**Infosys Springboard Virtual Internship**

PROJECT:- Crowd Count People Counting Using Video Analysis

(Integrated with all the milestone)

Name:-Shital Katre

Date:- 29-12-2025

* **Project Overview**

People Counting using Video Analysis is a computer vision–based project developed to detect, track, and count people using video feeds. The system helps in monitoring crowd density in real time by analyzing live or recorded video streams. It provides zone-wise crowd counting, live analytics, and visualization through dashboards, which can help in effective crowd management and safety monitoring.

* **Objectives of the Project**

To capture live video feed from camera

To define and manage zones on video frames

To detect and track people in real time

To perform zone-wise people counting

To visualize crowd data using dashboards

To generate analytics and downloadable reports

* **System Overview**

The system takes video input from a webcam or IP camera. Zones are defined on the video feed for monitoring specific areas. Using object detection and tracking algorithms, people are detected and assigned unique IDs. The number of people in each zone is counted and analyzed. The results are displayed on a live dashboard, and an admin panel allows secure access, analytics viewing, and report export.

* **Technologies Used**

Python

OpenCV

YOLOv8 (Object Detection)

DeepSORT / BYTETrack (Tracking)

Streamlit / Flask (Dashboard)

Pandas

Matplotlib

**Milestone 1**– Video Feed & Zone Management

* **Introduction**

This project aims to build a real-time People Counting System using video analysis. The system will detect and monitor how many people enter, exit, or stay inside a specific region.

Milestone-1 focuses on Video Feed & Zone Management, which is the foundation of the entire project.

The major capabilities implemented in this milestone include:

* Opening and displaying live video feed using OpenCV.
* Allowing the user to draw customizable rectangular zones.
* Moving, resizing, deleting, and clearing zones..
* Saving and loading zones through JSON files.
* Preparing the environment for future milestones like object detection and crowd counting

Milestone-1 ensures that the graphical interface and zone-handling logic are perfectly functional before integrating AI models.

* **Workflow & Architecture**

1. Start the application – The user runs the Python script, and the webcam video feed begins.

2. Initialize the video feed – The system captures live video frames and displays them in a window.

3. Activate Zone Management Mode –

Press M → Enables mouse-based zone drawing mode.

The user can now draw rectangular zones on the video feed.

4. Draw and edit zones –

Left-click → Start drawing a rectangle

Drag → Set the rectangle size

Release → Finalize the zone

Multiple zones can be created.

5. Real-time zone visualization –

All drawn rectangles appear immediately on the screen.

The user can visually confirm zone placement.

6. Save zones to file –

Press S → Saves all created zones into a JSON file.

file stores zone coordinates (x, y, width, height).

7. Load saved zones –

Press L → Loads previously saved zones from the JSON file.

The rectangles reappear at the same positions on the video feed.

Useful after restarting the program.

8. Verify loaded zones –

The user checks if the loaded rectangles are correctly displayed.

If needed, new zones can be drawn again.

9. Quit the application –

Press Q → Closes the webcam window and exits safely.

* **Code Implementation-** Running file code

**main.py:**

import cv2

from video\_feed import VideoFeed

from draw\_zones import ZoneDrawer

drawer = ZoneDrawer()

feed = VideoFeed(0)

cv2.namedWindow("Milestone 1")

cv2.setMouseCallback("Milestone 1", drawer.mouse\_callback)print("\nControls:")

print("Draw: Drag mouse")

print("Move: Drag inside")

print("Resize: Drag corners")

print("S = Save zones")

print("L = Load zones")

print("D = Delete selected")

print("C = Clear all zones")

print("Q = Quit\n")

while True:

    frame = feed.get\_frame()

    if frame is None:

        break

view = drawer.draw(frame)

    cv2.imshow("Milestone 1", view)

key = cv2.waitKey(1) & 0xFF

if key == ord('s'): drawer.save()

    elif key == ord('l'): drawer.load()

    elif key == ord('d'):

        if drawer.selected is not None:

            drawer.zones.pop(drawer.selected)

            drawer.selected = None

            print(" Deleted")

    elif key == ord('c'):

        drawer.zones = []

        drawer.selected = None

        print(" Cleared")

    elif key == ord('q'):

        break

feed.release()

* **Code Explanation**

main.py –

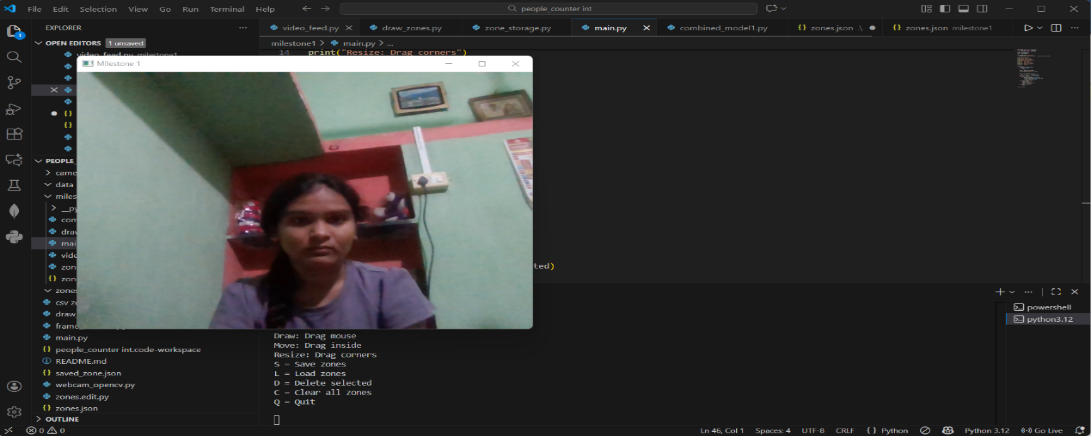
This is the main running script.

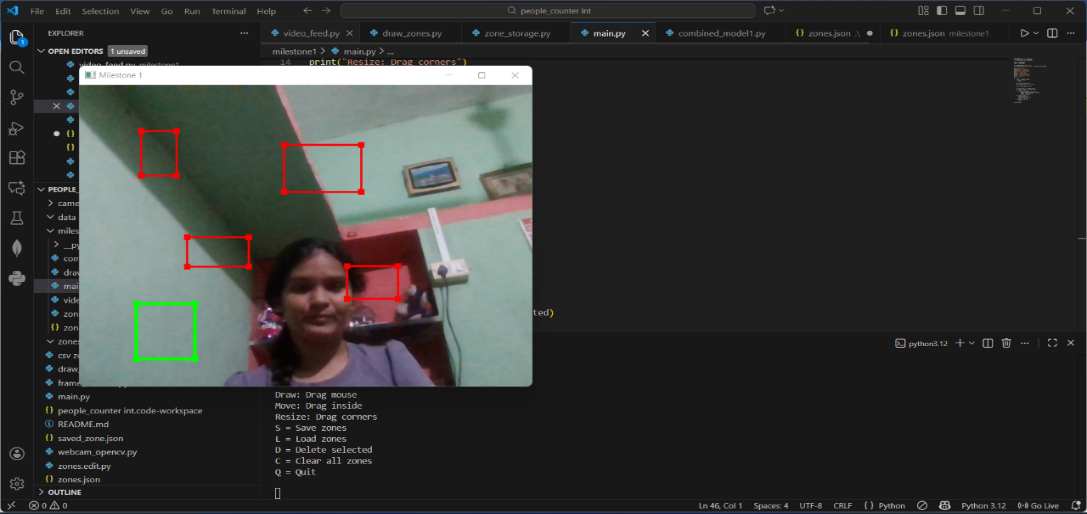
It handles:

