2024

# PROJECT 381 : MILESTONE 3 DRONE PROJECT GROUP 5 SOFTWARE DOCUMENTION


1. Tshepang Kagiso Mashigo : 578012
2. Tiaan Theron:  577856
3. Tiaan Kritzinger: 577643
4. Traver Mhere: 576809
5. Tinashe Ndawe: 577936
6. Thubelihle Nkuna: 577200

# Table of Contents


SKYLYNX

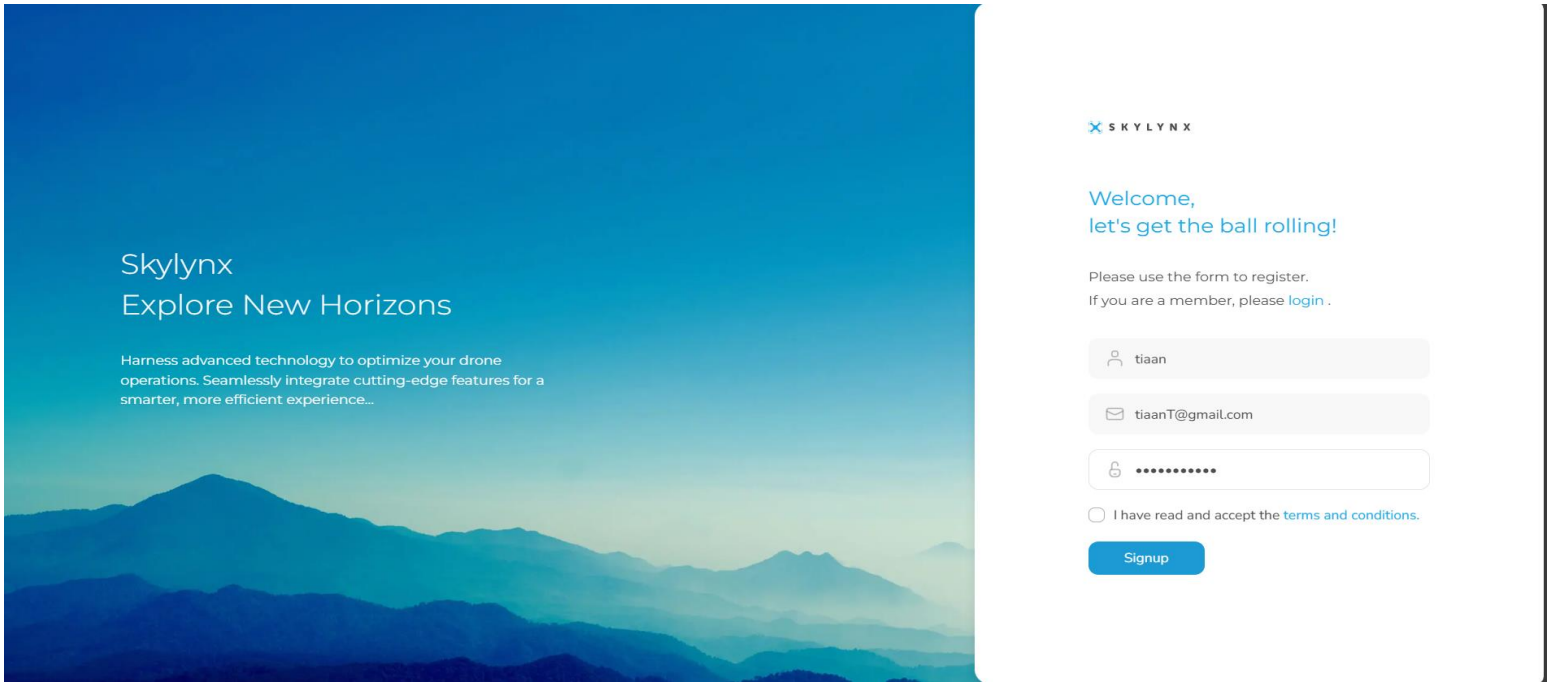
Welcome,
let's get started!

