**Logo, company name

Description automatically generated**

**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

2. Design Methodology and Software Process Model 1

3. System Overview 1

3.1 Architectural Design 1

4. Design Models 1

5. Data Design 2

5.1 Data Dictionary 2

6. Human Interface Design 2

6.1 Screen Images 2

6.2 Screen Objects and Actions 2

7. Implementation 3

7.1 Algorithm 3

7.2 External APIs/SDKs 5

7.3 User Interface 5

7.4 Deployment 8

8. Testing and Evaluation 9

8.1 Unit Testing 9

8.2 Functional Testing 9

8.3 Business Rules Testing 10

8.4 Integration Testing 10

Appendix A 11

Appendix B 13

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\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# 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

## 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.

## 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.

## 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.

## 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.

## 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.

## 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.

## 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.

## 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.

## 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.

## 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.

## 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

# 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 1 Modified Incremental Model**

# 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.

A diagram of a system

Description automatically generated

**Figure 1 Modified Incremental Model**

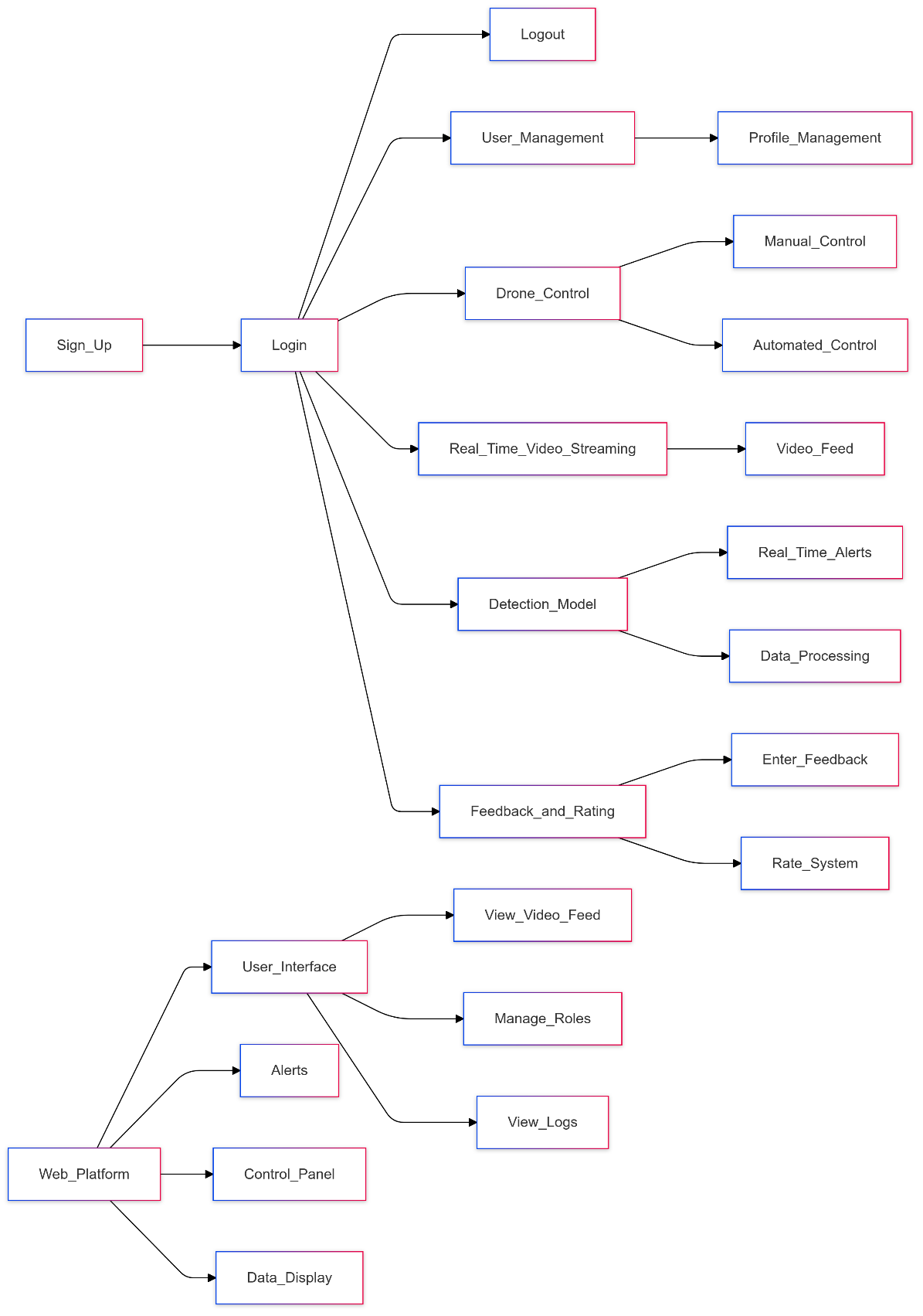
## Architectural Design

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.