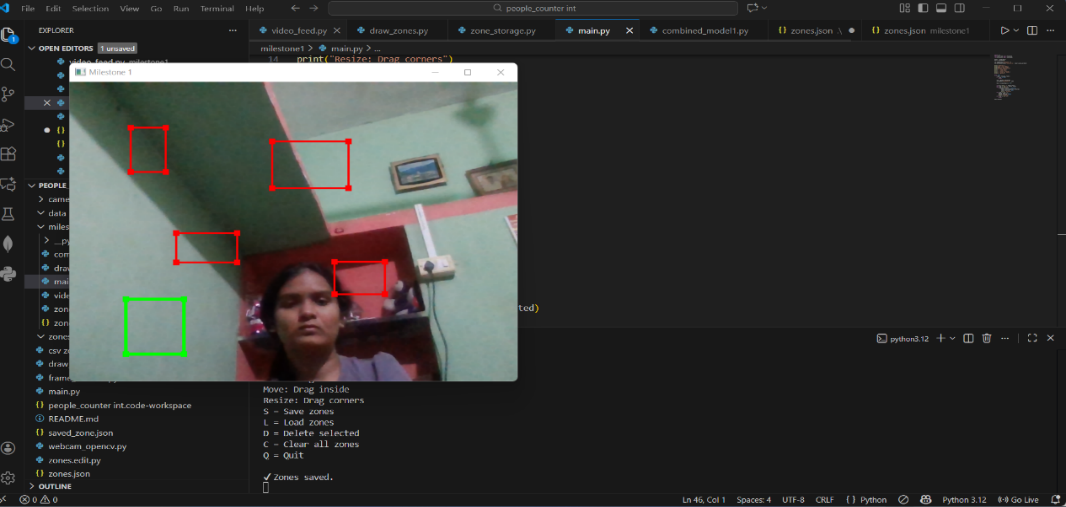
* Window creation
* Connecting mouse handler
* Fetching video frames
* Drawing updated frames
* Detecting keyboard shortcuts:

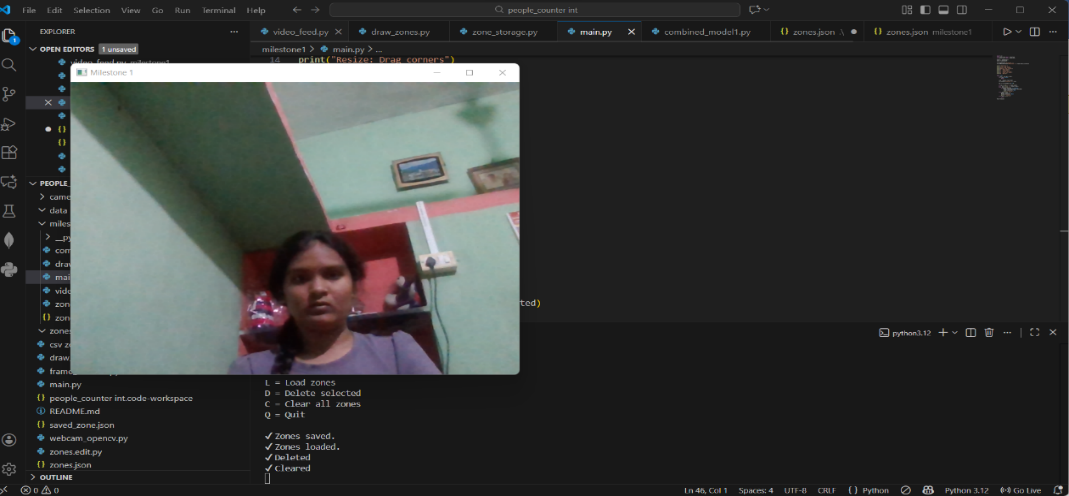
Key Function

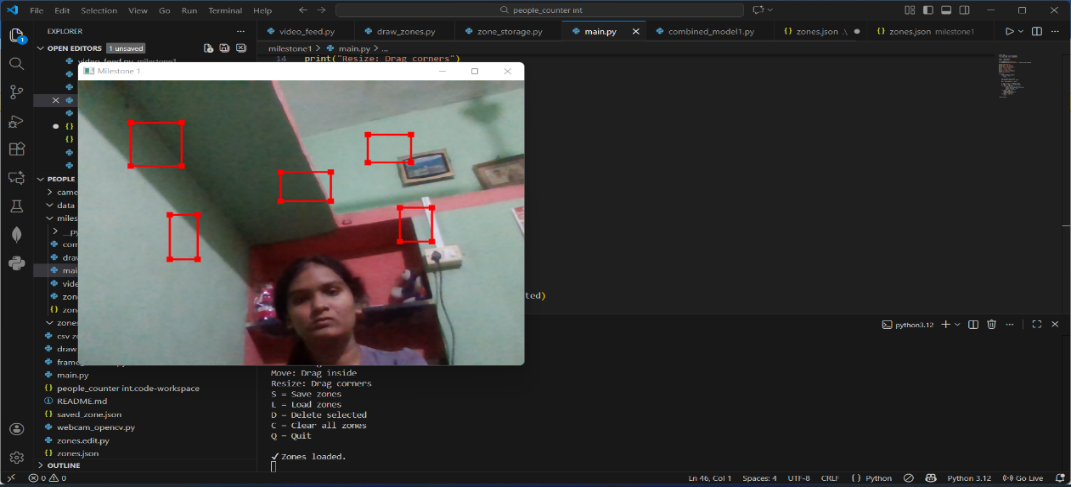
* S Save zones
* L Load zones
* D Delete selected zone
* C Clear all zones
* Q Quit
* **Output Screenshots**

****

****



****

****

**Milestone 2** – People Detection and Counting

* **Introduction**

This project implements a real-time People Detection, Tracking, and Zone-based Counting System using a webcam.

Milestone 1: Video feed + mouse-based zone creation and management

Milestone 2: People detection, DeepSORT tracking, and zone-based counting

The final system allows drawing zones using the mouse, saving/loading zones, and counting how many people enter each zone in live video feed.

* **System Workflow & Architecture**

1. Run the program → Webcam opens.

2 Draw zones using mouse (click + drag → rectangle).

3. Press ENTER → Zone saved and displayed on screen.

4. YOLO detects persons in each frame.

5. DeepSORT assigns unique tracking IDs.

6. If person enters a zone → Count for that zone increases.

7. On-screen display shows:

Bounding boxes

Person IDs

Zone rectangles

Zone names

Zone-wise live count

8. Keyboard controls for zone management:

s → Save zones

l → Load zones

d→ Delete last zone

c→ Clear all zones

q→ Quit

* **Code Implementation**

import cv2

import numpy as np

from ultralytics import YOLO

from pathlib import Path

import os

ZONE\_FILE = "zones.npy"

# Safe load

zones = []

if Path(ZONE\_FILE).exists():

    try:

zones = np.load(ZONE\_FILE, allow\_pickle=True).tolist()

    except:

        zones = []

drawing = False

ix, iy = -1, -1

current\_zone = None

# ------------------- MOUSE CALLBACK -------------------

def mouse\_draw(event, x, y, flags, param):

    global ix, iy, drawing, current\_zone, frame

    if event == cv2.EVENT\_LBUTTONDOWN:

        drawing = True

        ix, iy = x, y

        current\_zone = [(ix, iy), (ix, iy)]

    elif event == cv2.EVENT\_MOUSEMOVE and drawing:

        current\_zone[1] = (x, y)

    elif event == cv2.EVENT\_LBUTTONUP:

        drawing = False

        current\_zone[1] = (x, y)

# ------------------- DRAW ZONES -------------------

def draw\_existing\_zones(frame):

    for i, zone in enumerate(zones):

        cv2.rectangle(frame, zone[0], zone[1], (0, 255, 255), 3)

        cv2.putText(frame, f"Zone {i+1}", (zone[0][0], zone[0][1]-10),

                    cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 255, 255), 2)

    if current\_zone:

    return x1 <= point[0] <= x2 and y1 <= point[1] <= y2

model = YOLO("yolov8n.pt")

cap = cv2.VideoCapture(0)

cv2.namedWindow("video")

cv2.setMouseCallback("video", mouse\_draw)

while True:

    ret, frame = cap.read()

    if not ret:

        break

    results = model(frame, verbose=False)

    detection\_data = results[0].boxes.data.cpu().numpy()

    draw\_existing\_zones(frame)

zone\_counts = [0 for \_ in range(len(zones))]

            continue

        x1, y1, x2, y2 = map(int, [x1, y1, x2, y2])

        cx, cy = int((x1+x2)/2), int((y1+y2)/2)

cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)

        cv2.putText(frame, f"ID {idx+1}", (x1, y1-10),

                    cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0,255,0), 2)

        cv2.circle(frame, (cx, cy), 5, (255, 255, 255), -1)

for i, zone in enumerate(zones):

            if point\_in\_zone((cx, cy), zone):

                zone\_counts[i] += 1

# show counts

    y\_offset = 40

    for i, count in enumerate(zone\_counts):

        cv2.putText(frame, f"Zone {i+1} Count: {count}",

                    (10, y\_offset), cv2.FONT\_HERSHEY\_SIMPLEX, 1.5, (0,0,255), 3)

        y\_offset += 60

cv2.imshow("video", frame)

    key = cv2.waitKey(1)

  if key == ord('q'): # quit

        break

    elif key == ord('n') and current\_zone: # save current rectangle

        zones.append(current\_zone)

        np.save(ZONE\_FILE, zones)

        current\_zone = None

    elif key == ord('d') and zones: # delete last zone

        zones.pop()

        np.save(ZONE\_FILE, zones)

    elif key == ord('c'): # clear all zones

        zones = []

        current\_zone = None

        if os.path.exists(ZONE\_FILE):

            os.remove(ZONE\_FILE)

    elif key == ord('l'): # load zones

        if Path(ZONE\_FILE).exists():

            zones = np.load(ZONE\_FILE, allow\_pickle=True).tolist()

* **Code Explanation**

1. Importing required libraries

The program begins by importing important Python libraries:

OpenCV for video capture, drawing, and mouse interaction.

