error: 5: Retrain model with new data 6: Update model parameters 7: End for 8: Return success or updated model	
Algorithm: APIIntegration	
Input: API request from external system	
Output: Response to external system	
1: Initialize api_client ← external_system_api_client 2: Send API request to external system 3: If response is successful: 4: Process response and send back required data 5: Else: 6: Retry API request or log failure 7: Return success or error status	
Algorithm: RealTimeReporting	
Input: System data, report type	
Output: Generated report	
1: Initialize reporting_system ← report_generation_tool 2: Collect real-time data from system 3: Generate report based on collected data 4: Display updated report on dashboard 5: If report generation fails: 6: Retry or log failure 7: Continue generating reports periodically 8: Return success or error status	
Algorithm: VideoFeedAnalysis	
Input: Video feed, AI model	
Output: behavior detection	
1: Initialize ai_model ← trained_anomaly_detection_model 2: For each frame in video_feed:	

3: Process frame using ai_model 4: If anomaly detected: 5: Log anomaly with timestamp and location 6: Trigger alert if necessary 7: Continue processing next frame 8: Return success or error status	
Algorithm: VideoFeedAnalysis	

7.2 External APIs/SDKs

The following table lists the third-party APIs/SDKs used in the Mihawk project implementation:

Table : Details of APIs used in the project

Name of API and version	Description of API	Purpose of usage	List down the API endpoint/function/class in which it is used
Weather API (Version 1.0)	Provides real-time and forecasted weather data, including temperature, humidity, and wind conditions.	Displays weather monitoring data on the website and for integration with drone flight planning.	GET /current GET /forecast weather.data.fetch()

7.3 User Interface

7.3.1 Landing Page

Landing page of our application where users are introduced to the platform. It includes an overview of features, a navigation menu, and options to log in.

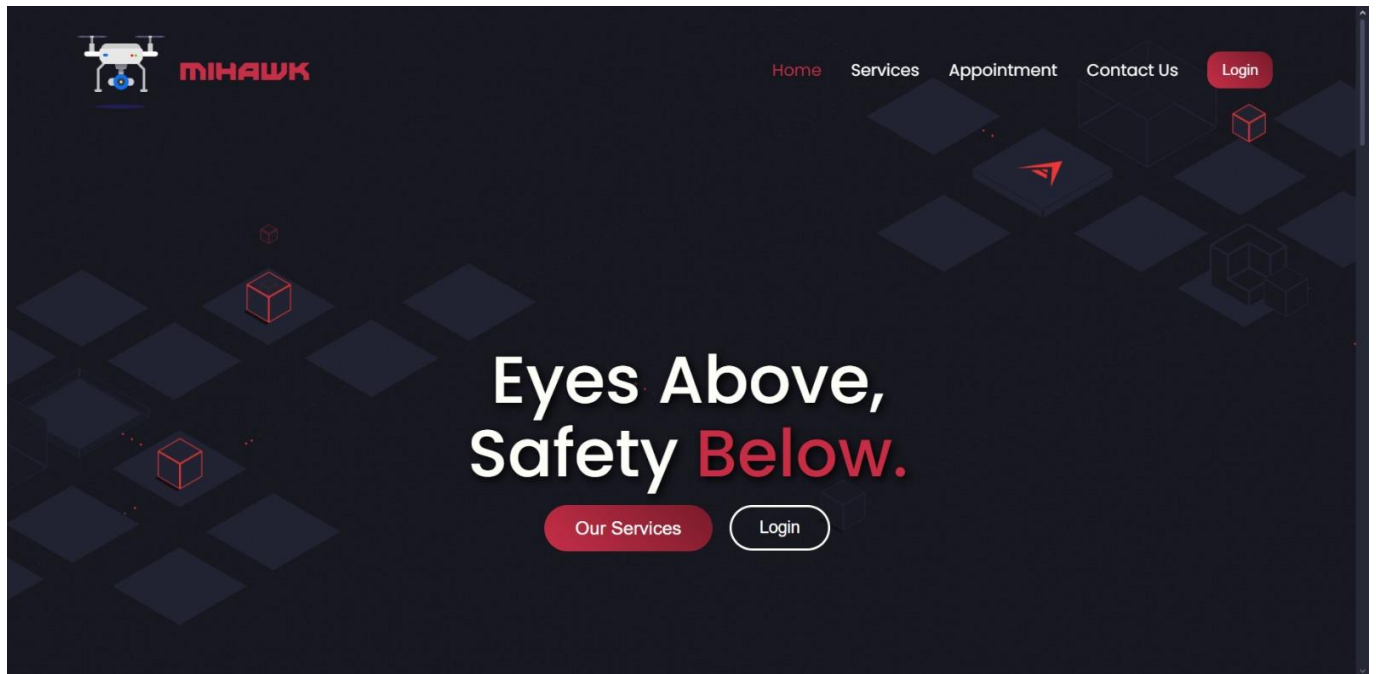


Figure : Landing Page

7.3.2 Services Page

The services page of our application where users can explore detailed information about the services offered, including descriptions, benefits.

