Infosys Springboard Virtual Internship

CrowdCount – People Counting Using Video Analysis

Name : Pragati Yashwant Andewadi

Batch : 6

Mentor : Pavithra Yuvaraj

1. Introduction

With the rapid growth of urban infrastructure and public spaces, monitoring crowd density has become a crucial requirement for ensuring public safety, efficient resource management, and real-time decision-making. Manual crowd monitoring methods are often inaccurate, time-consuming, and impractical in large or dynamic environments.

The **CrowdCount Dashboard System** is a web-based application designed to visualize and monitor crowd data in real time. It integrates backend data processing with a responsive frontend dashboard to display live people count, cumulative count, zone-wise statistics, and time-based trends. The system also incorporates **role-based authentication**, allowing controlled access for administrators and regular users.

This project provides a scalable and user-friendly solution for real-time crowd analysis and monitoring.

2. Objectives

The main objectives of the CrowdCount Dashboard System are:

1. **Real-Time Crowd Monitoring**
   To display live crowd counts and cumulative counts updated dynamically from the backend.
2. **Zone-wise Crowd Analysis**
   To monitor and display crowd distribution across multiple predefined zones.
3. **Threshold-Based Alerting**
   To highlight zones where the crowd count exceeds a configurable threshold, enabling quick attention and preventive action.
4. **Time-Series Visualization**
   To visualize crowd trends over time using a compact and readable line graph.
5. **Role-Based Access Control**
   - **Admin users** can modify thresholds and download crowd data logs.
   - **Regular users** have view-only access to the dashboard.
6. **Data Export Capability**
   To allow administrators to download crowd data logs in CSV format for further analysis and reporting.
7. **User-Friendly Interface**
   To provide a modern, intuitive, and visually appealing dashboard interface for easy interpretation of data.

3. System Workflow

The workflow of the CrowdCount Dashboard System follows a structured and modular approach:

Step 1: User Authentication

- When the application loads, a login overlay is displayed.
- Users enter their credentials.
- The backend validates the credentials and assigns a role (Admin/User).
- Only authenticated users can access the dashboard.

Step 2: Data Retrieval

- The backend reads real-time crowd data from a structured JSON file.
- This data includes:
  - Total live count
  - Total cumulative count
  - Zone-wise live and total counts
  - Timestamp of last update

Step 3: Dashboard Rendering

- After successful login, the dashboard becomes visible.
- Key metrics such as live count, total count, and last update time are displayed.
- A line graph visualizes the people count over time.

Step 4: Zone-wise Monitoring

- Crowd data is displayed in a tabular format for each zone.

- Zones exceeding the defined threshold are visually highlighted.
- This enables quick identification of overcrowded areas.

Step 5: Admin Controls

- Admin users can:
  - Modify the threshold value.
  - Change refresh frequency.
  - Download crowd data logs in CSV format.
- User-level access restricts these controls.

Step 6: Continuous Updates

- The dashboard automatically refreshes at a user-defined interval.
- New data is fetched and displayed without reloading the page.

4. Technology Stack

The CrowdCount Dashboard System is developed using a combination of backend, frontend, and visualization technologies to ensure real-time data processing, secure access, and an interactive user interface.

Backend

- **Python**
  Used as the core programming language for backend logic, data processing, and file handling due to its simplicity and wide ecosystem.
- **Flask**
  A lightweight Python web framework used to build REST APIs, manage routing, and implement session-based authentication with role-based access control.
- **JSON**
  Used as a lightweight data storage format to store live crowd data, cumulative counts, zone-wise information, and timestamps.

Frontend

- **HTML5**
  Used to structure the dashboard layout, login interface, tables, and data visualization components.
- **CSS3**
  Used to design a modern, responsive, and visually appealing dark-themed user interface with proper color coordination and layout styling.

- **JavaScript (Vanilla JavaScript)**
  Used to fetch real-time data from backend APIs, dynamically update the dashboard, handle user interactions, and manage role-based UI behavior.

Data Visualization

- **Chart.js**
  A JavaScript library used to display a compact and responsive line graph representing people count over time.

Authentication & Security

- **Session-Based Authentication**
  Implemented using Flask sessions to securely manage user login states.
- **Role-Based Access Control (RBAC)**
  Differentiates between Admin and User roles to restrict sensitive actions such as threshold modification and CSV data download.