NumPy for saving and loading zone data.

YOLO model from Ultralytics for human detection.

Path and OS modules for checking and managing files.

These libraries together provide detection, drawing, storage, and file-handling abilities.

2. Loading previously saved zones.

The program defines a file name where zones will be stored.

It checks if the zone file exists.

If the file is found, it tries to load all saved zones.

If loading fails or there is no file, the zones list starts empty.

3. Defining variables for drawing.

Some variables are created for handling mouse actions:

A flag to check if drawing has started.

The starting coordinates of the mouse.

A temporary zone that stores the rectangle currently being drawn.

These help the program understand where the user starts and stops drawing a rectangle.

4. Mouse callback function

A mouse callback function detects and handles three main events:

* When the left mouse button is pressed, drawing begins and the starting point is recorded.
* While moving the mouse with the button held, the rectangle keeps updating.
* When the mouse button is released, the rectangle is completed.

This allows the user to visually draw rectangular zones on the screen.

5. Function to draw zones on the screen

A separate function displays all previously saved zones on each frame.

Every zone shows a rectangle and a label such as “Zone 1”, “Zone 2”, etc.

The current ongoing rectangle also appears on the screen.

This helps users see all active zones clearly.

6. Function to check if a point is inside a zone

The program checks whether the center of a detected person is inside a zone.

It compares the point coordinates with the zone boundaries.

If the point lies inside, it returns true.

This is needed for counting people inside each zone.

7. Loading the YOLO model

A lightweight YOLO model is loaded to detect people in real time.

This model identifies humans and gives bounding boxes for each detected person.

8. Starting the webcam video loop

The webcam starts capturing video frames.

A window is created to show the live video.

The mouse callback is connected to this window so the user can draw rectangles.

9. Running detection on each frame

Every frame is passed to the YOLO model.

The detection results are converted into usable values.

Only detections where the class is “person” are processed.

10. Drawing zones on the frame

All existing zones and the current rectangle are drawn on the live frame before showing it.

11. Counting people in each zone

A list is created to maintain the count for each zone.

For every detected person:

* The bounding box is drawn
* A unique ID is displayed
* The center point of the person is calculated
* The point is checked for all zones
* If the point lies inside a zone, that zone’s count increases
* This allows real-time people counting in each defined area.

12. Displaying zone-wise counts

On the top-left of the screen, counts for all zones are displayed in large red text:

* “Zone 1 Count: X”
* “Zone 2 Count: Y”

and so on.

This gives the live tracking output clearly.

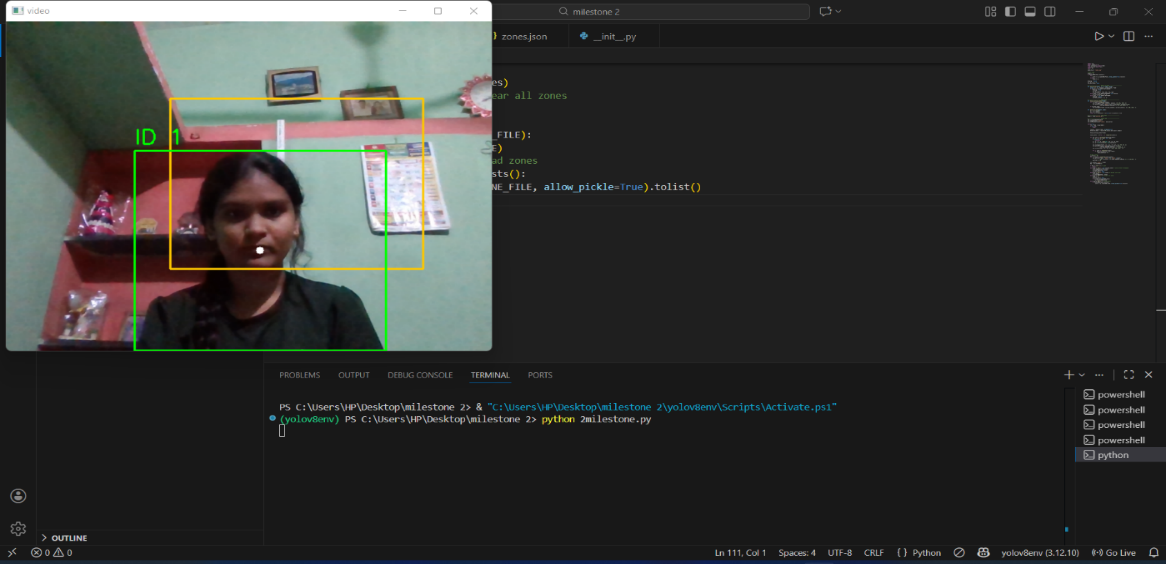
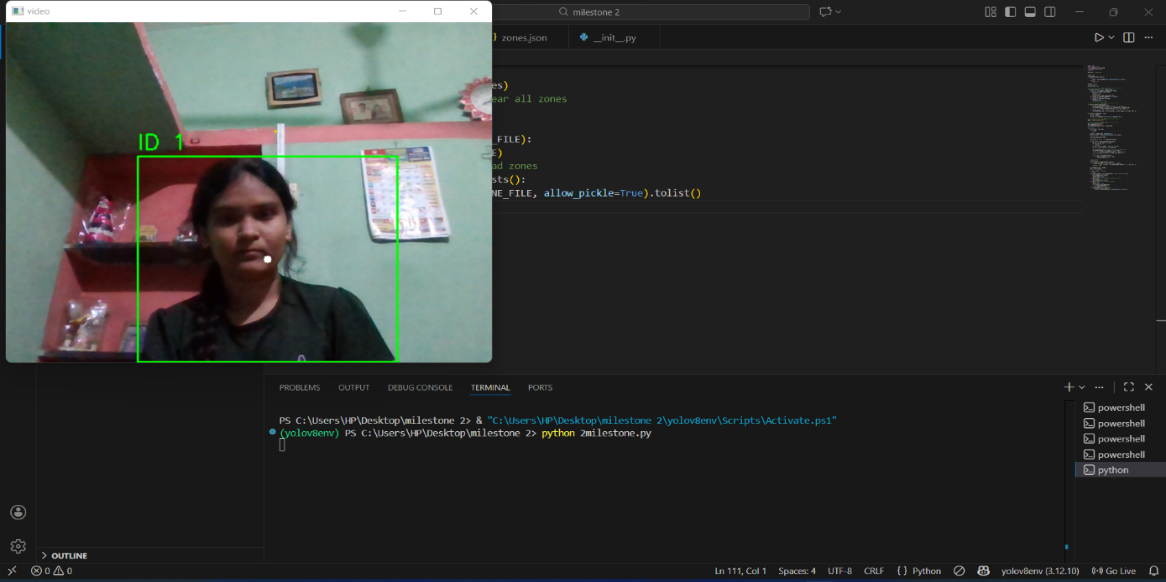
13. Handling keyboard controls

