AI-Based Crowd Monitoring and Safety Management System

Internship Project Report

Author: Tushar Dayma
Mentor: Umme Asma

Training Dates: 22 September 2025 – 21 November 2025

Abstract

This project report details the design, development, and implementation of an "AI-Based Crowd Monitoring and Safety Management System." This system automates real-time people counting, tracking, and monitoring, addressing the inefficiencies and inaccuracies of manual supervision.

Leveraging a tech stack of Flask, YOLOv8, OpenCV, and PostgreSQL, the application provides a full-stack solution. The system ingests video feeds and uses the YOLOv8 object detection model with a ByteTrack tracker to identify and assign unique IDs to individuals. A key feature is the ability to define interactive "Danger Zones," with the system automatically monitoring the dwell time of each person within these areas.

A 3-tier alert mechanism triggers for breaches in per-person dwell time, zone-specific population, and overall crowd density. All data, alerts, and user credentials are managed in a robust PostgreSQL database. The frontend features a secure, role-based login system (User and Admin) using JWT, a live analytics dashboard with Chart.js, and a full admin panel for system configuration and user management. The system is capable of generating downloadable PDF and CSV reports for historical analysis. This report covers the complete system architecture, implementation details, database design, and results.

Keywords: Crowd Monitoring, Computer Vision, YOLOv8, Flask, PostgreSQL, Real-time Analytics, Object Tracking, Dwell Time Analysis.

Acknowledgements

I would like to express my sincere gratitude to my mentor, **Umme Asma**, for her invaluable guidance, support, and encouragement throughout this internship project. Her insights and feedback were instrumental in the successful development and completion of this system.

I am also thankful to the **Infosys Springboard** program for providing the opportunity and resources to work on this challenging and meaningful project.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Project Background and Motivation

Traditional crowd monitoring in public spaces such as malls, airports, stadiums, and transportation hubs relies heavily on manual supervision by security personnel. This approach is often inefficient, prone to human error, and lacks the capability for real-time data collection and analysis. In high-density environments, the delay in identifying potential safety risks, such as overcrowding or unauthorized access to restricted areas, can have severe consequences.

The advent of powerful computer vision models and accessible web frameworks presents an opportunity to automate and enhance crowd management. An intelligent system can tirelessly monitor multiple video feeds, quantify crowd behavior, and provide immediate alerts, transforming security from a reactive to a proactive discipline.

1.2 Problem Statement

Manual crowd monitoring is inherently limited. It is not scalable, suffers from operator fatigue, and is subjective. This results in several key challenges:

- **Delayed Response:** It is difficult for human operators to monitor all areas simultaneously, leading to delays in responding to overcrowding or security breaches.
- **Inaccurate Data:** Manual crowd counting is error-prone, making it difficult to gather accurate statistics for operational planning or safety compliance.
- **Lack of Analytics:** Manual systems cannot easily track metrics like dwell time, footfall patterns, or historical trends, which are vital for optimizing space and security protocols.
- **High Cost:** Effective manual monitoring requires a large number of personnel, incurring significant operational costs.

There is a clear need for an automated, accurate, and intelligent system that can detect, track, and count people in real-time, monitor specific zones, and trigger configurable alerts to enhance safety and operational efficiency.

1.3 Project Objectives

The primary goal of this project is to design and develop a comprehensive AI-Based Crowd Monitoring and Safety Management System. The specific objectives are:

- **Automate Detection and Tracking:** To utilize the YOLOv8 model to detect and track each individual in a live or recorded video feed with a persistent, unique ID.
- **Implement Zone-Based Monitoring:** To allow users to interactively define "Danger Zones" or "Restricted Areas" on the video feed and monitor the precise time each person spends within them.

  To create a heatmap visualization overlay to intuitively show crowd density.
- **Develop a Real-time Alert System:** To automatically trigger and log a 3-tier alert system based on configurable thresholds:
  1. Per-Person Alert (Dwell time in danger zone)
  2. Zone Population Alert (Number of people in danger zone)
  3. Overall Population Alert (Total people in frame)
- **Provide Secure Access and Data Visualization:** To build a full-stack web application with secure, role-based user authentication (User and Admin) and an interactive dashboard displaying real-time analytics, charts, and alert logs.
- **Enable System Configuration and Reporting:** To create a secure admin panel for managing users and dynamically configuring system-wide alert thresholds.