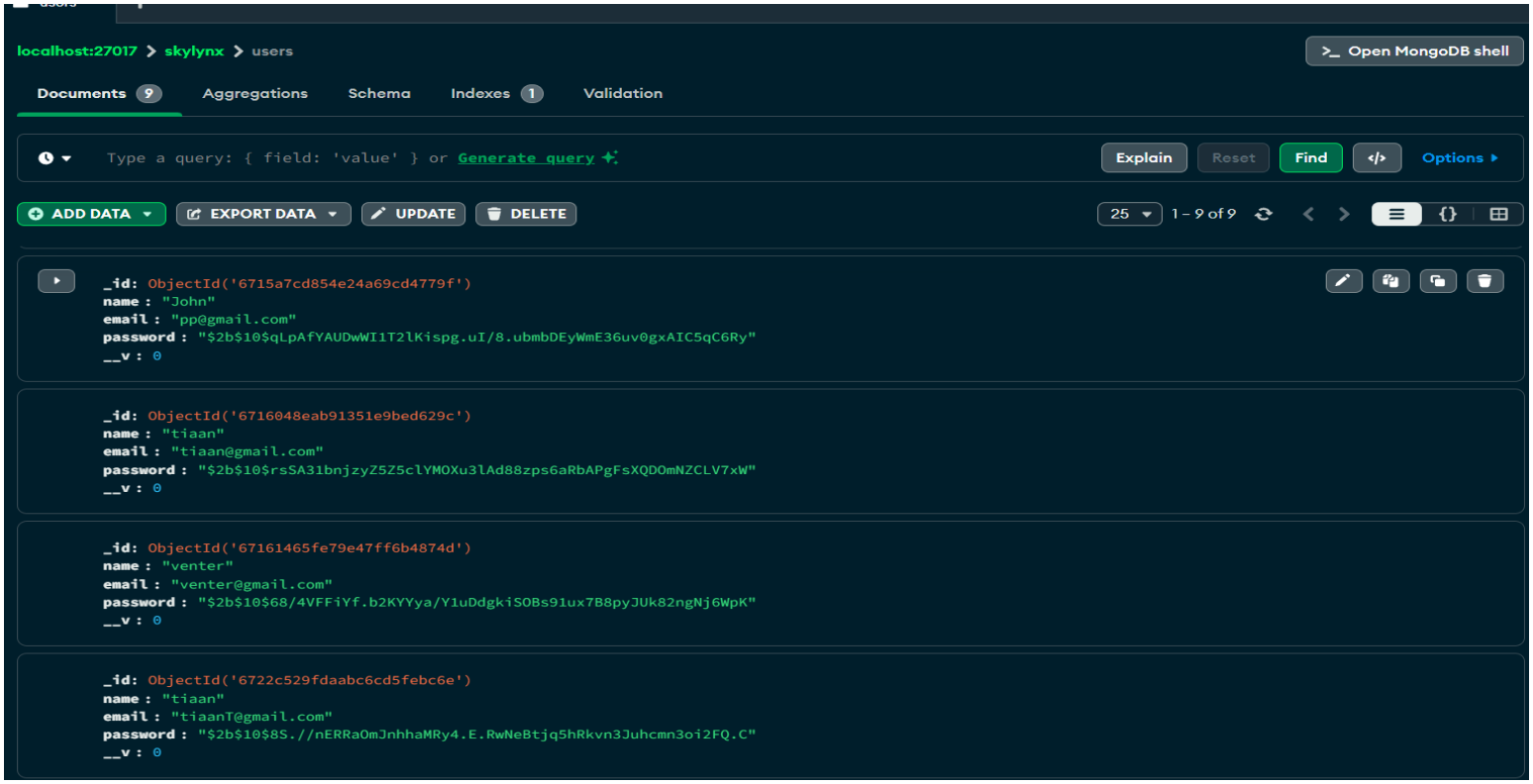
Please use your credentials to login.
If you are not a member, please register .