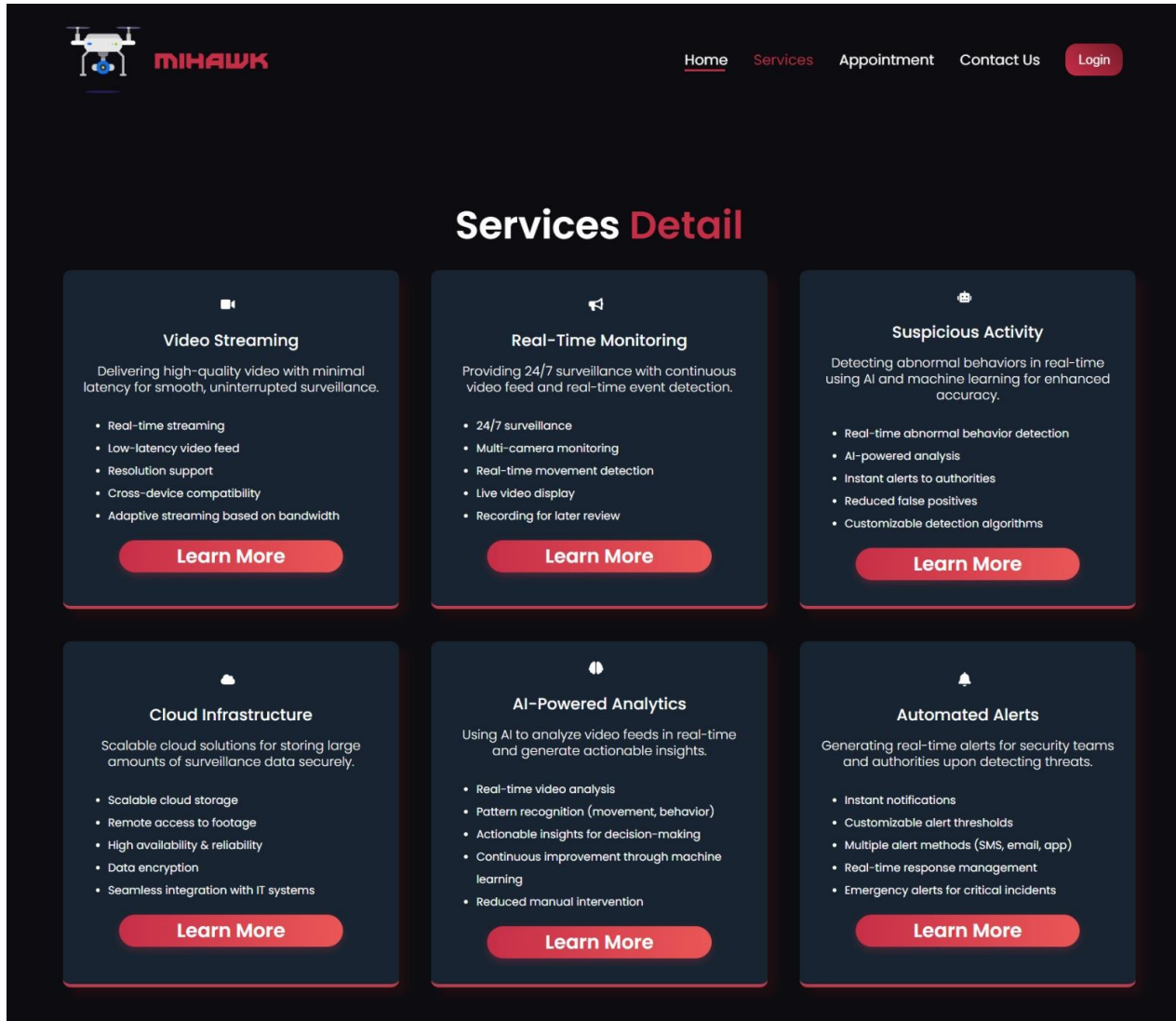


Figure : Services Page

7.3.3 Book an Appointment Screen

Book an Appointment screen of our application where users can select a service, choose a date and time, and provide necessary details to schedule an appointment.