  To provide functionality for exporting historical alert data and individual-person summaries in PDF and CSV formats.

1.4 Report Structure

This report is organized into the following chapters:

- **Chapter 1: Introduction** – Outlines the project's background, problem statement, and objectives.
- **Chapter 2: System Design and Technology** – Details the high-level system architecture, technical workflow, and the rationale for the selected technology stack.
- **Chapter 3: Implementation Details** – Provides a low-level overview of the core software modules, the database schema, and the logic of the alert mechanism.
- **Chapter 4: Results and System Functionality** – Demonstrates the final application's features, including user authentication, the live dashboard, admin panel, and reporting, with illustrative screenshots.
- **Chapter 5: Challenges and Future Scope** – Discusses the challenges faced during development and proposes potential enhancements for future work.

- **Chapter 6: Conclusion** – Summarizes the project's achievements and its overall success in meeting the defined objectives.
- **Bibliography** – Lists the research papers and resources cited in this report.
- **Appendix** – Provides the complete setup and installation guide for reproducing the project environment.

Chapter 2

System Design and Technology

This chapter outlines the high-level architecture of the system, the step-by-step technical workflow, and the rationale behind the chosen technology stack.

2.1 System Architecture

The system is designed as a full-stack web application with a modular architecture, separating the video processing pipeline from the web server and database. The workflow, as shown in Figure 2.1, follows a clear data-processing path.

The workflow is as follows:

1. **Video Input:** The system ingests raw video frames from a file or live webcam feed.
2. **YOLOv8 Detection:** Each frame is processed by the YOLOv8 (yolov8n.pt) model, which detects all instances of the "person" class.
3. **PersonTracker (`detector.py`):** The detection bounding boxes are passed to a tracker (ByteTrack), which assigns and maintains a unique ID for each person. This module also calculates dwell time within the user-defined zone and checks for alert conditions.
4. **Flask Backend (`app.py`):** This is the central controller. It serves the video stream, handles user authentication, and provides API endpoints. It logs alerts to the database and reads settings from it.
5. **Database (PostgreSQL):** The database stores persistent data: user accounts, all generated alert logs, and system-wide settings.
6. **Frontend (Client):** The user's web browser interacts with the backend. It displays the live video stream, fetches data from the API to populate dashboard charts (Chart.js), and allows admins to configure settings.
7. **Report Generator:** A separate module (`report_generator.py`) is called by the Flask backend to generate PDF reports on demand.

Placeholder: System Architecture Diagram

A flowchart showing Video Input -> YOLOv8 -> PersonTracker -> Flask Backend, which then splits to Database, Frontend, and Report Generator.

Figure 2.1: High-Level System Architecture

2.2 Technical Workflow

The operational flow from the user's perspective is as follows:

1. A user registers or logs into the web application. The Flask backend authenticates them using Flask-JWT-Extended and assigns a role ('user' or 'admin').
2. The user navigates to the analysis page and selects a video source (webcam or file upload).
3. An OpenCV window is presented, prompting the user to interactively draw a polygonal "Danger Zone" with their mouse.
4. Once the zone is defined, the `detector.py` module begins processing the video stream.
5. The YOLOv8 model, configured with a 'bytetrack.yaml' tracker, detects and tracks individuals, assigning persistent IDs (e.g., P1, P2).
6. For each frame, the system checks the position of each person (P1, P2, etc.) against the defined zone.

7. It calculates the cumulative time spent by each person inside the "Red Zone" versus the "Green Zone."
8. The system checks alert conditions against thresholds read from the `SystemSettings` table in the database.
9. If an alert is triggered (e.g., "P1 dwell time > 10s"), it is logged to the `AlertHistory` table in PostgreSQL and visually overlaid on the video stream.
10. The frontend dashboard continuously polls the `/person_data` API endpoint using AJAX.
11. This API returns a JSON object with real-time data, which JavaScript uses to update the Chart.js graphs (Zone Population, Total Population, etc.) without a page reload.
12. The user can download a PDF report for an individual or a CSV of their entire alert history. An admin can perform these actions for any user and can also modify the alert thresholds in the admin panel.