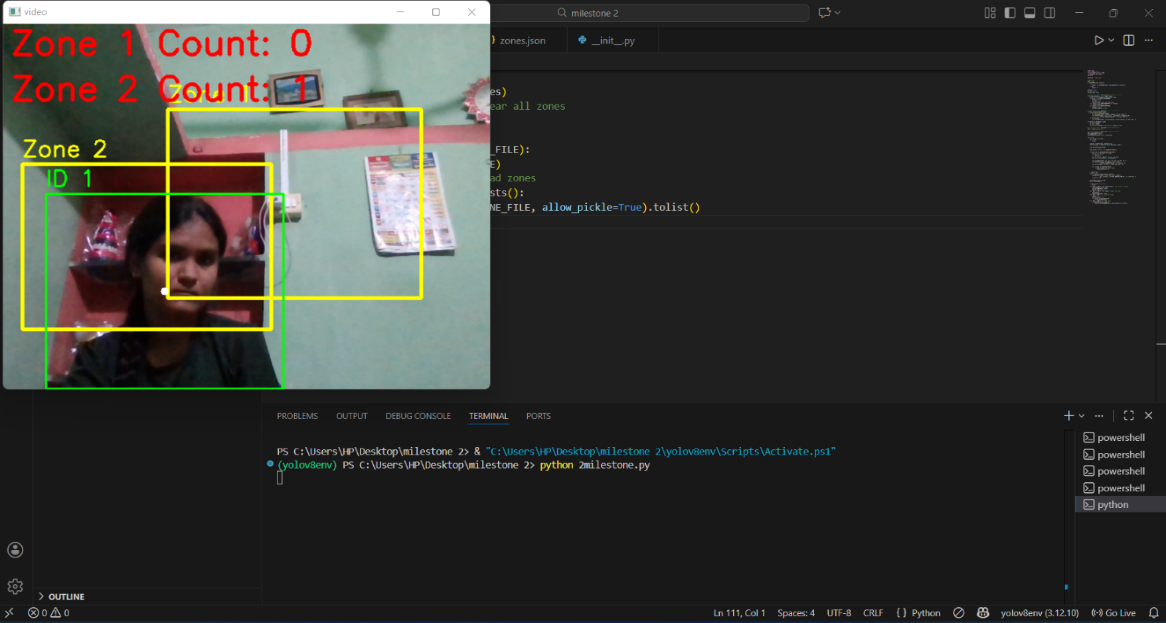
The program listens for specific keys:

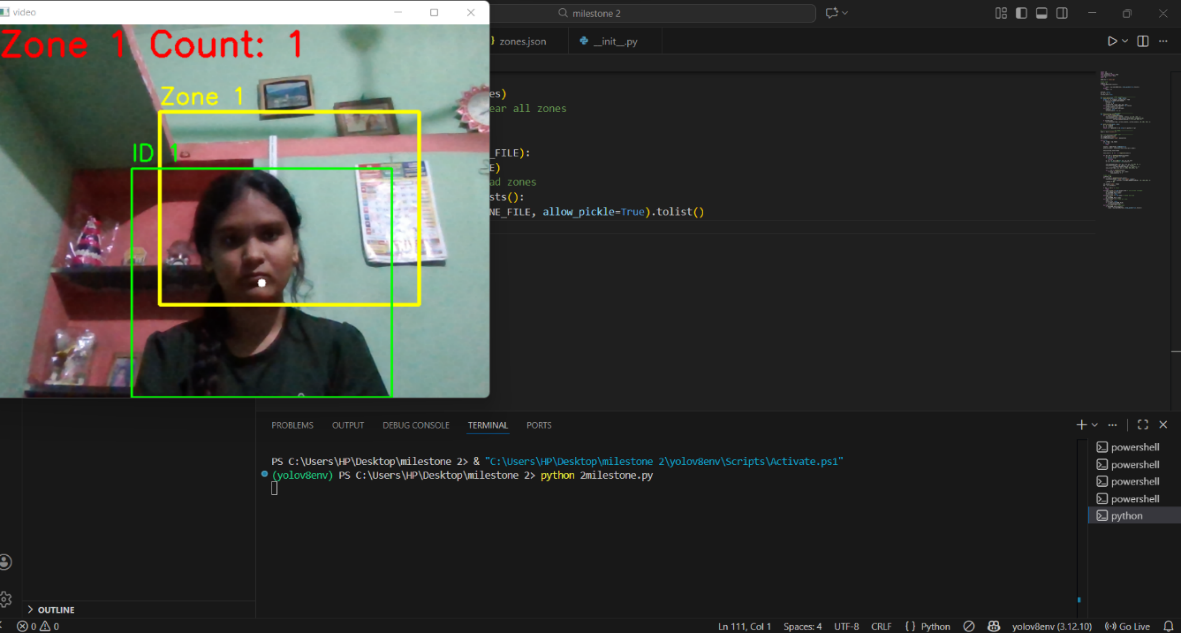
* ‘q’ → Quit the program
* ‘s’ → Save the current rectangle as a zone
* ‘d’ → Delete the most recently added zone
* ‘c’ → Clear all zones and remove the saved file
* ‘l’ → Load the saved zones file

These controls make zone management very easy without touching the code.

* **Output Screenshots**







**Milestone 3:-** Live Dashboard & Crowd Analytics

* **Introduction**

Milestone 3 focuses on transforming the processed crowd data into meaningful visual insights using a live dashboard. After detecting and counting people in different zones, it becomes essential to present this information in a clear and understandable manner. This milestone introduces real-time analytics and visualization techniques to monitor crowd distribution, density, and trends. The dashboard helps users and administrators quickly analyze crowd conditions and make informed decisions based on graphical data representation.

* **System Workflow & Architecture**

1. Data Reception

Zone-wise people count data is received continuously from the detection and counting module (Milestone 2).

1. Data Processing

The incoming data is formatted and organized for visualization.

Time-based records are maintained for trend analysis.

Analytics Generation

Zone-wise analytics are generated using bar charts to compare crowd count across zones.

Line charts are created to visualize changes in crowd count over time.

Heatmaps are generated to represent crowd density visually.

1. Dashboard Visualization

All analytics are displayed on a live dashboard interface.

The dashboard updates automatically as new data is received.

1. Alert Mechanism

Predefined threshold values are checked for each zone.

Alerts are triggered if the crowd count exceeds the threshold.

1. Output Display

Real-time graphs, heatmaps, and alerts are shown on the dashboard for monitoring.

* **Code Implementation-** Running file code

**main.py:**

import cv2

import json

import threading

import numpy as np

import pandas as pd

import time

from flask import Flask, render\_template, jsonify, send\_file

from ultralytics import YOLO

from io import BytesIO

from matplotlib.backends.backend\_pdf import PdfPages

import matplotlib.pyplot as plt

VIDEO\_FILE = "videos/Entrance area.mp4"

ZONE\_FILE = "zones.json"

THRESHOLD = 4

app = Flask(name)

model = YOLO("[yolov8n.pt](http://yolov8n.pt/)", verbose=False)

zone\_data = {}

zone\_names = ["Entrance", "Exit", "Common"]

counts = {"Entrance": 0, "Exit": 0, "Common": 0}

heatmap = {"Entrance": None, "Exit": None, "Common": None}

def draw\_zone(event, x, y, flags, param):

global ix, iy, drawing, zone\_data, current\_zone\_name

if event == cv2.EVENT\_LBUTTONDOWN:

drawing = True

ix, iy = x, y

elif event == cv2.EVENT\_LBUTTONUP:

drawing = False

zone\_data[current\_zone\_name] = [ix, iy, x, y]

def save\_zones():

with open(ZONE\_FILE, "w") as f:

json.dump(zone\_data, f)

def load\_zones():

global zone\_data

try:

with open(ZONE\_FILE) as f:

zone\_data = json.load(f)

except:

zone\_data = {}

def video\_loop():

global counts, heatmap

load\_zones()

cap = cv2.VideoCapture(VIDEO\_FILE)

for z in zone\_names:

  frame = cv2.resize(frame, (640, 480))

    results = model.track(frame, persist=True, classes=[0])

    for z in zone\_names:

        counts[z] = 0

    if results and results[0].[boxes.id](http://boxes.id/) is not None:

      for box, tid in zip(results[0].boxes.xyxy, results[0].[boxes.id](http://boxes.id/)):

            x1, y1, x2, y2 = map(int, box)

            cx, cy = (x1+x2)//2, (y1+y2)//2

            cv2.rectangle(frame, (x1,y1), (x2,y2), (0,255,0), 2)

            cv2.circle(frame, (cx,cy), 4, (0,255,0), -1)

      cv2.putText(frame, f"ID {int(tid)}", (x1,y1-5),

                        cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255,255,255), 1)

            for z in zone\_names:

                if z in zone\_data:

                    zx1, zy1, zx2, zy2 = zone\_data[z]

                    if zx1 < cx < zx2 and zy1 < cy < zy2:

                        counts[z] += 1

                        heatmap[z][cy, cx] += 2

    for z in zone\_names:

      if z in zone\_data:

            zx1, zy1, zx2, zy2 = zone\_data[z]

            cv2.rectangle(frame, (zx1,zy1), (zx2,zy2), (0,0,255), 2)

            cv2.putText(frame, z, (zx1,zy1-5),

                      cv2.FONT\_HERSHEY\_SIMPLEX, 0.6, (0,0,255), 2)