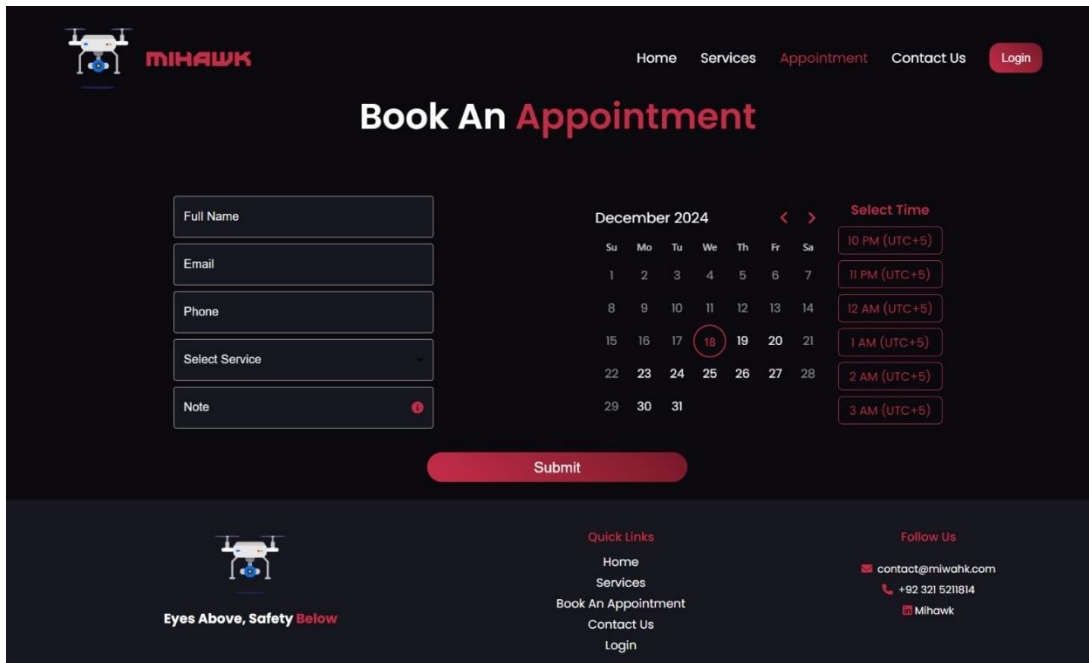


Figure : Book an Appointment Page

7.3.4 Contact us Screen

Contact Us page of our application where users can reach out for support, inquiries, or feedback by filling out a form or accessing contact details such as phone number and email.

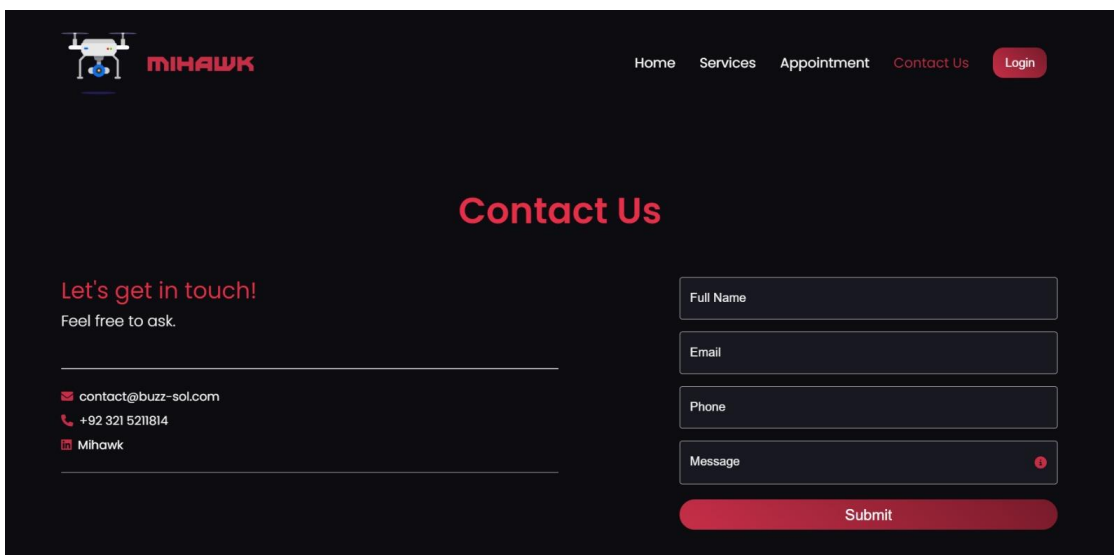


Figure : Contact Us Page

7.3.5 Login Screen

Login screen of our application where users enter their credentials, such as email and password, to access their accounts securely.