2.3 Technology Stack Rationale

The selection of technologies was crucial to balancing real-time performance, scalability, and development speed. The rationale for each component is detailed in Table 2.1.

Table 2.1: Technology Stack Rationale

Component	**Technology**	**Purpose in This Project**
Backend Framework	Flask	The core web server handling routes, authentication (JWT), logic, and serving all data and video streams.
AI Detection	YOLOv8 (Ultralytics)	State-of-the-art real-time object detection model. The lightweight ‘yolov8n.pt‘ model was used to find people in video frames with high speed.
AI Tracking	ByteTrack / BoT-SORT	The tracking algorithm (configured via YAML) used by YOLOv8 to assign and maintain persistent, unique IDs for each detected person.
Video Processing	OpenCV-Python	Used for capturing the video feed (from file or webcam), processing individual frames, drawing overlays (zones, boxes, text), and generating the MJPEG live stream.
Database	PostgreSQL	A robust, scalable, and open-source relational database used to store all persistent data, including user credentials, alert logs, and system settings.

Table 2.1 – continued from previous page

Component	**Technology**	**Purpose in This Project**
Database ORM	Flask-SQLAlchemy	Manages the database schema and handles all database queries in an object-oriented way within Python, improving code readability and security.
Authentication	Flask-JWT-Extended	Manages user sessions using JSON Web Tokens (JWT) for secure, stateless, and role-based access control to the dashboard and admin panel.
Frontend Logic	JavaScript (AJAX/Fetch)	Used to asynchronously fetch data (e.g., from the `/person_data` API) from the backend to update the dashboard live without requiring a page reload.
Data Visualization	Chart.js	A JavaScript library used to create the responsive, live-updating charts (line, bar, scatter) on the user dashboard and admin panel.
PDF Reporting	ReportLab	A Python library used to dynamically generate and serve PDF reports summarizing an individual person's activity and alerts.
DB Connector	psycopg2-binary	The Python driver that allows Flask and SQLAlchemy to communicate with the PostgreSQL database.
Password Security	Werkzeug Security	A Flask dependency used to securely hash user passwords (using 'generate_password_hash') and verify them during login.
Configuration	python-dotenv	Loads sensitive credentials (database URI, secret keys) from a `.env` file into the application as environment variables, keeping them out of the source code.
AI Configuration	PyYAML	Used by `detector.py` to load AI model settings (like the specific tracker file) from a `config.yaml` file.
Scientific Computing	NumPy	Used in `detector.py` to perform efficient mathematical operations for calculating the heatmap overlay.

Chapter 3

Implementation Details

This chapter provides a low-level description of the project's core components, focusing on the software modules and the database schema that form the system's backbone.

3.1 Core Software Modules

The application logic is encapsulated in several key Python files and JavaScript files.

`app.py` (Backend Controller) This is the main Flask application file. Its primary responsibilities include:

- Initializing the Flask app, SQLAlchemy (database), and JWTManager (authentication).
- Defining all application routes (e.g., `/login`, `/dashboard`, `/admin`).
- Handling user registration, login, and logout logic.
- Protecting routes using `@jwt_required` and custom admin-only decorators.
- Defining API endpoints to serve data (`/person_data`) and manage the system (`/admin/update_settings`).
- Serving the live video stream as a multipart response (MJPEG).
- Handling report generation requests by calling the report generator and serving the resulting file.
- Defining the database models (User, AlertHistory, SystemSettings).

`detector.py` (AI Detection Engine) This module is the AI "brain" of the system, run as a separate process or thread managed by `app.py`.

- Initializes the YOLOv8 model with the specified tracker configuration.
- Contains the logic for capturing frames from the video source.
- Manages the interactive zone-drawing functionality using OpenCV's mouse callback functions.
- In its main processing loop, it passes frames to the YOLO model.
- Receives track data (bounding boxes and IDs) from the tracker.
- Maintains a dictionary to store `track_data` (e.g., dwell time) for each unique ID.