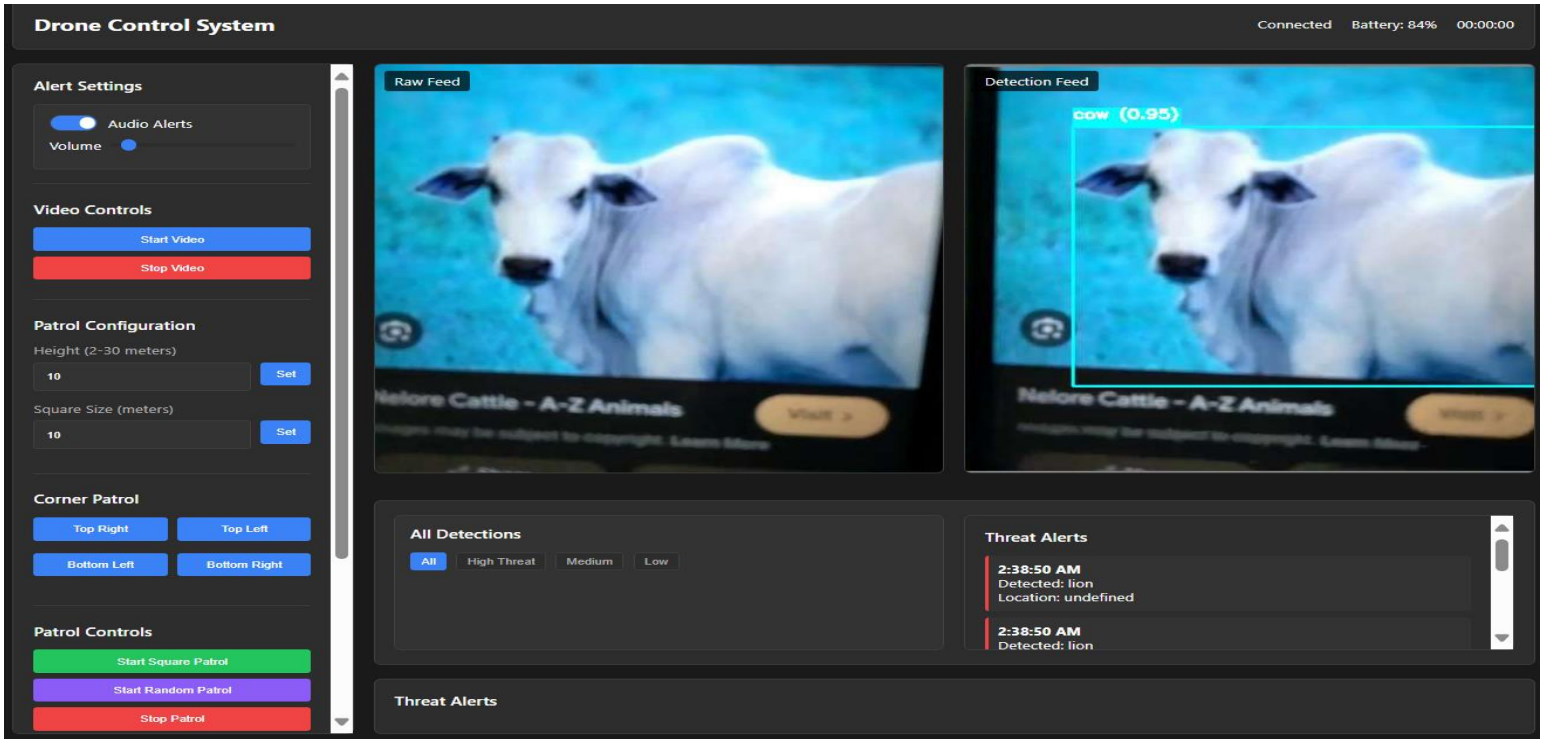
tiaanT@gmail.com

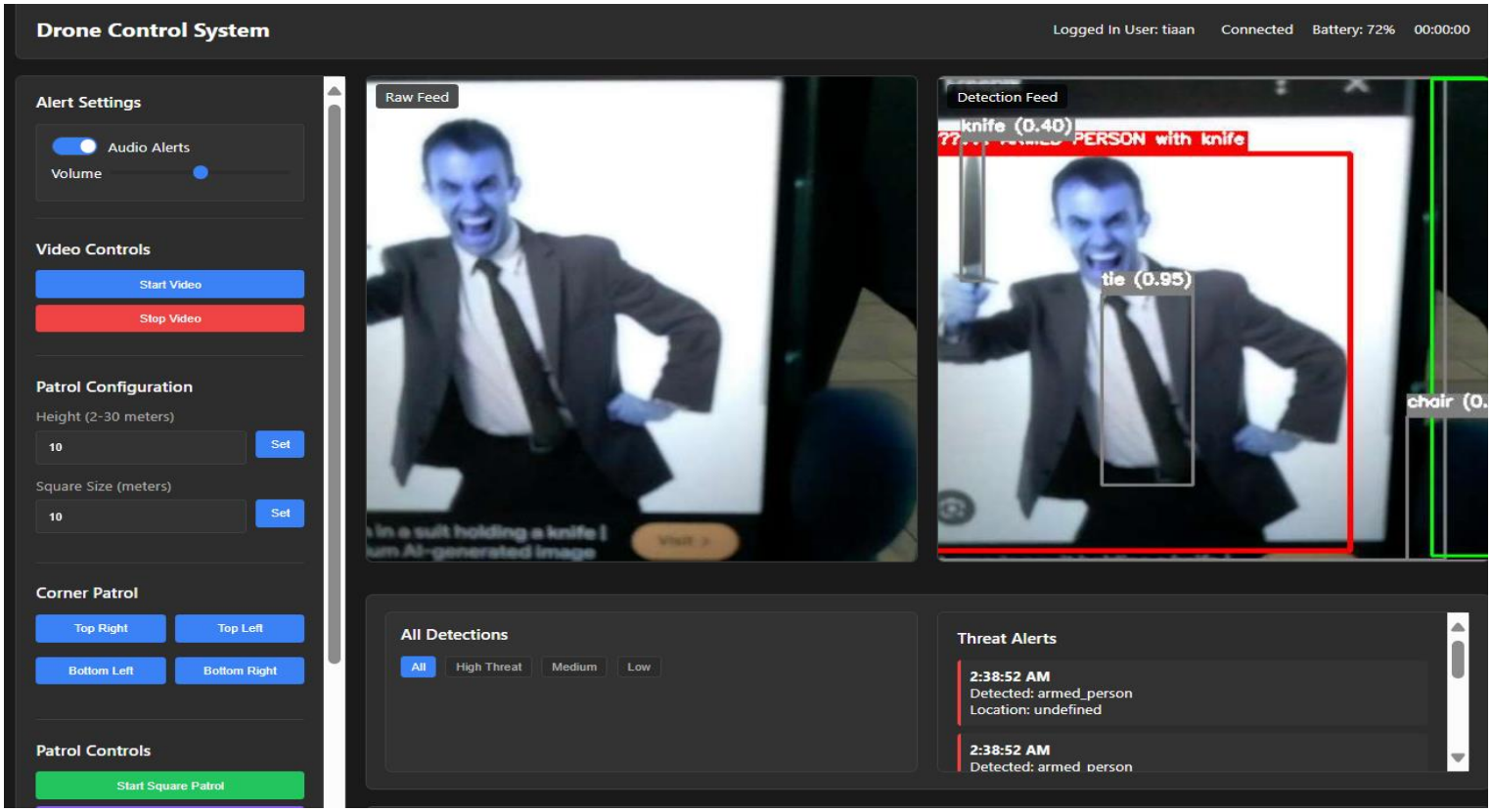
••••••••••

Login


Skylynx
Explore New Horizons
Harness advanced technology to optimize your drone operations. Seamlessly integrate cutting-edge features for a smarter, more efficient experience...
SKYLYNX
Welcome,
let's get the ball rolling!
Please use the form to register.
If you are a member, please login .
tiaan
tiaanT@gmail.com
I have read and accept the terms and conditions.
Signup


localhost:27017 > skylynx > users
Open MongoDB shell
Documents 9
Aggregations
Schema
Indexes 1
Validation
Type a query: { field: 'value' } or Generate query
Explain
Reset
Find
Options
ADD DATA
EXPORT DATA
UPDATE
DELETE
25
1 - 9 of 9
_id: ObjectId('6715a7cd854e24a69cd4779f')
name : "John"
email : "pp@gmail.com"
password : "$2b$10$qLpAfYAUDwWI1T2lKispg.uI/8.ubmbDEyWmE36uv0gxAIC5qC6Ry"
__v : 0
_id: ObjectId('6716048eab91351e9bed629c')
name : "tiaan"
email : "tiaan@gmail.com"
password : "$2b$10$rsSA31bnjzyZ5Z5clYMOXu3lAd88zps6aRbAPgFsXQDOmNZCLV7xW"
__v : 0
_id: ObjectId('67161465fe79e47ff6b4874d')
name : "venter"
email : "venter@gmail.com"
password : "$2b$10$68/4VFFiYf.b2KYYya/Y1uDdgkiSOBs91ux7B8pyJUk82ngNj6WpK"
__v : 0
_id: ObjectId('6722c529fdaabc6cd5febc6e')
name : "tiaan"
email : "tiaanT@gmail.com"
password : "$2b$10$8S.//nERRaOmJnhhaMRy4.E.RwNeBtjq5hRkvn3Juhcmn3oi2FQ.C"