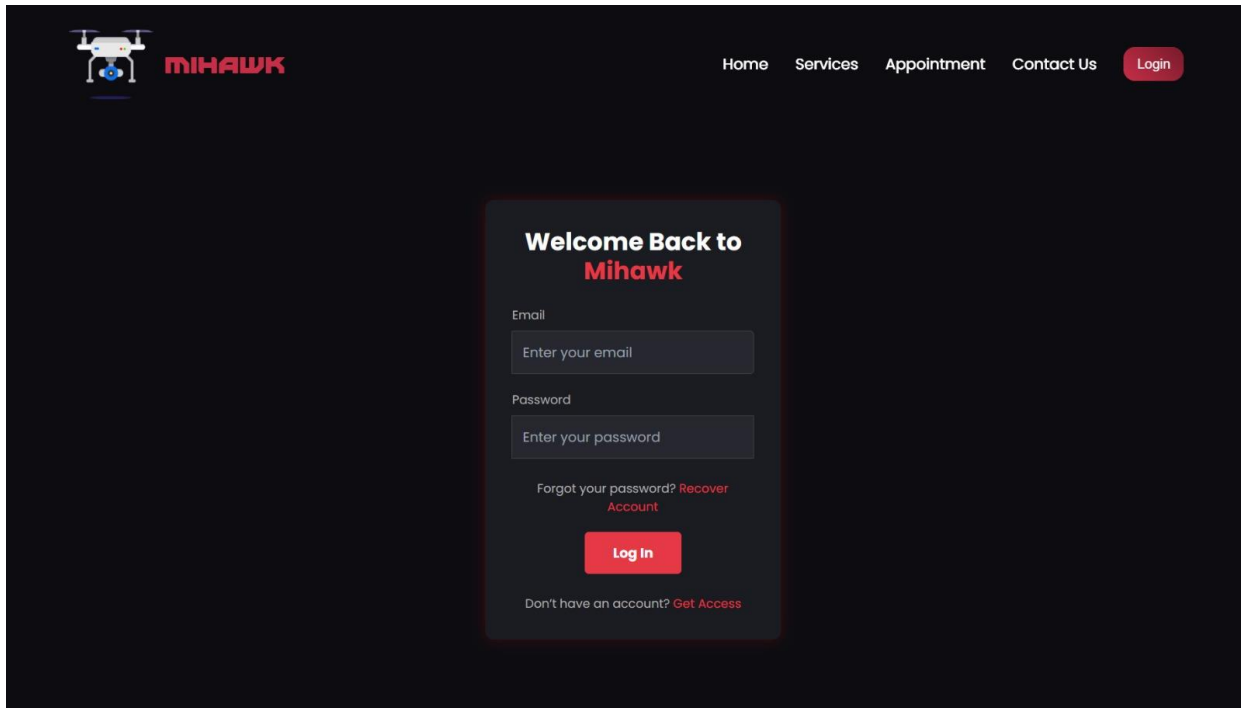


Figure : Login Page

7.3.6 Manage User

Manage User screen of our application where administrators can view, edit, delete, or update user information and manage their roles and permissions

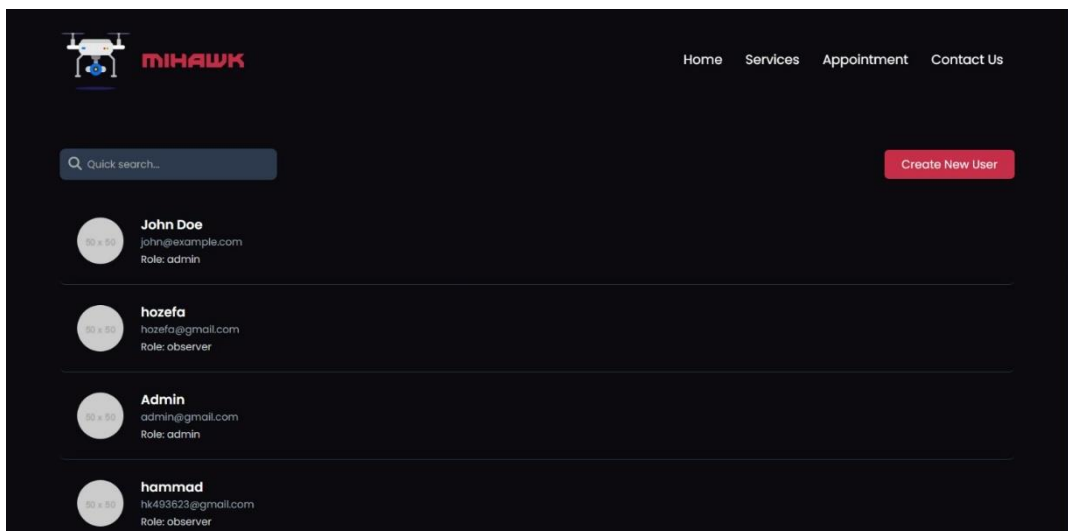


Figure : Manage Page

7.3.7 Create User

Create User screen of our application where administrators can add new users by entering their details, assigning roles, and setting initial account credentials.