- Checks each person's location against the defined zone polygon.
- Increments dwell time and checks against the thresholds (fetched from the DB via `app.py`) to trigger alerts.
- Draws all visualizations on the frame (bounding boxes, zones, heatmaps, alert text) before yielding it to the web server.

`report_generator.py` (PDF Generator) A utility module dedicated to creating PDF reports.

- Contains a function that accepts person-specific data (ID, zone times, alerts).
- Uses the ReportLab library to programmatically create a PDF document.
- Styles the document and formats the data into a clean table structure.
- Returns the generated PDF, which `app.py` then serves to the user.

`script.js` (User Dashboard Logic) This file controls the interactivity of the main user dashboard.

- Uses `fetch()` and `setInterval()` to continuously poll the `/person_data` API endpoint.
- Parses the incoming JSON response.
- Updates the live alert list, the person data table, and the Chart.js graphs with the new data.

`admin.js` (Admin Panel Logic) This file manages the UI and logic for the admin panel.

- Handles the tabbed interface for switching between User Management, Settings, and Logs.
- Populates the "Alerts Per User" chart.
- Manages the form submissions for updating system settings and user roles via AJAX, preventing page reloads.

3.2 Database Schema

The system's persistent data is stored in a PostgreSQL database, managed by Flask-SQLAlchemy. The schema is normalized and consists of three main tables, as detailed in Table 3.1 and illustrated in the ER diagram (Figure 3.1).

3.2.1 User Table

This table stores all user accounts, credentials, and access levels. The `password_hash` field stores passwords encrypted using Werkzeug's security functions. The `role` field is critical for access control, differentiating 'user' from 'admin'.

Placeholder: Entity-Relationship Diagram

A diagram showing three tables: User, AlertHistory, and SystemSettings. A one-to-many relationship is shown from User to AlertHistory (one User can have many Alerts).

Figure 3.1: Database Entity-Relationship Diagram

3.2.2 AlertHistory Table

This table acts as a permanent log for every alert generated by the system. It is linked to the `User` table via the `user_id` foreign key, allowing for queries that retrieve all alerts generated under a specific user's session.

3.2.3 SystemSettings Table

This table is a key-value store for system-wide configuration, allowing for dynamic updates from the admin panel without restarting the server. It stores the thresholds that `detector.py` uses to trigger alerts. Default keys include:

- `person_threshold`: Max time (in seconds) a person can stay in the red zone. (e.g., 10)
- `zone_threshold`: Max number of people allowed in the red zone. (e.g., 5)
- `overall_threshold`: Max total people allowed in the entire frame. (e.g., 20)

Table 3.1: Database Schema Details

Table	Field	Description
User		
	`id (PK)`	Integer, Primary Key
	`username`	String(80), Unique, Not Null
	`password_hash`	String(256), Not Null
	`first_name`	String(100)
	`last_name`	String(100)
	`email`	String(120), Unique
	`profile_pic`	String(120), Default: 'default.png'
	`role`	String(50), Not Null, Default: 'user'
AlertHistory		
	`id (PK)`	Integer, Primary Key
	`user_id (FK)`	Integer, Foreign Key to User.id
	`timestamp`	DateTime, Default: UTC Now
	`alert_type`	String(50)
	`message`	String(255)
SystemSettings		
	`id (PK)`	Integer, Primary Key
	`key`	String(50), Unique, Not Null
	`value`	String(100)

Chapter 4

Results and System Functionality

This chapter showcases the final application, demonstrating its core features and functionality through screenshots of the user interface.

4.1 User Authentication

The system is protected by a secure login and registration system. Access to any analysis page requires authentication.

Placeholder: Login Page Screenshot

A clean, minimalist login form with fields for Username and Password.

Figure 4.1: User Login Page

4.2 Analysis and Live Dashboard

After logging in, the user can start a new analysis from their webcam or by uploading a video file (Figure 4.3). The main dashboard (Figure 4.4) displays the live, annotated

Placeholder: Registration Page Screenshot
A registration form with fields for Username and Password.