__v : 0


Drone Control System
Connected
Battery: 84%
00:00:00
Alert Settings
Audio Alerts
Volume
Video Controls
Start Video
Stop Video
Patrol Configuration
Height (2-30 meters)
10
Set
Square Size (meters)
10
Set
Corner Patrol
Top Right
Top Left
Bottom Left
Bottom Right
Patrol Controls
Start Square Patrol
Start Random Patrol
Stop Patrol
Raw Feed
Detection Feed
cow (0.95)
All Detections
All
High Threat
Medium
Low
Threat Alerts
2:38:50 AM
Detected: lion
Location: undefined
2:38:50 AM
Detected: lion
Threat Alerts


Drone Control System
Logged In User: tiaan
Connected
Battery: 72%
00:00:00
Alert Settings
Audio Alerts
Volume
Video Controls
Start Video
Stop Video
Patrol Configuration
Height (2-30 meters)
10
Set
Square Size (meters)
10
Set
Corner Patrol
Top Right
Top Left
Bottom Left
Bottom Right
Patrol Controls
Start Square Patrol
Raw Feed
Detection Feed
knife (0.40)
PERSON with knife
tie (0.95)
chair (0.
All Detections
All
High Threat
Medium
Low
Threat Alerts
2:38:52 AM
Detected: armed_person
Location: undefined
2:38:52 AM
Detected: armed_person

# The Security Drone Monitoring System

## 1. System Overview

The Security Drone Monitoring System is a comprehensive solution for automated surveillance using DJI Tello drones. This system integrates real-time video streaming, threat detection, and automated patrol capabilities.