  
  
**Block Line Diagram: Mihawk**

# 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.

## Activity Diagram

**Module 1: Drone Flight System**

A diagram of a drone

Description automatically generated

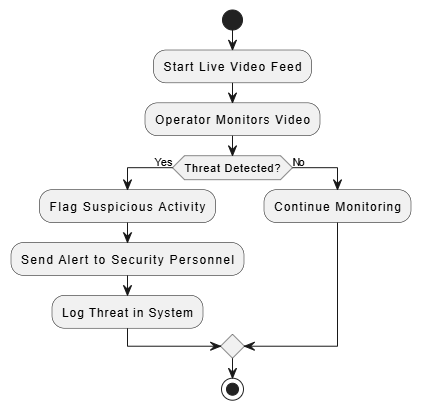
**Figure 1 Modified Incremental Model**

A diagram of a flight process

Description automatically generated

**Figure 1 Emergency landing**

**Module 2: Surveillance and Threat Detection**



**Figure 1 Surveillance and Threat Detection**

**Module 3: Alert Management**

A diagram of a system

Description automatically generated

**Figure 1 Alert Management**

**Module 4: User Management**

A diagram of a system

Description automatically generated

**Figure 1** User Management

**Module 5: Surveillance Monitoring Interface**

A flowchart of a video surveillance system

Description automatically generated

**Figure 1 Surveillance Monitoring Interface**

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

A diagram of a drone

Description automatically generated

**Figure 1 Interactive Mapping and Location Visualization**

**Module 7: Data Handling**

A diagram of a data flow

Description automatically generated

**Figure 1 Data Handling**

**Module 8: Reporting and Analysis**

A diagram of a security system

Description automatically generated

**Figure 1 Reporting and Analysis**

**Module 9: Admin Dashboard**

A diagram of a system

Description automatically generated

**Figure 1 Admin Dashboard**

## Module 10: Raspberry Pi Integration

A diagram of a system

Description automatically generated

**Figure 1** R**aspberry Pi Integration**

## Data Flow Diagram (DFD)

### DFD Level 0:

A diagram of a system

Description automatically generated

**Figure 1 Context Diagram**

### DFD Level 1:

A diagram of a diagram

Description automatically generated

**Figure 1 System Processes**

### DFD Level 2:

### A diagram of a diagram Description automatically generated

**Figure 1 DFD Threat Detection Process**

### A diagram of a process Description automatically generated

**Figure 1 DFD Drone Monitoring Process**

A diagram of a software process

Description automatically generated

**Figure 1 Detailed User Management Process**

## State Transition Diagram

### Drone State:

A diagram of a flight process

Description automatically generated

**Figure 1 Modified Incremental Model**

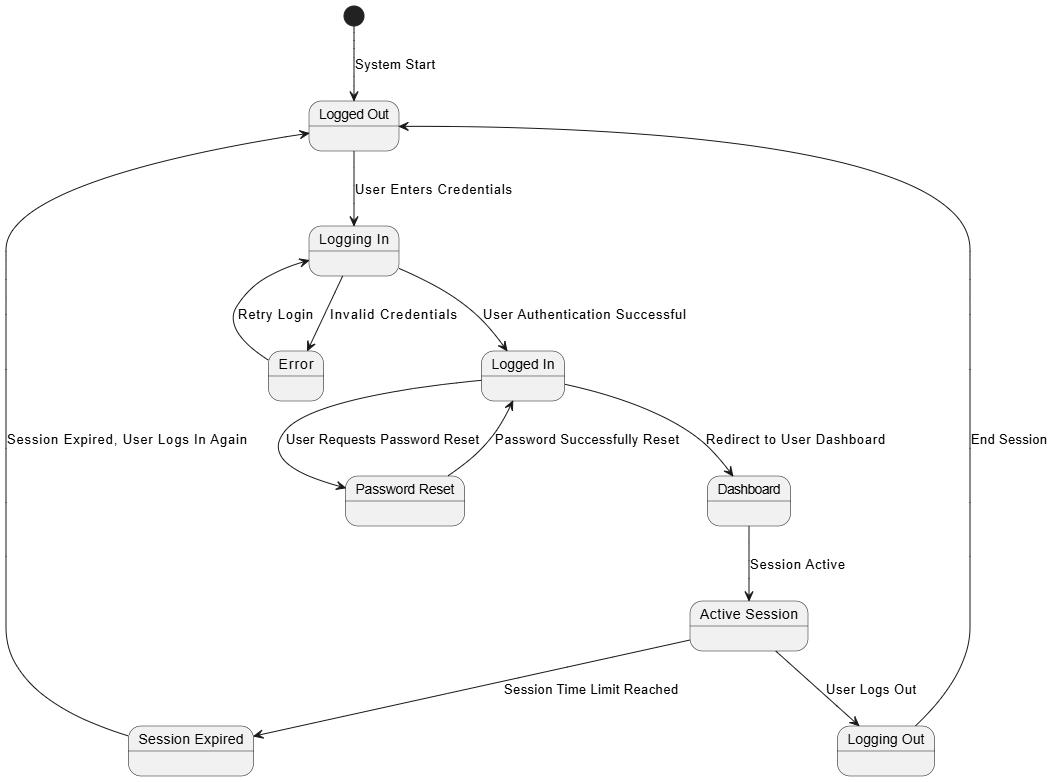
## 4.42 Alert Lifecyle

A diagram of a system

Description automatically generated

**Figure 1 Modified Incremental Model**

## 4.43 User States



**Figure 1 Modified Incremental Model**

# Data Design

Explain how the information domain of your system is transformed into data structures. Describe how the major data or system entities are stored, processed, and organized.

List any databases or data storage items.