Figure 4.2: User Registration Page

video stream. Bounding boxes are color-coded (green for safe, red for in-zone), and alerts are overlaid directly on the video. Below the stream, a data table provides per-person details, including dwell times and alert status, with options to download individual PDF reports.

Placeholder: New Analysis Page Screenshot
Shows two buttons: "Live Webcam Analysis" and "Or Upload a Video File".

Figure 4.3: New Analysis Selection Page

Placeholder: Live Dashboard Screenshot

Shows the main dashboard: a large video feed with bounding boxes, tracker IDs (P1, P2), a red "Danger Zone," and overlaid text alerts. Below is a data table of all tracked persons.

Figure 4.4: Live Dashboard Overview

4.3 Data Visualization and Reporting

The "Summary" page (Figure 4.5) provides live-updating charts, including a bar chart for Zone Population (Red vs. Green) and a line chart for Total Population Over Time. The "History" page (Figure 4.6) displays a clean, tabular log of all historical alerts for the user, with a master "Download History (CSV)" button.

4.4 Admin Panel and System Configuration

Users with the 'admin' role have access to a secure "Admin Panel" (Figure 4.7). This panel features a tabbed interface for:

- **System Statistics:** Viewing charts like "Total Alerts Per User."

Placeholder: Live Summary Dashboard Screenshot

Shows various Chart.js graphs: a bar chart for "Live Population Chart (Bar)" and a line chart for "Population Over Time (Line)".

Figure 4.5: Live Summary Dashboard with Charts

- **User Management:** Promoting/demoting users and deleting accounts.
- **Global Settings:** Modifying the live alert thresholds for `person_threshold`, `zone_threshold`, and `overall_threshold`. Changes made here are saved to the database and instantly applied to the `detector.py` module without a server restart.
- **Global Activity Log:** Viewing the complete alert history for all users.

Placeholder: Alert History Page Screenshot
A table showing "Date & Time (UTC)", "Alert Type", and "Message" for past alerts.

Figure 4.6: Alert History Page with CSV Download

Placeholder: Admin Panel Screenshot

Shows the admin dashboard with tabs for "System Statistics," "User Management," "Global Settings," and "Global Activity Log." A bar chart for "Total Alerts Per User" is visible.

Figure 4.7: Admin Panel for System Configuration

Chapter 5

Challenges and Future Scope

5.1 Challenges Encountered

During development, two primary technical challenges were identified:

- **Object Occlusion:** In dense crowds, when individuals cross paths or are obscured by other people, the ByteTrack tracker occasionally swapped IDs or lost track of a person. While ByteTrack is robust, severe and prolonged occlusion remains a significant challenge in person tracking, leading to potential inaccuracies in dwell time calculation.
- **Inference Speed vs. Accuracy:** Achieving real-time performance (e.g., >25 FPS) is a trade-off with detection accuracy. The lightweight `yolov8n.pt` (nano) model was chosen for its high speed, but it can occasionally miss detections in complex scenes compared to larger models like `yolov8m.pt` (medium). This is a critical consideration for deployment on non-GPU hardware.

5.2 Future Enhancements

The current system provides a strong foundation. The following enhancements could significantly expand its capabilities:

- **Multi-Zone and Multi-Level Alerts:** Enhance the system to support drawing multiple, named zones (e.g., "Exit," "Restricted," "Queue"). These zones could have different alert levels, such as a 'Yellow' warning zone and a 'Red' danger zone.
- **External Alert Integration:** Integrate with third-party messaging services like Twilio (for SMS) or SendGrid (for email). This would allow the system to send automatic, critical alerts to security personnel's mobile devices, rather than relying on dashboard monitoring.
- **Cloud Deployment and Multi-Camera View:** Re-architect the application for cloud deployment (e.g., on AWS or Azure) to ensure scalability. This would involve creating a centralized dashboard that can ingest and manage multiple camera streams simultaneously from different physical locations.
- **Edge AI Device Compatibility:** Optimize the AI model and processing pipeline to run efficiently on low-power Edge AI devices (e.g., NVIDIA Jetson, Raspberry

Pi). This would allow for decentralized, on-site processing, reducing latency and network bandwidth requirements.