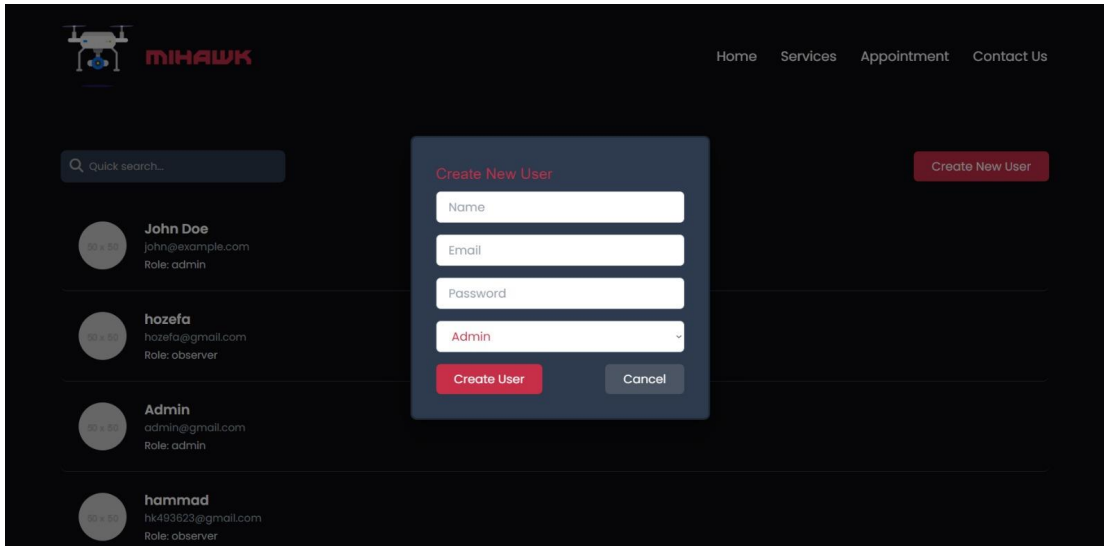


Figure : Create User Page

7.4 Deployment

The core application, including the user interface and backend logic, is deployed and tested on local machines during development. This setup ensures efficient testing of features and functionalities. Weather monitoring will be integrated using a cloud-based Weather API, accessed via secure HTTPS requests. Drone monitoring subsystems process data locally, with future for cloud storage to handle large-scale operations. The threat detection AI model will be hosted on Kaggle's cloud platform during development, utilizing GPU support for real-time computations and API calls for integration. In production, the system will transition to a scalable cloud-based hosting environment, such as AWS or Azure, with deployment pipelines using Docker and CI/CD tools for streamlined updates.

8. Testing and Evaluation

8.1 Unit Testing

Unit Testing 1: Login as User with valid and invalid credentials

Testing Objective: To ensure the login form is working correctly with valid and invalid credentials/inputs.

No.	Test case/Test script	Attribute and value	Expected result	Result
1	Check the email field of login to validate that it takes proper email	Email: abc@gmail.com	Validates email address and moves cursor to next textbox	Pass
2	Check the email field of login to validate that it displays error message.	Email: abc@gmail.com	Highlights field and displays error message	Pass

Unit Testing 2: User Registration

Testing Objective: To ensure that the user registration form is functioning as expected.

No.	Test case/Test script	Attribute and value	Expected result	Result
1	Check the registration form for valid input	Name: John Doe, Email: john@example.com, Password: John123	Successfully registers the user and redirects to the login page	Pass
2	Check for invalid email format in registration form	Name: John Doe, Email: johnexample.com, Password: John123	Highlights email field and displays error message indicating invalid email format	Pass

Unit Testing 3: Drone Control

Testing Objective: To ensure that the drone can be controlled successfully by security personnel.

No.	Test case/Test script	Attribute and value	Expected result	Result
1	Check manual control for drone navigation	Action: Move Forward, Left, Right, Hover	Drone moves in the specified direction and hovers as commanded	Pass
2	Check emergency stop feature for drone	Action: Emergency Stop	Drone immediately halts all movement and alerts the user of the action	Pass

Unit Testing 4: Threat Detection - Weapons

Testing Objective: To verify that the system can correctly detect and classify weapons in the surveillance feed.

No.	Test case/Test script	Attribute and value	Expected result	Result
-----	-----------------------	---------------------	-----------------	--------

1	Check weapon detection in surveillance area	Object: Weapon, Type: Gun	Detects weapon in the surveillance feed and triggers an alert notification to security personnel	Pass
2	Check weapon detection in surveillance area	Object: Weapon, Type: Knife	Detects weapon (e.g., knife) in the surveillance feed and triggers an alert notification	Pass
3	Check false positive for weapon detection	Object: Metal Object, Type: Unrelated Item	System correctly identifies the object as non-threatening and does not trigger an alert	Pass
4	Check response to weapon detection at various distances	Object: Weapon, Distance: Far, Medium, Close	System detects the weapon at various distances and triggers an appropriate alert based on proximity	Pass

Unit Testing 5: Threat Detection - Unattended Bags

Testing Objective: To ensure the system can detect and classify unattended bags as potential threats.

No.	Test case/Test script	Attribute and value	Expected result	Result
1	Check unattended bag detection in surveillance area	Object: Bag, Status: Unattended	Detects an unattended bag in the surveillance area and triggers an alert notification to security personnel	Pass
2	Check system response to unattended bags in crowded areas	Object: Bag, Location: Crowd Area	Detects an unattended bag in a crowded area and triggers an alert notification with a priority flag for quick attention	Pass
	Check false positive for unattended bag detection	Object: Bag, Status: Attended	System identifies the attended bag and does not trigger an alert, as it's not a threat	Pass

	Check unattended bag detection with varying bag types	Object: Bag, Type: Backpack, Suitcase, Handbag	System successfully detects and classifies different types of unattended bags and sends alerts accordingly	Pass
--	---	--	--	------

Unit Testing 6: User Management (Role-based access control)

Testing Objective: To ensure the login form is working correctly with valid and invalid credentials/inputs.