### 1.1. Core Components

1. Drone control and management
2. Video streaming with real-time processing
3. Object detection and threat classification
4. Automated patrol patterns
5. Real-time alerts and monitoring

### 1.2. System Architecture

├── DroneManager: Core drone operations

├── VideoStreamer: Video processing

├── ObjectDetector: Threat detection

├── PatrolSystem: Patrol logic

└── App.py: Main application server

# 2. Class Documentation

## 2.1. DroneManager

**Purpose**: Manages drone operations, connection, and movement control.

**Dependencies**:

- djitellopy
- asyncio
- numpy
- logging

### 2.1.1. Methods

#### connect_drone()

**Input**: None

**Process**:

1. Initializes drone connection
2. Performs initial battery check
3. Sets up video stream
4. **Output**: Boolean (connection success/failure)

**move_to(x: float, y: float, z: float)**

**Input**: Target coordinates in meters

**Process**:

1. Validates movement safety
2. Converts coordinates to drone commands
3. Executes movement sequence
4. **Output**: Boolean (movement success/failure)

**scan_360()**

**Input**: None

**Process**:

1. Performs 360-degree rotation
2. Maintains stable height
3. Executes in 45-degree increments
4. **Output**: Boolean (scan completion status)

## 2.2. VideoStreamer

**Purpose**: Handles video streaming and frame processing.

**Dependencies**:

- OpenCV
- base64
- threading
- socketio

### 2.2.1. Stream Control

#### start_streaming(drone)

**Input**: DJITello instance

**Process**:

1. Initializes streaming thread
2. Sets up frame processing
3. Begins capture loop
4. **Output**: Boolean (stream start status)

### 2.2.2. Frame Processing

**process_frame(frame)**

**Input**: OpenCV frame (numpy.ndarray)

**Process**:

1. Resizes frame to standard size
2. Performs threat detection
3. Encodes frame for streaming
4. **Output**: Processed frame data via socket

## 2.3. ObjectDetector

**Purpose**: Performs threat detection and classification.

**Dependencies**:

- YOLO

- OpenCV

- numpy

### 2.3.1. Detection Methods

#### detect_objects(frame)

**Input**: OpenCV frame (numpy.ndarray)

**Process**:

1. Runs YOLO detection
2. Filters and classifies threats
3. Validates temporal consistency
4. **Output**: List of detected objects with classifications

## 2.4. PatrolSystem

**Purpose**: Manages patrol logic and mission control.

**Dependencies**:

- asyncio

- numpy

- DroneManager

- VideoStreamer

#### 2.4.1. Patrol Control

**start_patrol(patrol_type: str)**

**Input**: Patrol type ('square' or 'random')

**Process**:

1. Initializes mission
2. Generates patrol points
3. Executes movement sequence
4. Performs scans at each point
5. **Output**: Boolean (mission success/failure)

# 3. API Documentation

## 3.1. REST Endpoints

**GET /status**

**Description**: Returns current system status

**Authentication**: Required (JWT)

**Request**: None

**Response**:


```
{
    "connected": true,
    "battery": 85,
    "streaming": true,
    "patrol_status": "idle",
    "current_command": "move_to",
    "queue_size": 0
}
```


### 3.2. Response Formats

All API responses follow the standard format:


```
{
    "success": true,
    "data": {},
    "error": null
    }
```


On error:


```
{
```

```
    "success": false,
    "data": null,
    "error": {
    "code": "ERROR_CODE",
    "message": "Error description"
    }
}
```


### 3.3. Error Handling

Common error codes:

AUTH_001: Authentication failed

DRONE_001: Drone connection failed

PATROL_001: Invalid patrol parameters

STREAM_001: Stream initialization failed

# 4. Socket Events

## 4.1. Connection Events

**connect**

**Purpose**: Establishes socket connection and initializes drone

**Emits**:

```
{
    "data": "Connected to drone",
    "type": "success"
}
```

**disconnect**

**Purpose**: Handles client disconnection and cleanup

**Process**: Stops active operations and cleans up resources

## 4.2. Video Control Events

**start_video**

**Purpose**: Initiates video streaming

**Input**: None

**Emits**:

```
{
        "video_frame": {
        "frame": "base64_encoded_frame",
        "timestamp": 1635789012.45,
        "fps": 20.5
        }
}
```