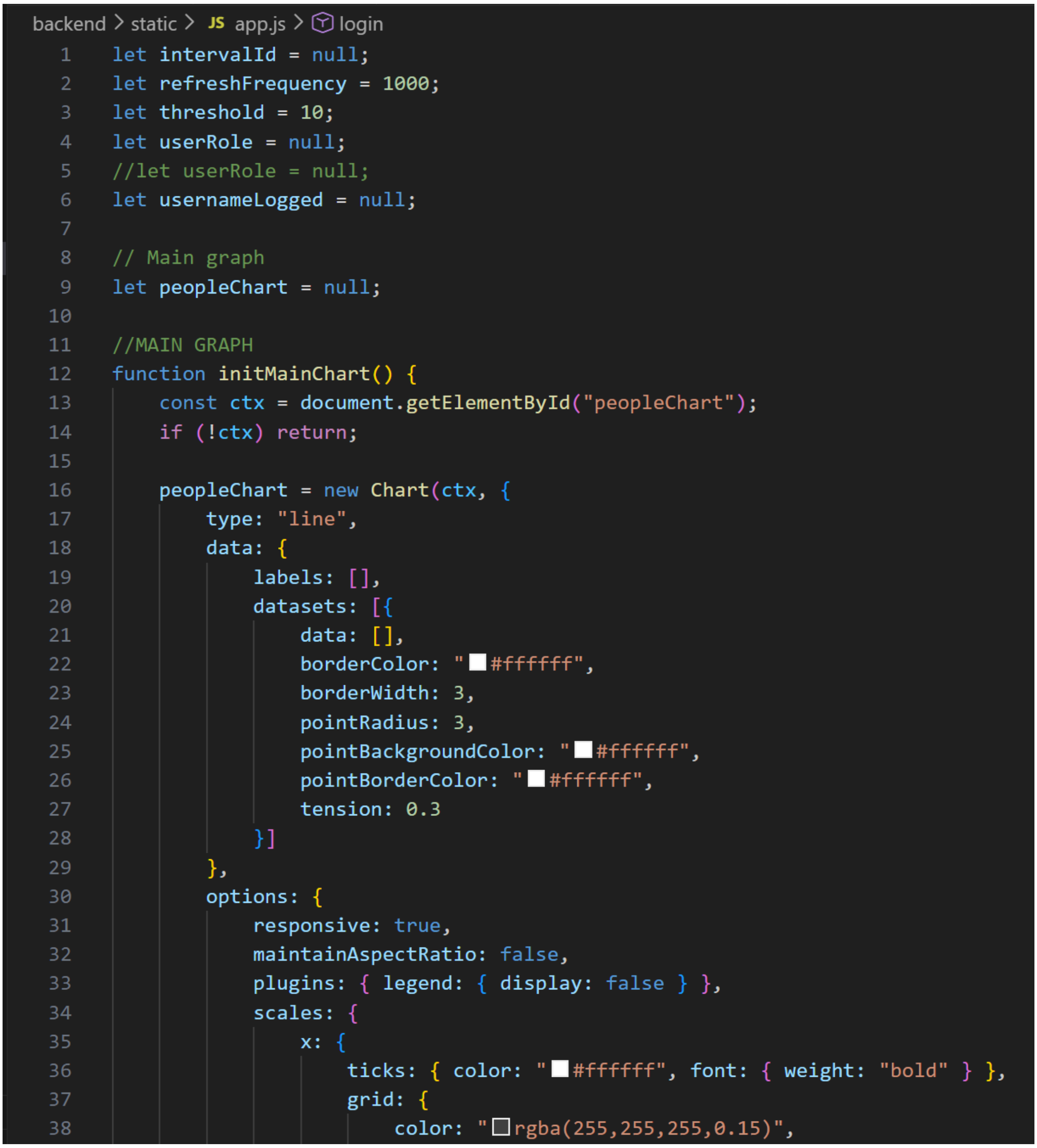
Development Environment

- **Flask Development Server**
  Used to run and test the application locally during development.
- **Web Browser**
  Used for testing and debugging the frontend interface and real-time dashboard behavior.