No.	Test case/Test script	Attribute and value	Expected result	Result
1	Check access for User role	Role: User	User should only have access to basic functionalities such as viewing live surveillance feeds and receiving alerts.	Pass
2	Check access for Operator role	Role: Operator	Operator should have access to additional functionalities, such as controlling drones and managing surveillance settings, but not system configuration.	Pass
3	Check access for Admin role	Role: Admin	Admin should have full access to all system functionalities, including user management, system configuration, and full surveillance control.	Pass
4	Check restricted access for User role to administrative functions	Role: User, Action: Access Admin Dashboard	User should be denied access to the Admin	Pass

			Dashboard and see a "Permission Denied" message.	
5	Check restricted access for Operator role to system settings	Role: Operator, Action: Access System Settings	Operator should be denied access to system settings and see a "Permission Denied" message.	Pass
6	Check Admin role access to modify user roles	Role: Admin, Action: Modify User Role	Admin should be able to modify the roles of other users (e.g., assign User, Operator, or Admin roles) without restrictions.	Pass
7	Check Admin role access to manage system configuration	Role: Admin, Action: Access System Settings	Admin should be able to access system settings and modify configurations, including security and alert settings.	Pass

Unit Testing 7: Data Handling for Efficient Storage and Retrieval

Testing Objective: To ensure that the system can efficiently store and retrieve surveillance data.

No.	Test case/Test script	Attribute and value	Expected result	Result
1	Check data compression for storage	Data: Surveillance Video	Data is compressed efficiently to reduce storage requirements without losing quality	Pass
2	Check data retrieval functionality	Action: Retrieve stored surveillance data	Stored data is retrieved quickly and accurately for review or playback	Pass

8.2 Functional Testing

Functional Testing 1: Login with different roles (User, Operator, Admin)

Objective: To ensure that the correct page with the correct navigation bar is loaded based on the user role.

No.	Test case/Test script	Attribute and value	Expected result	Actual result	Result
1.	Login as a User member	Username: user123, Password: userPass123	Main page for User is loaded with the User navigation bar.	Logged in and redirected to User main page with User navigation bar.	Pass
2.	Login as an Operator member	Username: operator456, Password: operatorPass456	Main page for Operator is loaded with the Operator navigation bar.	Logged in and redirected to Operator main page with Operator navigation bar.	Fail
3	Login as an Admin member	Username: admin789, Password: adminPass789	Main page for Admin is loaded with the Admin navigation bar, and full admin functionalities are available.	Logged in and redirected to Admin main page with Admin navigation bar and access to admin functionalities.	Pass
4	Login with invalid User credentials	Username: user123, Password: wrongPassword123	Login fails – invalid credentials error message displayed.	Login failed – invalid credentials error.	Pass
5	Login with invalid Operator credentials	Username: operator456, Password: wrongPassword456	Login fails – invalid credentials error message displayed.	Login failed – invalid credentials error.	Pass
6	Login with empty Username and Password	Username: [blank], Password: [blank]	Login fails – both fields are required, and error message is displayed.	Login failed – both fields required.	Fail
7	Login with correct Admin credentials but without internet	Username: admin789, Password: adminPass789	Login fails – "No internet connection" error is displayed.	Login failed – No internet connection.	Fail

Functional Testing 2: Drone Control Access by Roles

Objective: To ensure that the system allows appropriate access to drone control based on the user's role.

No.	Test case/Test script	Attribute and value	Expected result	Actual result	Result
1.	Operator accesses Drone Control	Role: Operator, Action: Access Drone Control	Operator should be able to control the drone and view its status in real time.	Successfully controlled drone and viewed status.	Pass
2.	Admin accesses Drone Control	Role: Admin, Action: Access Drone Control	Admin should be able to access and control the drone, along with modifying drone settings and configurations.	Successfully controlled drone and modified settings.	Pass
3	User accesses Drone Control	Role: User, Action: Access Drone Control	User should be denied access to the drone control, with a "Permission Denied" message.	Access denied – Permission Denied.	Pass
4	Operator accesses Drone Control without sufficient permissions	Role: Operator, Action: Unauthorized access attempt	Operator should see an error message indicating insufficient permissions to control advanced drone functions.	Access denied – Insufficient permissions.	Fail
5	User tries to control the drone by manually entering commands	Role: User, Action: Manually enter drone commands	System should not allow drone control and show a "Permission Denied" message.	Error message "Permission Denied" displayed.	Pass

Functional Testing 3: Threat Detection (Weapon and Unattended Bag)

Objective: To ensure that the system correctly detects threats (weapons and unattended bags) in surveillance.