    #  HEATMAP OVERLAY FIX

    combined = np.zeros((480,640), dtype=np.float32)

  for z in zone\_names:

        combined += heatmap[z]

    combined = cv2.GaussianBlur(combined, (31,31), 0)

    norm = cv2.normalize(combined, None, 0, 255, cv2.NORM\_MINMAX)

    color = cv2.applyColorMap(norm.astype(np.uint8), cv2.COLORMAP\_JET)

    frame = cv2.addWeighted(frame, 0.6, color, 0.4, 0)

    # VIDEO ALERT TEXT

    for i, z in enumerate(zone\_names):

        if counts[z] > THRESHOLD:

            cv2.putText(frame, f"{z} area is crowded",

                        (10, 30+30\*i), cv2.FONT\_HERSHEY\_SIMPLEX,

                        0.9, (0,0,255), 2)

    cv2.imshow("Crowd Analytics", frame)

    if cv2.waitKey(30) & 0xFF == 27:

        break

cap.release()

cv2.destroyAllWindows()

@app.route("/")

def dashboard():

return render\_template("dashboard.html")

@app.route("/data")

def data():

alerts = []

for z in zone\_names:

if counts[z] > THRESHOLD:

alerts.append(f"{z} area is crowded")

history.append({

    "time\_sec": int(time.time() - start\_time),

    "Entrance": counts["Entrance"],

    "Exit": counts["Exit"],

    "Common": counts["Common"],

    "crowded": "YES" if alerts else "NO"  })

if name == "main":

load\_zones()

if not zone\_data:

    for z in zone\_names:

        current\_zone\_name = z

        cap = cv2.VideoCapture(VIDEO\_FILE)

        ret, frame = cap.read()

        frame = cv2.resize(frame, (640,480))

        cv2.namedWindow(f"Draw Zone - {z}")

        cv2.setMouseCallback(f"Draw Zone - {z}", draw\_zone)

        while True:

            temp = frame.copy()

            if z in zone\_data:

                zx1,zy1,zx2,zy2 = zone\_data[z]

                cv2.rectangle(temp,(zx1,zy1),(zx2,zy2),(0,0,255),2)

            cv2.imshow(f"Draw Zone - {z}", temp)

            if cv2.waitKey(1) & 0xFF == ord("s"):

                save\_zones()

                break

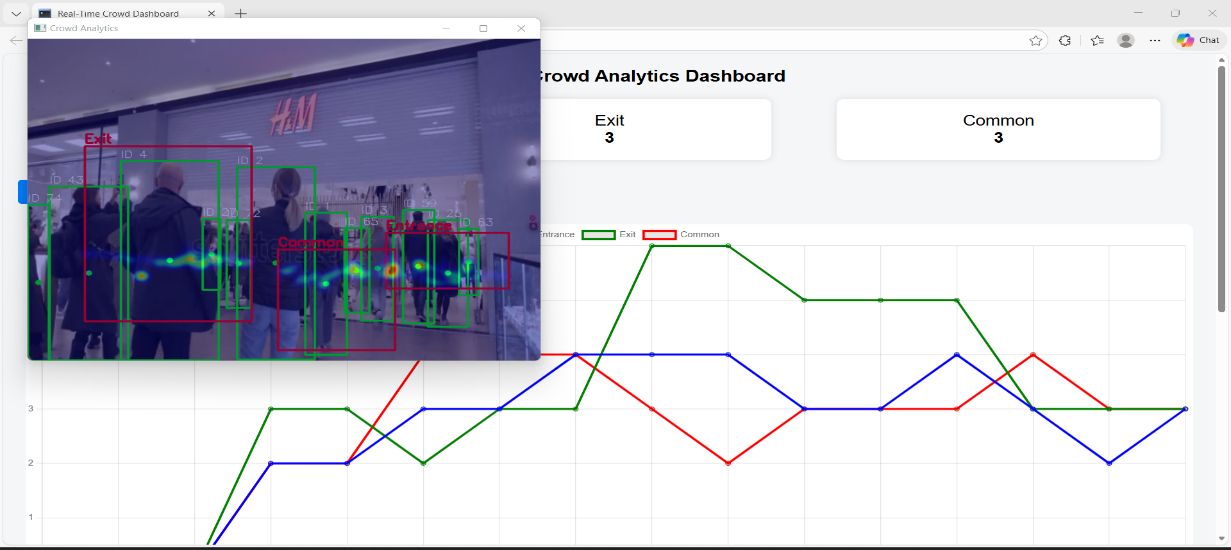
        cap.release()

        cv2.destroyAllWindows()

threading.Thread(target=video\_loop, daemon=True).start()

app.run(debug=False)

* **Code Explanation**
* This code implements the live dashboard and crowd analytics module of the CrowdCount system.
* The system reads video frames and uses the YOLOv8 model to detect and track people in real time.
* Predefined zones (Entrance, Exit, Common) are loaded from a JSON file and used for zone-wise crowd counting.
* For each detected person, the center point of the bounding box is checked to determine which zone the person belongs to, and the corresponding count is updated.
* A heatmap is generated by accumulating movement points of people and overlaying it on the video frame to visualize crowd density.
* If the number of people in any zone exceeds the defined threshold, a crowd alert message is displayed on the video.
* A Flask-based dashboard continuously receives live crowd data and displays zone-wise counts and alerts.
* Crowd history is stored over time and can be downloaded as CSV reports.
* A PDF report is generated showing a summary of crowd occupancy using graphical visualization.
* **Output Screenshots**





**Milestone 4**:- Admin Panel, Reporting & System Management

* **Introduction**

Milestone 4 focuses on providing administrative control and reporting features for the CrowdCount system. After implementing real-time crowd detection and live analytics, it is important to manage data, view historical analytics, and generate reports. This milestone introduces an admin-controlled dashboard that allows monitoring of crowd statistics, alert management, and exporting data in CSV and PDF formats. It helps in maintaining records and supports decision-making through stored analytics.

* **System Workflow & Architecture**

1. Admin Access

The admin opens the dashboard interface using a web browser.

Only authorized users can access analytics and report features.

1. Live Data Retrieval

The system receives real-time crowd count data from the analytics module (Milestone 3).

Zone-wise crowd information is continuously updated.

1. Alert Monitoring

Crowd counts are compared with predefined threshold values.

Alerts are generated and displayed when any zone becomes overcrowded.

1. Data Storage

Crowd data is stored periodically to maintain historical records.

Stored data includes time-based zone counts and crowd status.

1. Report Generation

Admin can download historical crowd data in CSV format.

Summary reports are generated in PDF format using graphical visualization.

1. Analytics Review

Admin reviews crowd trends and occupancy summaries.

Exported reports can be used for documentation and future analysis.

1. System Control

The admin panel helps in monitoring overall system performance.

Threshold values and monitoring parameters can be adjusted if required.

* **Code Implementation-** Running file code

**App.py:**

import cv2, csv

import os

from datetime import datetime

from flask import Flask, render\_template, request, redirect, session, Response, jsonify, send\_file, url\_for

from vision.detector import PersonDetector

from vision.tracker import SimpleTracker

from vision.heatmap import HeatMap

from models.camera\_model import Camera

from models.zone\_model import zones, save\_zones

from models.count\_model import counts

from models.threshold\_model import thresholds

from models.log\_model import logs , add\_alert

app = Flask(name)

app.secret\_key = "milestone4\_admin"

cameras = []

active\_camera =None

cap=None

ALERTS & THRESHOLDS

alerts = [] # Stores alert events

thresholds\_dict = {} # Admin-set limits per zone

detector = PersonDetector()

tracker = SimpleTracker()

heatmap = None

def generate\_frames():

global heatmap, cap

if cap is None:

        return

while True:

    ret,frame=cap.read()

    if not ret:

    break

  if heatmap is None:

        heatmap = HeatMap(frame.shape)

    boxes = detector.detect(frame)

    objects = tracker.update(boxes)

    counts.clear()

    for obj\_id, (cx, cy) in objects.items():

        counts.append(type('Obj', (), {

            'id': obj\_id,

            'zone': 'N/A',

            'time': datetime.now()