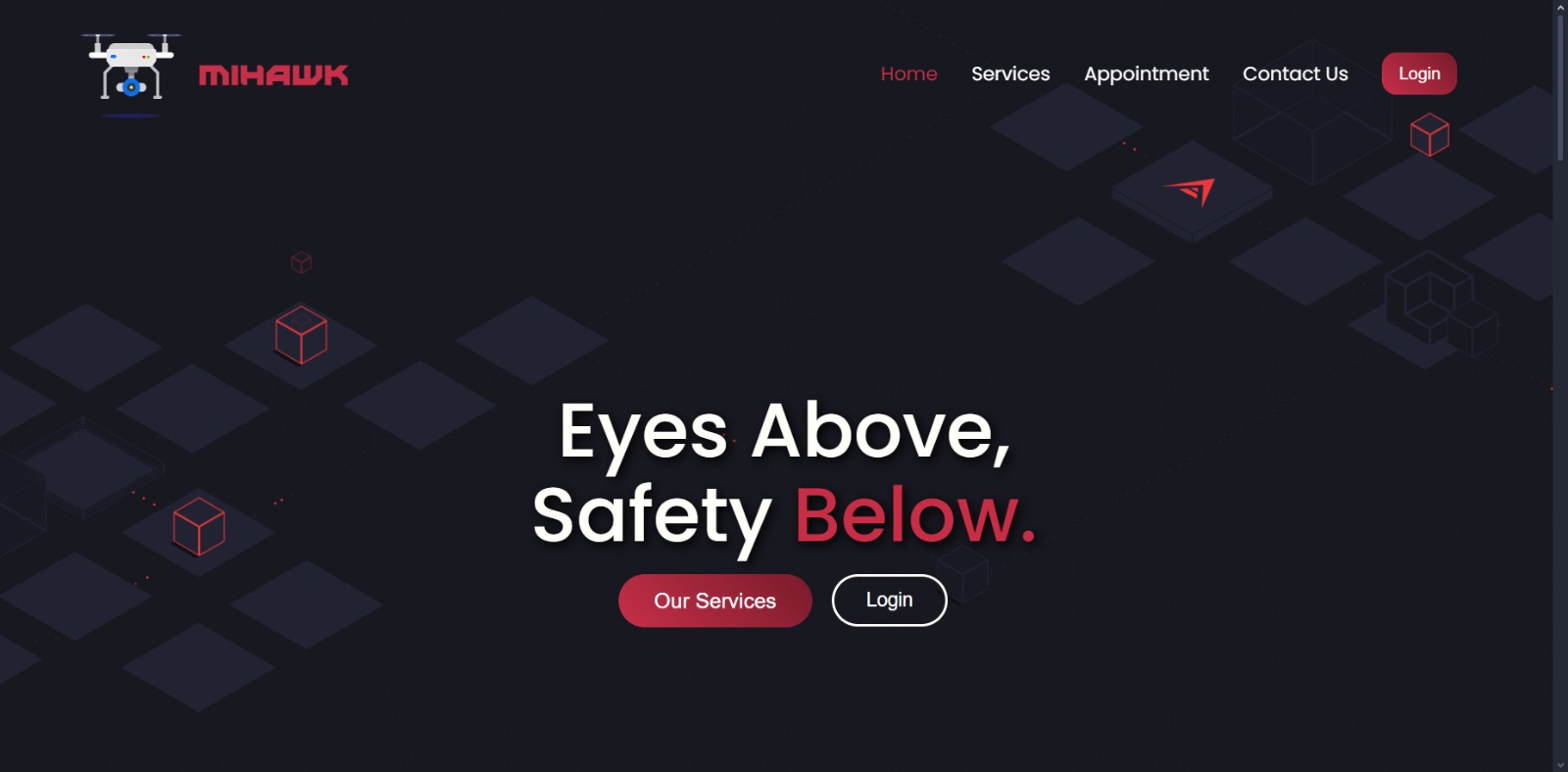
## Data Dictionary

Alphabetically list the system entities or major data along with their types and descriptions. If you provided a functional description, list all the functions and function parameters. If you provided an OO description, list the objects and its attributes, methods and method parameters.

# 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.

## Screen Images



A screenshot of a web page

Description automatically generated

A screenshot of a appointment

Description automatically generated

A screenshot of a contact us

Description automatically generated

A screenshot of a login form

Description automatically generated

A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

## Screen Objects and Actions

A discussion of screen objects and actions associated with those objects

# Implementation

This chapter will discuss implementation details of the project. You will not put your source code here, however, are required to write the core modules functionalities in pseudocode form (Following sections are required in this chapter).

Note: You are required to follow proper coding standard to write your source code. For guidelines, **General Coding Standards & Guidelines** are provided in Appendix D.



## Algorithm

Mention the algorithm(s) used in your project to get the work done with regards to major modules. Provide a pseudocode explanation regarding the functioning of the core features. Be sure to use the correct syntax and semantics for algorithm representations. Following are few examples of algorithms/pseudocode:

|  |  |  |
| --- | --- | --- |
| **Algorithm 1: ManualDroneFlightControl** | | |
| **Input:** Current drone position, user inputs for movement (direction, speed) | | |
| **Output:** Drone reaches each waypoint in sequence (manual control) | | |
| 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  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)** | | |
| 1Algorithm: 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  12: Continue streaming until drone operation ends  13: Return success or error status | 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: Return success or error status | |



|  |  |  |
| --- | --- | --- |
| **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 ofsimilar 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 | |
|  |  | |

|  |  |  |
| --- | --- | --- |
| 1. 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: VideoStreaming | | |
| Input: Video feed, available bandwidth, video resolution | | |
| Output: Streamed video with adaptive resolution | | |
| 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  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 ← 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 | |

|  |  |  |  |
| --- | --- | --- | --- |
| 1. 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 |  | | |
|  |  | | |

## External APIs/SDKs

Describe the third-party APIs/SDKs used in the project implementation in the following table. Few examples of APIs are provided in the table.

**Table 1 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** |
| Stripe (version 2020-08-27) | Credit Card payment integration | Sandbox used for the orders payment | stripe.paymentMethods.create |
| Cloudinary | Image and Video management solution | Uploading Product Images on Cloudinary server | https://api.cloudinary.com/v1\_1/demo/image/upload |