**detection_frame**

**Purpose**: Streams processed frames with detections

**Emits**:


```
{
        "frame": "base64_encoded_frame",
        "detections": [
                {
                "class": "person",
                "confidence": 0.85,
                "box": [100, 200, 300, 400],
                "threat_level": "high"
                }
        ],
"timestamp": 1635789012.45
}
```


## 4.3. Mission Events

**start_square_patrol**

**Purpose**: Initiates square patrol mission

**Input**:


```
{
    "size": 10.0,
    "height": 15.0
}
```


**Emits Status Updates**:


```
{
    "status": "patrolling",
    "current_point": "Point(x=10.00, y=0.00, z=15.00)",
    "battery": 85,
    "points_covered": 2,
    "total_points": 4
}
```


**start_random_patrol**

**Purpose**: Initiates random point patrol

**Input**: None (uses predefined square size)

**Emits Updates**: Same format as square patrol

## 4.4. Status Updates

**patrol_status**

**Purpose**: Real-time mission status updates

**Frequency**: Every 1 second or on state change

**Format**:


```
{
        "status": "patrolling",
        "mission_id": "20240130_143022",
        "current_position": {
        "x": 10.0,
        "y": 5.0,
        "z": 15.0
},
        "battery": {
        "level": 85,
        "status": "SAFE",
        "estimated_time": 1200
},
```

```
        "metrics": {
        "points_covered": 2,
        "total_points": 4,
        "threats_detected": 0,
        "mission_duration": 120
        }
}
```


**threat_alert**

**Purpose**: Real-time threat detection alerts

**Triggered**: On threat detection

**Format**:


```
{
        "threats": [
        {
                "class": "person",
                "confidence": 0.92,
                "location": "Point(x=10.00, y=0.00, z=15.00)",
                "threat_level": "high",
```

```
                "timestamp": 1635789012.45
            }
    ],
            "mission_context": {
            "patrol_type": "square",
            "current_point": 2,
            "status": "scanning"
            }
    }
```


## Client-Side Implementation Example

```
const socket = io('http://localhost:5000', {
            auth: {
            token: 'JWT_TOKEN'
            }
});

// Connection handling
socket.on('connect', () => {
            console.log('Connected to drone server');
            updateConnectionStatus(true);
```

```
});

socket.on('disconnect', () => {
    console.log('Disconnected from drone server');
    updateConnectionStatus(false);
});

// Video stream handling
socket.on('video_frame', (data) => {
    updateVideoFeed(data.frame);
    updateMetrics(data.fps);
});

// Threat detection handling
socket.on('threat_alert', (data) => {
    displayThreatAlert(data.threats);
    triggerAlarm(data.threats[0].threat_level);
});

// Mission status updates
```

```
socket.on('patrol_status', (data) => {
        updateDashboard(data);
        updateBatteryStatus(data.battery);
        updateMissionProgress(data.metrics);
});
```

# 5. Frontend Server

## 5.1. Express.js Setup

The frontend server runs on Express.js and provides a secure interface for accessing the drone control system.

**Server Configuration**:

```
const express = require('express');
const app = express();
const PORT = 3000;

app.set('view engine', 'ejs');
app.use(express.json());
app.use(express.static('public'));
```

## 5.2. Authentication System

1. JWT-based authentication

2. Session management
3. Route protection
4. Role-based access control

**Authentication Flow**:

Login Request -> Validate Credentials -> Generate JWT -> Set Cookie -> Redirect to Dashboard

## 5.3. Route Protection

Protected routes require valid JWT:

```
// Middleware example
const authMiddleware = (req, res, next) => {
    const token = req.cookies.token;
    if (!token) return res.redirect('/login');
    // Verify token and proceed
};
```

## 5.4. MongoDB Integration

**Database Schema**:

User

├── name

├── email

└── password(hashed)

### 5.5. EJS Templates

Template structure:

views/

├── login.ejs

├── register.ejs

└── dashboard.ejs

# 6. System Requirements

## 6.1. Hardware Requirements

1. **DJI Tello Drone**
2. Fully charged battery
3. Firmware updated

4. Working WiFi capability
5. **Computer System**
6. CPU: Quad-core or better
7. RAM: 8GB minimum
8. WiFi adapter
9. Storage: 10GB free space

## 6.2. Software Dependencies

**Backend Requirements**:

- Python 3.8+
- pip
- OpenCV
- YOLO
- Socket.IO
- asyncio

**Frontend Requirements**:

- Node.js 14+
- npm
- MongoDB 4.0+
- Express.js
- Socket.IO client

## 6.3. Pre-flight Setup

1. **Drone Preparation**:
2. Power on drone
3. Connect to drone's WiFi network
4. Verify drone battery level
5. **Network Configuration**:

**Drone WiFi:** TELLO-XXXXXX

**Backend Server:** localhost:5000

**Frontend Server:** localhost:3000

**MongoDB:** localhost:27017

## 6.4. Startup Sequence

1. **Start Database**:
    a. Mongod

2. **Start Python Backend**:
    a. pip install -r requirements.txt
    b. python app.py

3. **Start Frontend**:
    a. npm install
    b. npm run start

### 6.5. Common Issues

1. **Connection Issues**:
2. Verify drone WiFi connection
3. Check all ports are available
4. Ensure MongoDB is running
5. **Video Stream Issues**:
6. Verify camera access
7. Check bandwidth availability
8. Monitor system resources

# 7. Client Integration

## 7.1. Socket Event Handling

```
// Initialize socket connection

const socket = io('http://localhost:5000');



// Handle connection events

socket.on('connect', () => {

	console.log('Connected to drone server');

});
```

```
// Handle video streams
socket.on('video_frame', (data) => {
        const rawFeed = document.getElementById('raw-feed');
        rawFeed.src = `${data.frame}`;
});
```

### 7.2. Status Display Functions

```
function updateMissionStatus(status) {
        document.getElementById('patrol-status').textContent = status.status;
        document.getElementById('battery-level').textContent = `${status.battery}%`;
}
```

### 7.3. Video Feed Implementation

```
function initializeVideoFeeds() {
        const rawFeed = document.getElementById('raw-feed');
        const detectionFeed = document.getElementById('detection-feed');
        // Setup feed refresh rate
        setInterval(() => {
                if (isStreaming) {
                requestNewFrame();
```

```
                }
        }, 50); // 20 FPS
}
```

### 7.4. Error Handling

```
socket.on('error', (error) => {
        console.error('Socket error:', error);
        showErrorNotification(error.message);
});

function showErrorNotification(message) {
        // Display error to user
        const notification = document.getElementById('notification');
        notification.textContent = message;
        notification.classList.add('error');
}
```

### 7.5. Authentication Integration

```
async function login(credentials) {
try {
        const response = await fetch('/auth/login', {
        method: 'POST',
        headers: {
        'Content-Type': 'application/json'
        },
        body: JSON.stringify(credentials)
        });
        if (response.ok) {
        window.location.href = '/dashboard';
        }
} catch (error) {
        showErrorNotification('Login failed');
        }
}
```

# 8. Installation Guide

## 8.1. Database Setup

# Install MongoDB

```
sudo apt-get install mongodb
```

# Start MongoDB service

```
sudo service mongodb start
```

# Verify MongoDB is running

```
mongo --eval 'db.runCommand({ connectionStatus: 1 })'
```

## 8.2. Server Configuration

**Environment Variables**:

```
DRONE_SERVER_PORT=5000
FRONTEND_SERVER_PORT=3000
MONGODB_URI=mongodb://localhost:27017/security_drone
JWT_SECRET=your_secret_key
```

## 8.3. Environment Variables

Create `.env` file in project root:

```
NODE_ENV=production
JWT_SECRET=your_secret_key
MONGO_URI=mongodb://localhost:27017/security_drone
DRONE_SERVER_URL=http://localhost:5000
```

## 8.4. Starting Services

**Complete Startup Script**:

```
#!/bin/bash

# Start MongoDB
sudo service mongodb start

# Start Python backend
python app.py

# Start Frontend
npm run start
```

### 8.5. Verification Steps

1. Check Services:

```
# Check MongoDB
    mongo --eval "db.stats()"


# Check Python server

    curl http://localhost:5000/status


# Check Frontend server

    curl http://localhost:3000/login
```

# 9. Troubleshooting

### 9.1. Common Problems

1. **Drone Connection Issues**
2. Symptom: Cannot connect to drone
3. Solution: Verify WiFi connection
4. Check drone battery level
5. **Video Stream Issues**
6. Symptom: Black screen or lag
7. Solution: Check network bandwidth
8. Verify system resources

### 9.2. Error Messages

ERROR_001: Drone connection failed

- Check WiFi connection
- Verify drone power

ERROR_002: Stream initialization failed

- Check system resources
- Verify camera access

ERROR_003: Database connection failed

- Check MongoDB service
- Verify connection string

### 9.3. System Logs

Log locations:

```
/logs/
├── system/
│   ├── error.log
│   └── info.log
├── drone/
```

```
│   ├── flight.log
│   └── battery.log
└── missions/
    └── patrol.log
```