- **Advanced Analytics:** Implement more advanced analytics, such as anomaly detection (e.g., detecting falls or erratic movement) and predictive modeling to forecast overcrowding before it happens.

Chapter 6

Conclusion

This project successfully demonstrates the design, development, and deployment of a complete AI-Based Crowd Monitoring and Safety Management System. By integrating a high-performance YOLOv8 model with a robust Flask backend, PostgreSQL database, and a dynamic Chart.js dashboard, the project delivers a unified and automated solution for crowd analysis.

The system effectively replaces inefficient manual surveillance by providing:

- Real-time, reliable person detection and tracking with intelligent Danger Zone monitoring.
- A flexible 3-tier alert mechanism for per-person, zone, and overall population breaches.
- A secure, role-based platform featuring an advanced admin panel for live, dynamic system configuration.
- Actionable insights through a live analytics dashboard and exportable PDF/CSV reports for historical analysis.

The resulting application provides a strong and scalable foundation for real-world deployments in diverse fields, including public safety, retail analytics, event management, and operational monitoring. It showcases how modern AI and full-stack web technologies can be combined to build smart, scalable, and responsive safety systems.

Bibliography

[1] Dense-stream YOLOv8n: A lightweight framework for real-time crowd counting and safety (https://www.nature.com/articles/s41598-024-58937-z)

[2] YOLOv8 Object Tracking and Counting - LearnOpenCV (https://learnopencv.com/yolov8-object-tracking-and-counting/)

[3] Optimize Safety with Smart Crowd Management Solutions - Viso.ai (https://viso.ai/crowd/)

[4] Vision AI for crowd management - Ultralytics (https://ultralytics.com/solutions/crowd-management/)

[5] Crowd Gathering Detection using YOLOv8 - GitHub (https://github.com/ProRiko/Crowd-Gathering-Detection)

[6] Advanced Crowd and Event Management through AI - KE Leaders (https://keleaders.com/advanced-crowd-event-management/)

[7] AI Crowd Management for Real-Time Monitoring (https://iprogrammer.au/post/2756/ai-crowd-management-for-real-time-monitoring)

[8] How Spatial AI Is Transforming Crowd Management - Outsight (https://insights.outsight.ai/how-spatial-ai-is-transforming-crowd-management-in-public-spaces/)

Appendices

Appendix A

Setup and Installation Guide

This appendix provides the necessary steps to set up and run the project in a local development environment.

A.1 Prerequisites

- Python 3.8 or higher
- A PostgreSQL database server
- Git

A.2 Installation Steps

1. **Clone the repository:**

```
git clone https://github.com/your-username/crowd-monitoring-ai.git
cd crowd-monitoring-ai
```

2. **Create and activate a virtual environment:**

```
# Windows
python -m venv myenv
myenv\Scripts\activate

# macOS/Linux
python3 -m venv myenv
source myenv/bin/activate
```

3. **Install dependencies:** Install all required packages from the `requirement.txt` file.

```
pip install -r requirement.txt
```

4. **Configure PostgreSQL:** Create a new database in your PostgreSQL server. For example, named `crowd_monitoring`.

5. **Create the `.env` file:** In the root directory of the project, create a file named `.env`. This file stores sensitive credentials and configuration.

```
DB_USER=your_postgres_username
DB_PASSWORD=your_postgres_password
DB_HOST=localhost
DB_PORT=5432
DB_NAME=crowd_monitoring
SECRET_KEY=your_own_random_secret_key
JWT_SECRET_KEY=your_own_random_jwt_secret_key
```

A.3 Running the Application

1. **Initialize the database:** The first time you run the application, it will create all the necessary tables based on the models defined in `app.py`.
2. **Run the Flask server:**

   ```
   python app.py
   ```

3. **Access the application:** Open a web browser and navigate to http://127.0.0.1:5000.

Note: The first user to register will be automatically assigned the 'admin' role. All subsequent users will be 'user' by default.

A.4 Project Dependencies (`requirement.txt`)

```
Flask
flask_sqlalchemy
flask_jwt_extended
psycopg2-binary
python-dotenv
opencv-python
ultralytics
PyYAML
numpy
reportlab
```