        }))

    for (x1, y1, x2, y2) in boxes:

        cv2.rectangle(frame, (x1,y1), (x2,y2), (255,0,0), 2)

for obj\_id, (cx, cy) in objects.items():

        cv2.circle(frame, (cx,cy), 4, (0,255,0), -1)

        cv2.putText(frame, f"ID {obj\_id}", (cx+5, cy-5),

                    cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0,255,0), 2)

    heatmap.update(objects)

    frame = heatmap.draw(frame)

    ret, buffer = cv2.imencode(".jpg", frame)

    frame = buffer.tobytes()

    yield (b"--frame\r\n"

           b"Content-Type: image/jpeg\r\n\r\n" + frame + b"\r\n")

@app.route("/video\_feed")

def video\_feed():

if "admin" not in session:

return redirect("/")

return Response(generate\_frames(),

mimetype="multipart/x-mixed-replace; boundary=frame")

@app.route("/", methods=["GET","POST"])

def login():

error = None

if request.method == "POST":

if request.form["username"] == "admin" and request.form["password"] == "admin":

session["admin"] = True

return redirect("/dashboard")

error = "Invalid credentials"

return render\_template("login.html", error=error)

@app.route("/dashboard")

def dashboard():

if "admin" not in session:

return redirect("/")

return render\_template(

"dashboard.html",

cameras=cameras,

zones=zones,

thresholds=thresholds,

counts=counts,

logs=logs)

@app.route("/camera", methods=["GET","POST"])

def camera\_page():

global cameras, active\_camera , cap

if request.method == "POST":

    id= request.form.get("id")

    name = request.form.get("name")

    source = request.form.get("source")

          if not os.path.exists(source):

        return f"video file not found:{source}",400

  cam = Camera(name, source)

    cameras.append(cam)

  active\_camera=cam #  KEY LINE

    if cap is not None:

        cap.release()

cap=cv2.VideoCapture(cam.source)

    return redirect(url\_for("dashboard"))

return render\_template("camera.html", cameras=cameras)

ZONE

@app.route("/zone", methods=["GET", "POST"])

def zone():

if 'admin' not in session:

return redirect('/')

if request.method == "POST":

    name = request.form.get("name")

    x1 = request.form.get("x1")

    y1 = request.form.get("y1")

    x2 = request.form.get("x2")

    y2 = request.form.get("y2")

  # validation

    if not name or not x1 or not y1 or not x2 or not y2:

        return render\_template("zone.html", zones=zones, error="All fields required")

zone\_data = {

        "name": name,

        "coords": [(int(x1), int(y1)), (int(x2), int(y2))]      }

  zones.append(zone\_data)

    save\_zones(zones)

return redirect("/zone")

return render\_template("zone.html", zones=zones)

@app.route("/delete\_zone/int:index")

def delete\_zone(index):

if 'admin' not in session:

return redirect('/')if 0 <= index < len(zones):

    zones.pop(index)

    save\_zones(zones)

return redirect("/zone")

@app.route('/threshold')

def threshold\_page():

if 'admin' not in session: return redirect('/')

return render\_template('threshold.html', thresholds=thresholds\_dict, alerts=logs)

@app.route('/threshold', methods=['GET', 'POST'])

def threshold():

if 'admin' not in session:

return redirect('/')

if request.method == 'POST':

    zone\_name = request.form.get('zone')

    value = request.form.get('value')

    if not zone\_name or not value:

        return render\_template(

            'threshold.html',

            thresholds=thresholds,

            zones=zones,

            error="Zone and value required"       )

  thresholds.append({

        "zone": zone\_name,

        "value": int(value)     })

return render\_template(

    'threshold.html',

    thresholds=thresholds,

    zones=zones  )

@app.route('/analytics')

def analytics():

if 'admin' not in session:

return redirect('/')

# Safe default if zones or counts empty

zone\_names = [z.get('name','Zone') for z in zones] if zones else ['Zone1']

zone\_counts = [len(counts) for \_ in zone\_names] if counts else [0 for \_ in zone\_names]

  return render\_template('analytics.html',

                       zone\_names=zone\_names,

                       zone\_counts=zone\_counts)

@app.route('/reports')

def reports():

if 'admin' not in session:

return redirect('/')

return render\_template('reports.html')

@app.route('/export/daily')

def export\_daily():

path = 'exports/daily.csv'

with open(path,'w',newline='') as f:

writer = csv.writer(f)

writer.writerow(['ID','Zone','Time'])

for c in counts: # counts model से data

writer.writerow([[c.id](http://c.id/), c.zone, c.time])

return send\_file(path, as\_attachment=True)

@app.route('/export/threshold')

def export\_threshold():

path = 'exports/threshold.csv'

for t in thresholds:

writer.writerow([t['zone'], t['value'], t['time']])

return send\_file(path, as\_attachment=True)

@app.route('/export/camera')

def export\_camera():

path = 'exports/camera.csv'

with open(path,'w',newline='') as f:

writer = csv.writer(f)

writer.writerow(['id','name','source','status'])

for cam in cameras:

writer.writerow([[cam.id](http://cam.id/) ,[cam.name](http://cam.name/), cam.source, cam.status])

return send\_file(path, as\_attachment=True)

@app.route("/logout")

def logout():

session.clear()

return redirect("/")

if name == "main":

app.run(debug=True)

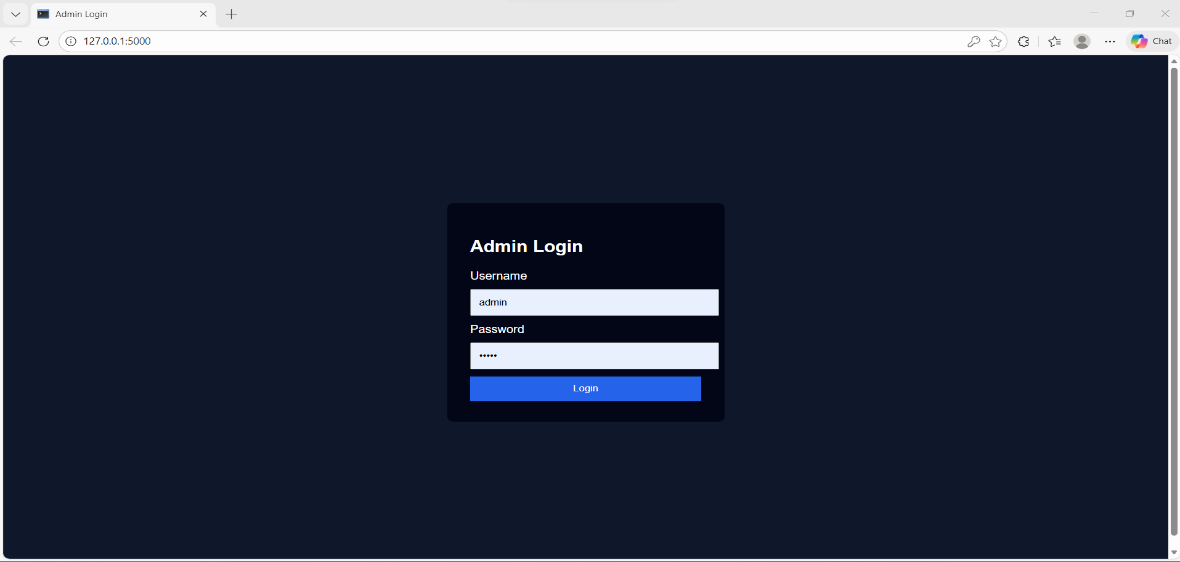
* **Code Explanation**  
  This code implements the Admin Panel, System Management, and Reporting module of the CrowdCount project.
* A Flask-based admin panel is created to manage the overall crowd monitoring system securely.
* An admin login system is implemented using session management to restrict unauthorized access.
* The admin can add and manage cameras, select the active camera, and stream live video feeds on the dashboard.
* People detection, tracking, zone visualization, and heatmap generation are integrated into the live video stream for monitoring.
* Zones and thresholds are managed dynamically by the admin to control crowd alert limits.
* When the number of people in any zone exceeds the defined threshold, alerts are generated and logged with timestamps.
* Crowd-related data such as zone counts, threshold breaches, and camera details are stored for analytics.
* The admin panel provides analytics views to observe zone-wise crowd distribution.
* Multiple report export options are implemented, allowing the admin to download data in CSV format for:

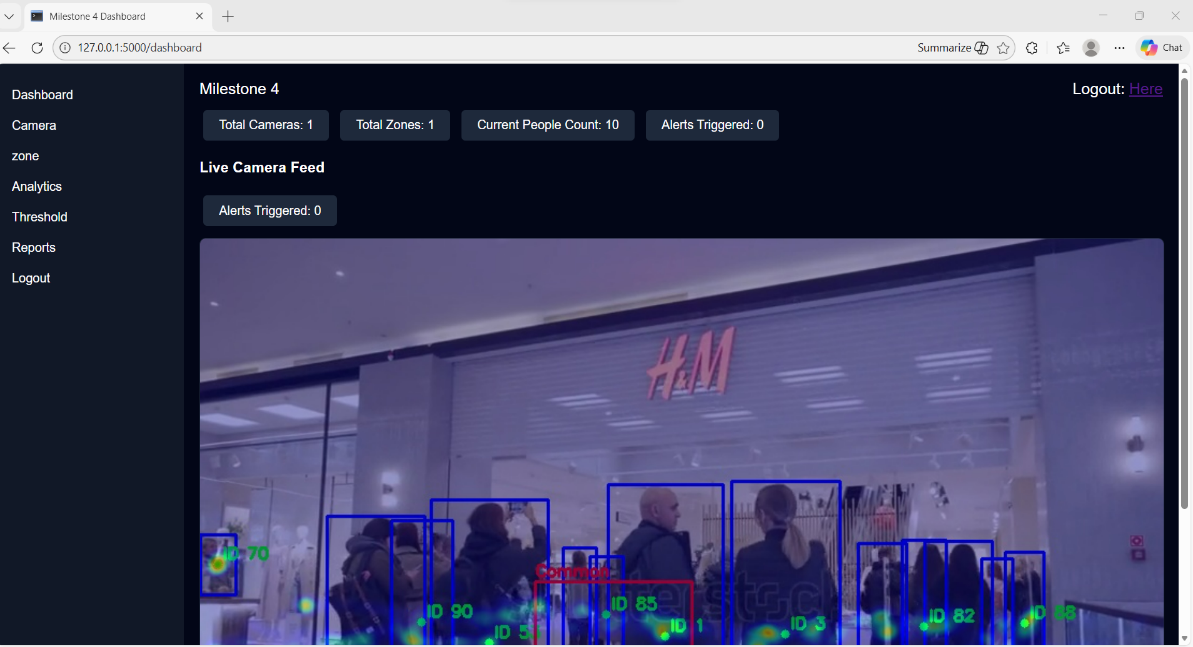
Daily crowd counts

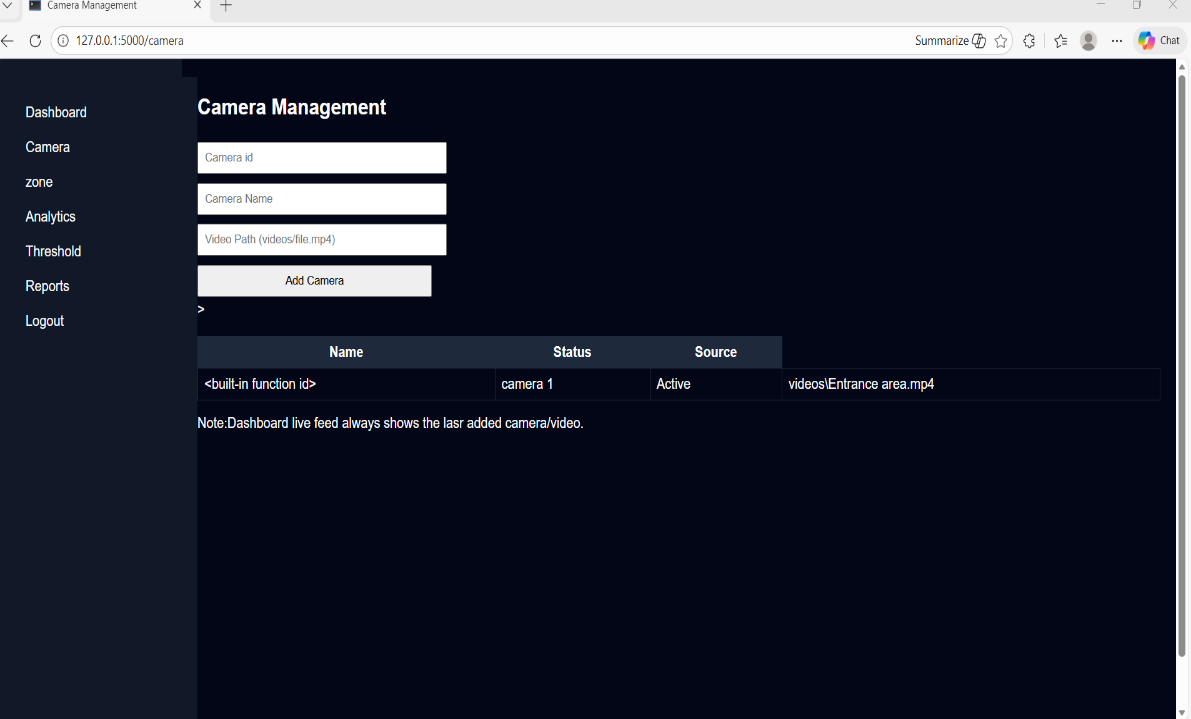
Threshold alerts

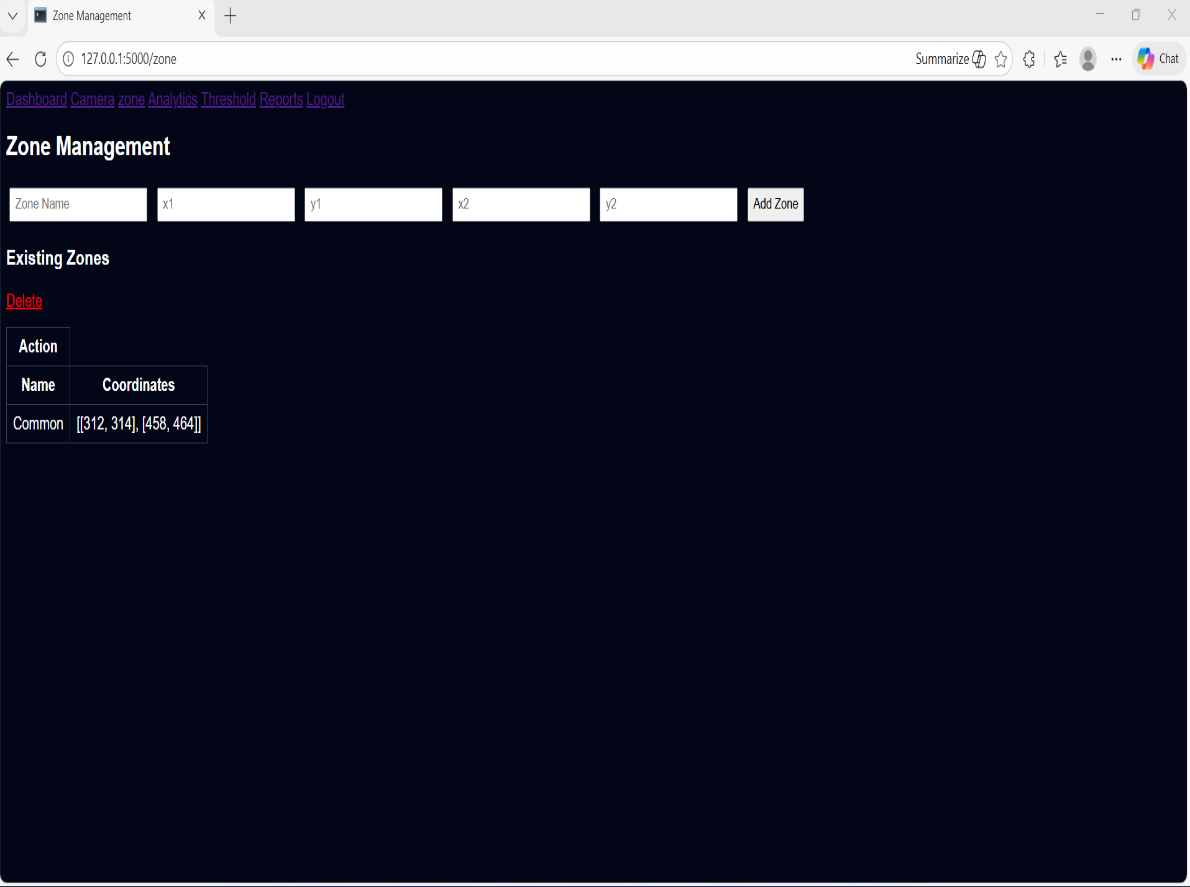
Camera information

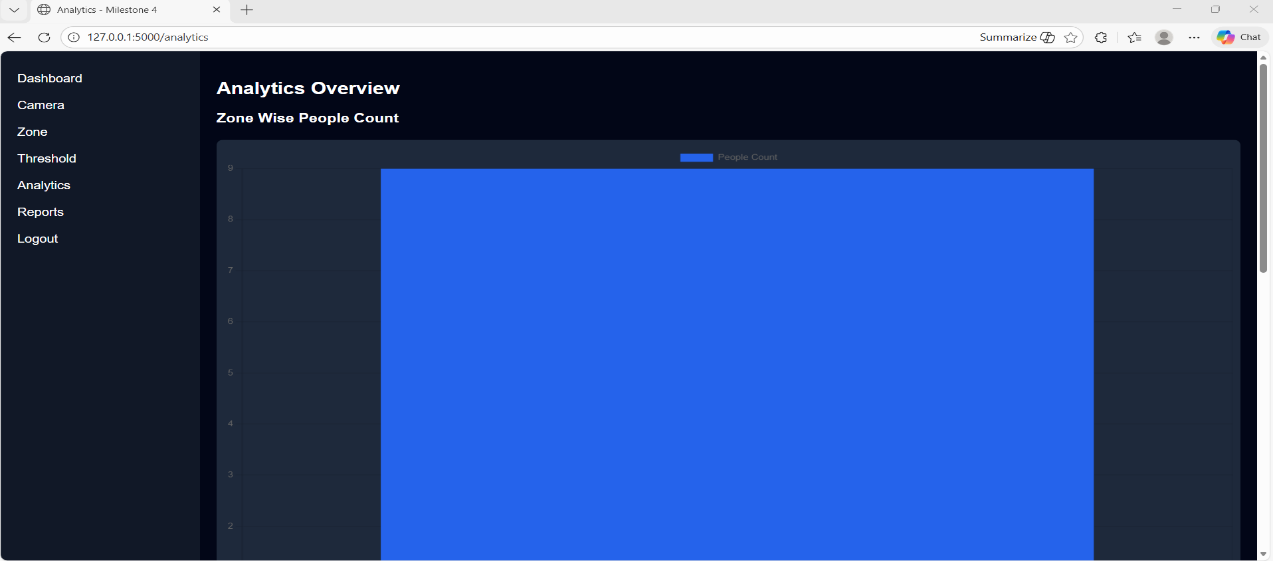
* A logout feature ensures secure session termination after admin usage.
* **Output Screenshots**

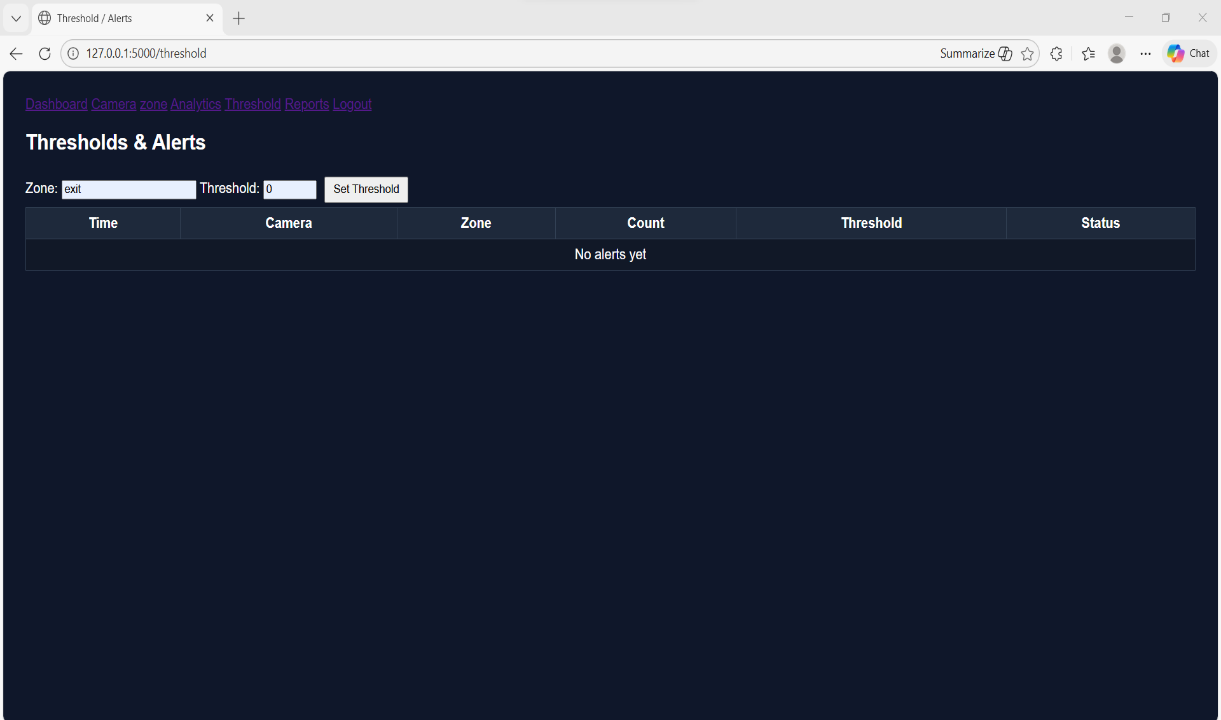