## User Interface

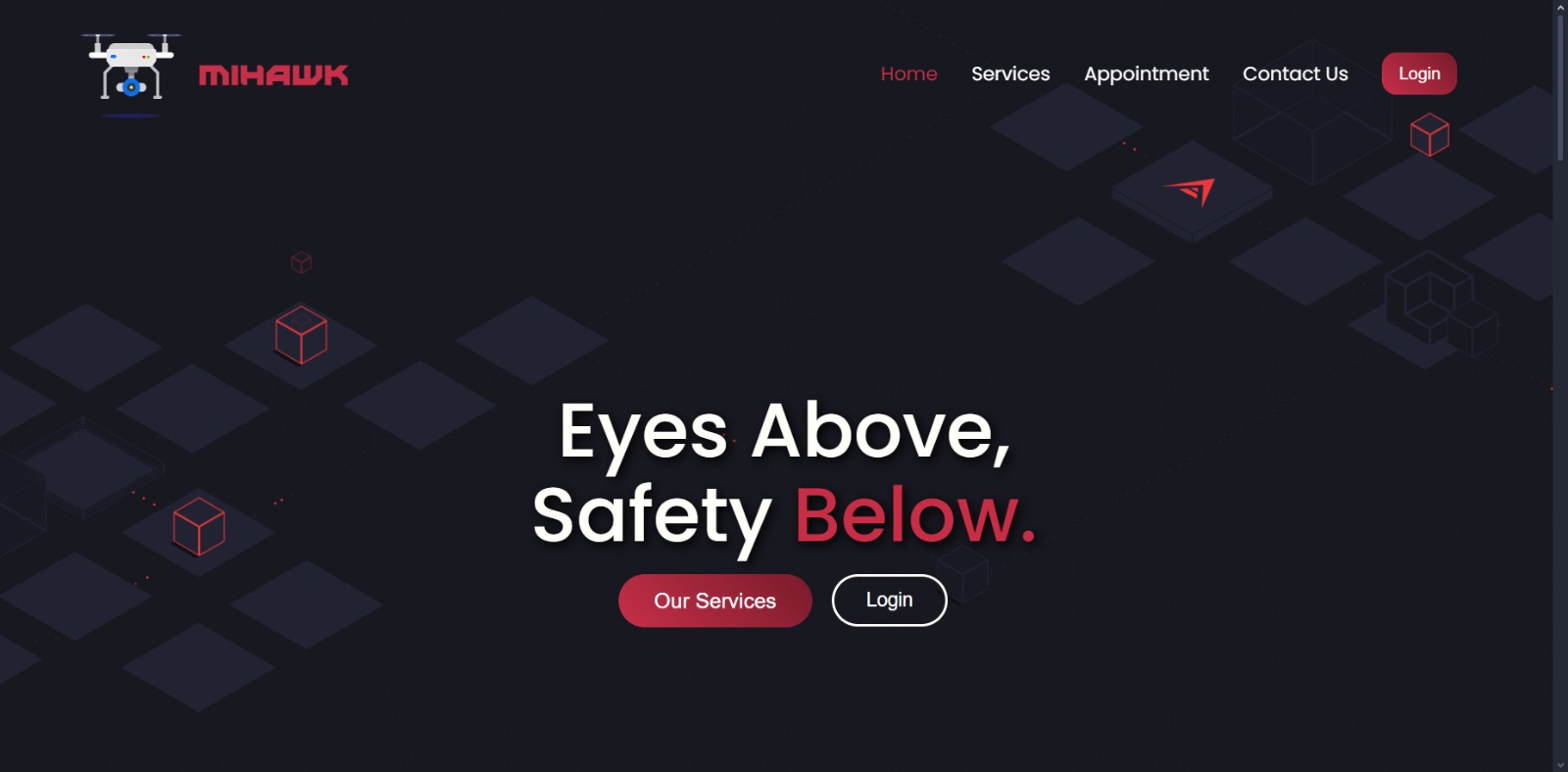
Details about user interface with descriptions. Provide the User Interface for each sub-system (such as Mobile App, Web App, Client App, Admin App). Provide description of each User Interface explaining the details.

When inserting User Interfaces, use appropriate size of the image, for example, for mobile app, 2-4 screens can be placed on a single page.

Following are few examples of User Interfaces:

* + 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 1 Landing Page**

* + 1. **Services Page**

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

**A screenshot of a web page

Description automatically generated**

**Figure 2 Services Page**

* + 1. **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.

A screenshot of a appointment

Description automatically generated

Figure 3 Book an Appointment

### 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.

A screenshot of a contact us

Description automatically generated

**Figure 4 Contact Us**

### Login Screen

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

A screenshot of a login form

Description automatically generated

**Figure 4 Login**

### Manage User

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

A screen shot of a computer

Description automatically generated

**Figure 4 Manage User**

### 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.

A screenshot of a computer

Description automatically generated

**Figure 4 Create User**

## Deployment

Specify the deployment environments used for hosting and live testing of all the sub-systems of the project. Provide the details of hosting/cloud service used, set of available software and their versions used etc.

# Testing and Evaluation



Once the system has been successfully developed, testing has to be performed to ensure that the system working as intended. This is also to check that the system meets the requirements stated earlier. Besides that, system testing will help in finding the errors that may be hidden from the user. The testing must be completed before it is deployed for use.

There are few types of testing which includes the unit testing, functional testing and integration testing.

You are *required* to perform each of these in-depth to ensure system quality.

## 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](mailto: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](mailto: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 Dashboard and see a "Permission Denied" message. | Pass |
| 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 |

## 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 |

## 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 |

## 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 an alert but does not notify authorities | 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 |

**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 |