No.	Test case/Test script	Attribute and value	Expected result	Actual result	Result
1.	Detect Weapon in surveillance footage	Object: Gun, Location: Entrance Area	System detects weapon (gun) and triggers an alert notification to security personnel.	Weapon detected and alert triggered.	Pass
2.	Detect Unattended Bag in surveillance footage	Object: Bag, Status: Unattended, Location: Lobby	Unattended bag detected and alert triggered.	Unattended bag detected and alert triggered.	Pass
3	False positive detection for Weapon	Object: Metal Object, Type: Non-threatening	System identifies the object as non-threatening and does not trigger an alert.	No alert triggered.	Pass
4	False positive detection for Unattended Bag	Object: Bag, Status: Attended, Location: Reception Area	System correctly identifies the attended bag and does not trigger an alert.	No alert triggered.	Pass
5	Weapon detection in a crowded environment	Object: Gun, Location: Crowded Area	System detects weapon in a crowded area and triggers an alert to security personnel for immediate action.	Weapon detected and alert triggered in crowded area.	Pass
6	Unattended Bag detection with poor lighting	Object: Bag, Status: Unattended, Location: Dark Room	System fails to detect the unattended bag in low-light conditions and does not trigger an alert.	No alert triggered – detection failed.	Fail

Functional Testing 4: Data Storage**Objective:** To verify that flagged surveillance data is securely stored and retrieved from **database**.

No.	Test case/Test script	Attribute and value	Expected result	Actual result	Result
1.	Store flagged Threat Data in database	Data: Flagged Surveillance Footage (Weapon detection)	System encrypts and stores the flagged data securely in the database for future retrieval.	Data stored securely in the database .	Pass
2.	Retrieve flagged Threat Data from database	Data: Flagged Surveillance Footage (Unattended Bag)	System retrieves the flagged data from the database , ensuring the integrity and security of the data.	Data retrieved securely from the database .	Pass
	System behavior when database is unavailable	Data: Flagged Surveillance Footage (Weapon detection)	If the database is unavailable, the system should handle retries and store data locally, then upload when connection is restored.	Data stored locally – upload successful once database is back online.	Pass
	Check database data retrieval failure	Data: Flagged Surveillance Footage (Unattended Bag)	If connection to the database is lost during data retrieval, the system should notify the user and retry after reconnection.	Retrieval failed due to connection loss; system retried and retrieved data successfully.	Pass
	Store flagged Threat Data when database is down	Data: Flagged Surveillance Footage (Weapon detection)	If the database is down, the system should store flagged data locally and notify the user. Data will be uploaded once the connection is restored.	Data stored locally – upload pending until the database is accessible again.	Fail

8.3 Business Rules Testing

Business Rule Testing 1: A threat is detected only if it is classified as either a weapon or an unattended bag.

Conditions:

Threat Detected: Whether the detected object is classified as a **Weapon** or **Unattended Bag**.

Action:

Success: If a valid threat is detected, the system triggers the appropriate alert and notifies authorities if necessary.

Failure: If the detected object is not classified as a valid threat, no action is triggered.

Conditions	R1	R2	R3	R4
Threat Detected	Weapon	Weapon	Unattended Bag	Unattended Bag
Actions				
Trigger Alert	T	T	T	T
Classify as High Priority	T	T	F	F
Notify Authorities	T	T	F	F

Business Rule Testing 2: A threat is classified based on its type and location.

Conditions:

Threat Type: The type of threat detected (Weapon or Unattended Bag).

Location Type: Where the threat is detected (Open Space or Crowded Area).

Action:

Success: Based on the threat type and location, the system takes appropriate actions such as triggering alerts and notifying authorities.

Failure: If the system does not recognize a valid threat, it does not take any actions.

Conditions	R1	R2	R3	R4
Threat Detected	Weapon	Weapon	Unattended Bag	Unattended Bag
Location Type	Open Space	Crowded Area	Open Space	Crowded Area
Actions				
Trigger Alert	T	T	T	T
Classify as High Priority	T	T	F	F
Notify Authorities	T	T	F	F

Business Rule Testing 3: System Behavior for Different Threat Detection Scenarios**Conditions:****Threat Type:** The type of detected threat (Weapon or Unattended Bag).**Location Type:** The area where the threat is detected (Open Space or Crowded Area).**Action:****Success:** The system takes appropriate actions based on the threat type and location, such as triggering alerts, classifying as high priority, and notifying authorities.**Failure:** If the conditions do not match recognized threats, no action is taken.

Conditions	R1	R2	R3	R4
Threat Detected	Weapon	Weapon	Unattended Bag	Unattended Bag
Location Type	Open Space	Crowded Area	Open Space	Crowded Area
Actions				
Trigger Alert	T	T	T	T
Classify as High Priority	T	T	F	F
Notify Authorities	T	T	F	F

8.4 Integration Testing

Integration Testing 1: Threat Detection and Alert System**Testing Objective:** To ensure seamless integration between threat detection and alert notification system.