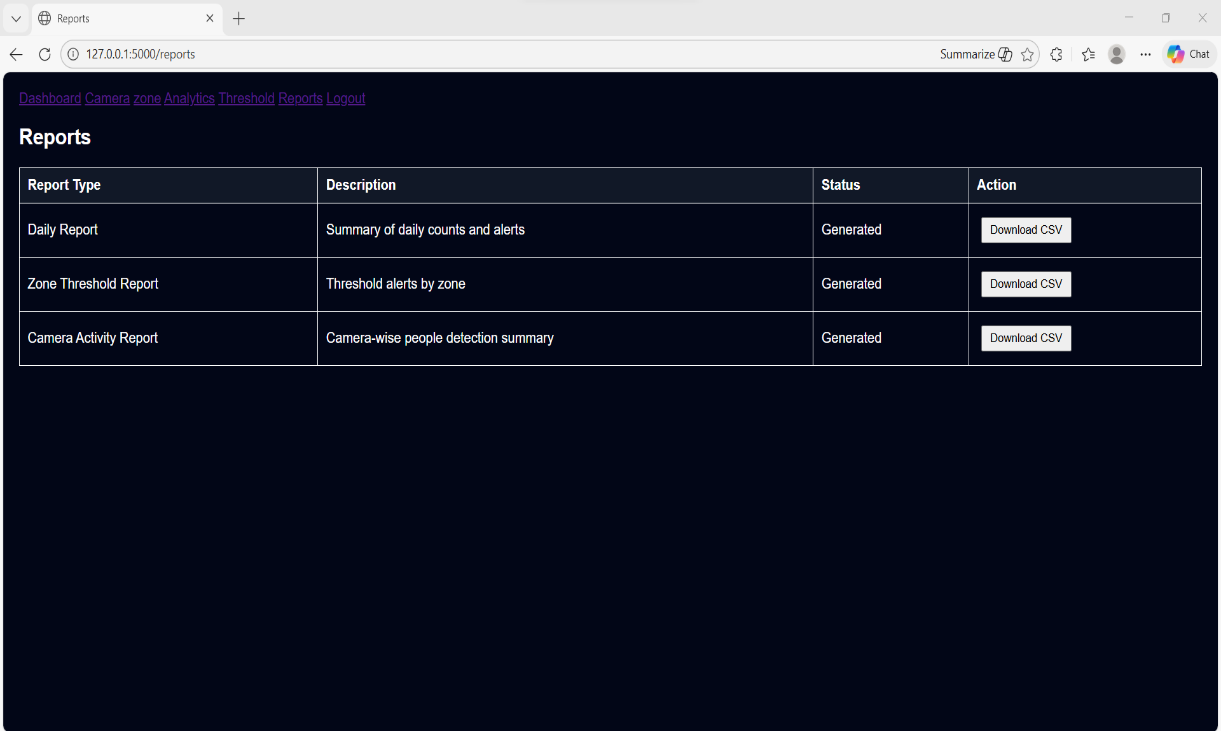












* **Final Conclusion**

The CrowdCount project successfully implements a video-based people counting system with zone management, real-time analytics, and secure administrative control. All milestones were completed as planned, and the system meets the project objectives effectively.