### 9.4. FAQ

1. **Q: How do I reset the drone connection?**
2. A: Restart drone and reconnect to WiFi
3. **Q: What causes video lag?**
4. A: Network bandwidth or processing power limitations

# 10. Maintenance

### 10.1. Backup Procedures

```
# Database backup
    mongodump --db security_drone --out /backup/

# Log rotation
    logrotate /etc/logrotate.d/security_drone
```

### 10.2. Updates

1. System Updates:

```
# Backend updates
    pip install --upgrade -r requirements.txt


# Frontend updates
    npm update
```

### 10.3. Performance Monitoring

```
// Monitor system metrics
{
    "cpu_usage": "<percentage>",
    "memory_usage": "<percentage>",
    "disk_space": "<available_gb>",
    "network_bandwidth": "<mbps>"
}
```

### 10.4. Security Measures

1. Regular security audits
2. JWT token rotation
3. SSL/TLS encryption
4. Rate limiting

# 11. Conclusion

## 11.1. System Overview

The Security Drone Monitoring System provides:

1. Automated surveillance
2. Real-time threat detection
3. Secure access control
4. Comprehensive monitoring

## 11.2. Best Practices

1. **Safety**:
2. Maintain visual contact
3. Monitor battery levels
4. Follow regulations
5. Regular system checks
6. **Operation**:
7. Pre-flight checks
8. Regular maintenance
9. Log monitoring
10. Update procedures

## 11.3. Limitations

1. Battery life constraints
2. Weather dependencies
3. WiFi range limitations

4. Processing requirements

### 11.4. Future Development

1. Enhanced detection models
2. Multi-drone support
3. Mobile applications
4. AI improvements
5. Extended battery life

### 11.5. Support

Contact for support:

1. System logs review
2. Documentation reference
3. Technical assistance
4. Custom implementations

# 12. Appendices

### 12.1. JSON Examples

**Patrol Status**:


```
{
    "status": "active",
    "battery": 85,
    "position": {
        "x": 10,
```

```
            "y": 20,
            "z": 15
        }
}
```

## 12.2. Error Codes

Complete list of system error codes:

1. 1000-1999: Connection Errors
2. 2000-2999: Video Stream Errors
3. 3000-3999: Patrol Errors
4. 4000-4999: Authentication Errors

## 12.3. Configuration Templates

Example configuration file:

```
{
    "drone": {
        "max_height": 30,
        "default_speed": 50,
        "battery_warning": 20
    },
```

```
    "video": {
        "resolution": "720p",
        "framerate": 30,
        "quality": "high"
        }
}
```

### 12.4. Log Formats

Standard log entry format:

```
[TIMESTAMP] [LEVEL] [MODULE] Message

2024-01-30 14:30:22 INFO DRONE Connected successfully
```

# Final Conclusion

## System Summary

The Security Drone Monitoring System represents a significant advancement in automated surveillance technology. By combining DJI Tello drone capabilities with advanced computer vision, secure web technologies, and robust authentication systems, this solution provides a comprehensive platform for security monitoring operations.

## Key Achievements

1. **Integration Success**
2. Seamless combination of Python drone control with Node.js frontend
3. Real-time video processing and threat detection

4. Secure user authentication and access control
5. Automated patrol systems with intelligent battery management
6. **Technical Innovation**
7. Multi-server architecture (Python backend, Node.js frontend)
8. Real-time video processing with YOLO detection
9. Socket.IO-based live streaming
10. JWT-secured API endpoints
11. **Security Features**
12. Multi-level threat detection
13. Real-time alerting system
14. Secure authentication
15. Comprehensive logging

# Operational Highlights

1. **Automated Patrols**: Systematic coverage with intelligent path planning
2. **Real-time Processing**: Immediate threat detection and notification
3. **User Interface**: Intuitive dashboard for monitoring and control
4. **Data Management**: Structured logging and threat data storage

# System Benefits

1. Reduced manual monitoring requirements
2. Enhanced threat detection capabilities
3. Automated reporting
4. Comprehensive system oversight
5. Resource optimization
6. **Organization**
7. Improved security coverage
8. Cost-effective solution
9. Scalable architecture

# Future Outlook

The system provides a strong foundation for future enhancements:

1. AI-powered decision making
2. Extended detection capabilities
3. Multi-drone coordination
4. Mobile application integration
5. Enhanced analytics and reporting

# Final Thoughts

The Security Drone Monitoring System demonstrates the potential of combining drone technology with modern web architecture and AI capabilities. Its modular design, secure implementation, and user-friendly interface make it a valuable tool for security operations.

# Support and Maintenance

For ongoing support and system maintenance:

1. Regular system updates
2. Performance monitoring
3. Security patches
4. Technical support