No.	Test case/Test script	Attribute and value	Expected result	Actual result	Result
T-42	To validate that a weapon threat detected in an open space triggers an alert and notifies authorities	Threat: Weapon, Location: Open Space	Alert is triggered, classified as high priority, and authorities are notified.	Alert triggered and authorities notified as expected.	Pass
T-43	To validate that an unattended bag detected in a crowded area triggers	Threat: Unattended Bag, Location: Crowded Area	Alert is triggered, but the system does not classify it as high priority or notify authorities	Alert triggered without high priority classification or authority notification.	Pass

	an alert but does not notify authorities				
--	--	--	--	--	--

Integration Testing 2: Threat Detection and Database Storage

Testing Objective: To ensure seamless integration between threat detection and the system's database for threat storage.

No.	Test case/Test script	Attribute and value	Expected result	Actual result	Result
T-42	To validate that a weapon threat is stored in the database correctly	Threat: Weapon, Location: Open Space	The weapon detection data should be saved in the database with all relevant details (e.g., location, time).	Weapon threat data saved correctly in the database.	Pass
T-43	To validate that an unattended bag threat is stored in the database correctly	Threat: Unattended Bag, Location: Crowded Area	The unattended bag threat data should be saved in the database, marking it as lower priority and no authority notification required.	Unattended bag data saved with appropriate status in the database.	Pass

Integration Testing 3: Alert System and Notification System

Testing Objective: To ensure seamless integration between the alert system and the notification system (email/SMS notifications).

No.	Test case/Test script	Attribute and value	Expected result	Actual result	Result
T-42	To validate that authorities are notified immediately when a weapon threat is detected	Threat: Weapon, Location: Open Space	Authorities should be notified instantly via email/SMS upon detection of a weapon..	Authorities notified immediately via email/SMS.	Pass
T-43	To validate that an unattended bag detected in a crowded area triggers an alert but does not notify authorities	Threat: Unattended Bag, Location: Crowded Area	No notification should be sent for unattended bag threats in a crowded area.	No notification sent for unattended bag detection.	Pass

Integration Testing 4: Data Storage and Data Retrieval

Testing Objective: To ensure seamless integration between data storage (database) and data retrieval for threat details.

No.	Test case/Test script	Attribute and value	Expected result	Actual result	Result
T-42	To validate that weapon threat data can be retrieved correctly from the database	Threat: Weapon, Location: Open Space	Weapon threat data should be retrievable from the database, including all relevant information (e.g., location, time).	Weapon threat data retrieved correctly from the database.	Pass
T-43	To validate that unattended bag threat data can be retrieved correctly from the database	Threat: Unattended Bag, Location: Crowded Area	Unattended bag threat data should be retrievable from the database, marking it as a low-priority threat.	Unattended bag threat data retrieved from the database successfully.	Pass

Integration Testing 5: Threat Detection and Alert System

Testing Objective: To ensure seamless integration between threat detection and alert notification system.

No.	Test case/Test script	Attribute and value	Expected result	Actual result	Result
T-42	To validate that a real-time alert is triggered and displayed on the user interface when a weapon threat is detected	Threat: Weapon, Location: Open Space	Alert is displayed in real-time on the user interface, with a notification of the high priority weapon threat.	Alert displayed on UI in real-time, with priority notification.	Pass
T-43	To validate that a real-time alert is triggered and displayed on the user interface when an unattended bag is detected	Threat: Unattended Bag, Location: Crowded Area	Alert is displayed on the user interface but without high priority classification or authority notification.	Alert displayed on UI without high priority classification or authority notification.	Pass

9. Plagiarism Report

ORIGINALITY REPORT			
5%	2%	2%	4%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS
PRIMARY SOURCES			
1	Submitted to Higher Education Commission Pakistan Student Paper	2%	
2	Submitted to Colorado Technical University Online Student Paper	1%	
3	Submitted to University of Maryland, Global Campus Student Paper	<1%	
4	Submitted to University of Alabama Student Paper	<1%	
5	Submitted to Manipal University Student Paper	<1%	
6	Daniel Birman, Kenneth J. Yang, Steven J. West, Bill Karsh, Yoni Browning, Joshua H. Siegle, Nicholas A. Steinmetz. "Pinpoint: trajectory planning for multi-probe electrophysiology and injections in an interactive web-based 3D environment", Cold Spring Harbor Laboratory, 2023 Publication	<1%	

7	www.shure.com Internet Source	<1%
8	Ervin Varga. "Practical Data Science with Python 3", Springer Science and Business Media LLC, 2019 Publication	<1%
9	Takako Nakatani, Kazuaki Sato, Osamu Shigo. "Traverser: A method and a tool to extract user's traverses within web systems from ontology", Intelligent Decision Technologies, 2023 Publication	<1%
10	scholarworks.sjsu.edu Internet Source	<1%