Code Snippets:

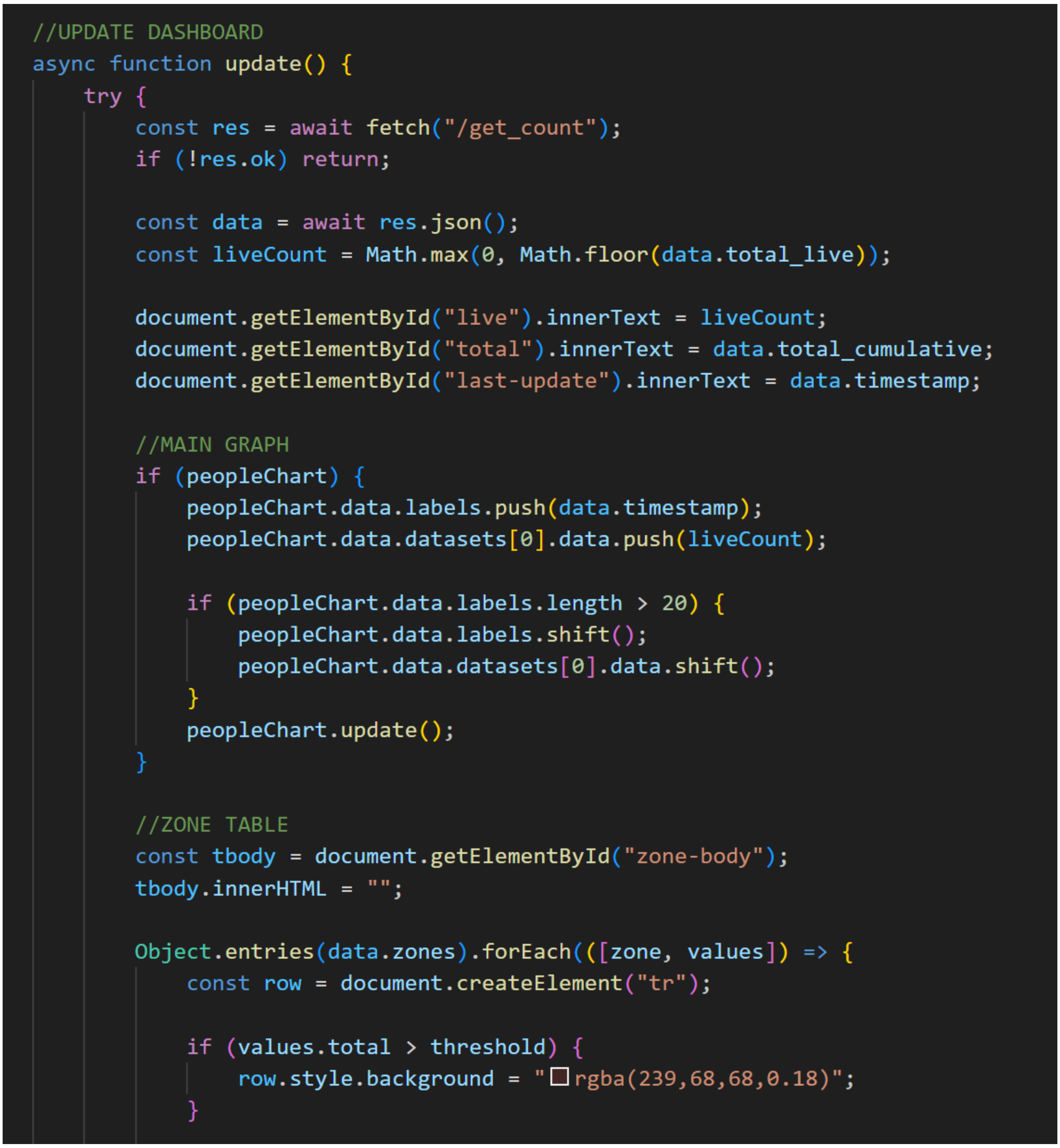
Backend:

app.js:



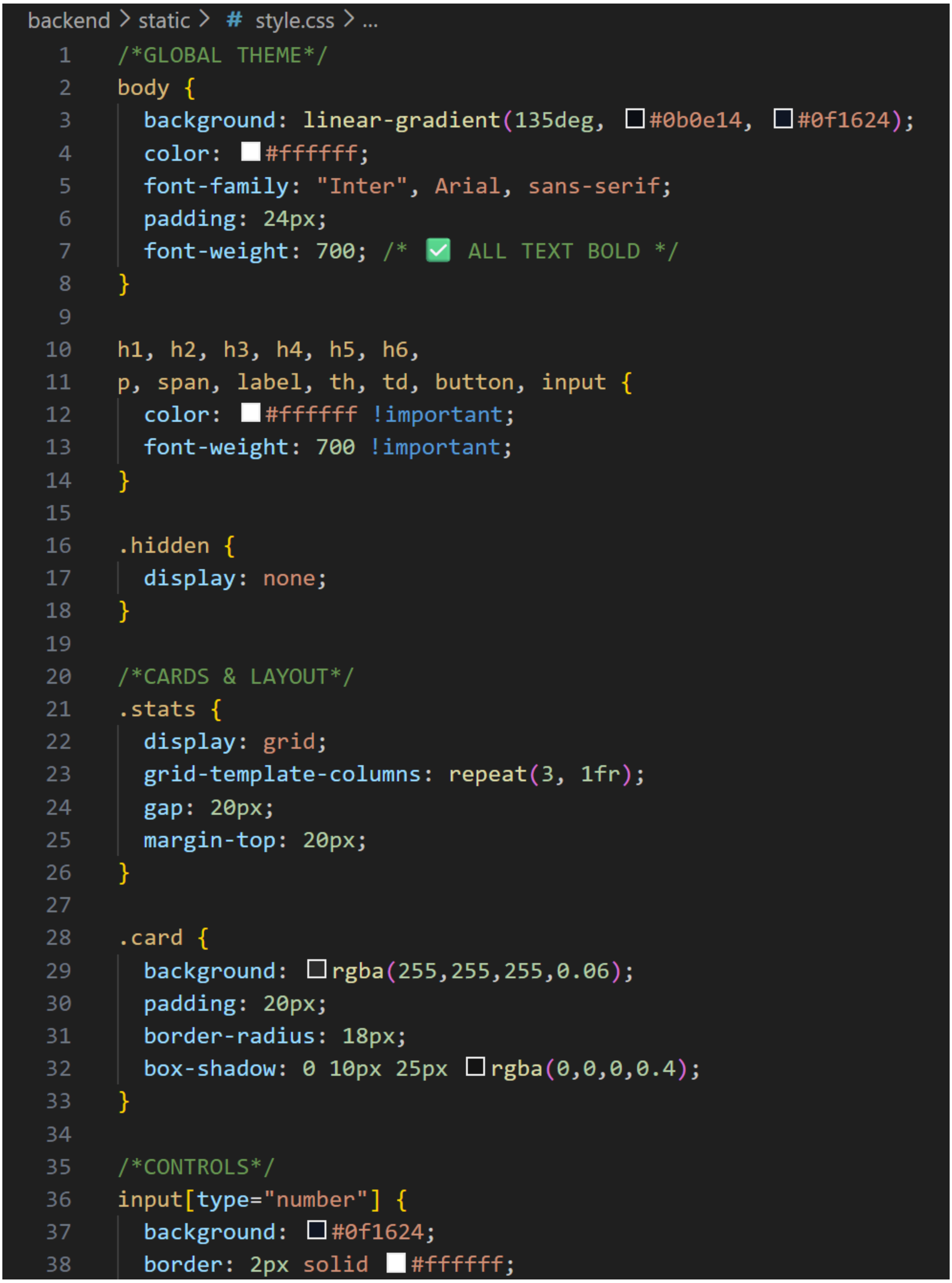
backend > static > JS app.js > login

```javascript
let intervalId = null;
let refreshFrequency = 1000;
let threshold = 10;
let userRole = null;
//let userRole = null;
let usernameLogged = null;

// Main graph
let peopleChart = null;

//MAIN GRAPH
function initMainChart() {
    const ctx = document.getElementById("peopleChart");
    if (!ctx) return;

    peopleChart = new Chart(ctx, {
        type: "line",
        data: {
            labels: [],
            datasets: [{
                data: [],
                borderColor: "#ffffff",
                borderWidth: 3,
                pointRadius: 3,
                pointBackgroundColor: "#ffffff",
                pointBorderColor: "#ffffff",
                tension: 0.3
            }]
        },
        options: {
            responsive: true,
            maintainAspectRatio: false,
            plugins: { legend: { display: false } },
            scales: {
                x: {
                    ticks: { color: "#ffffff", font: { weight: "bold" } },
                    grid: {
                        color: "rgba(255,255,255,0.15)",
```



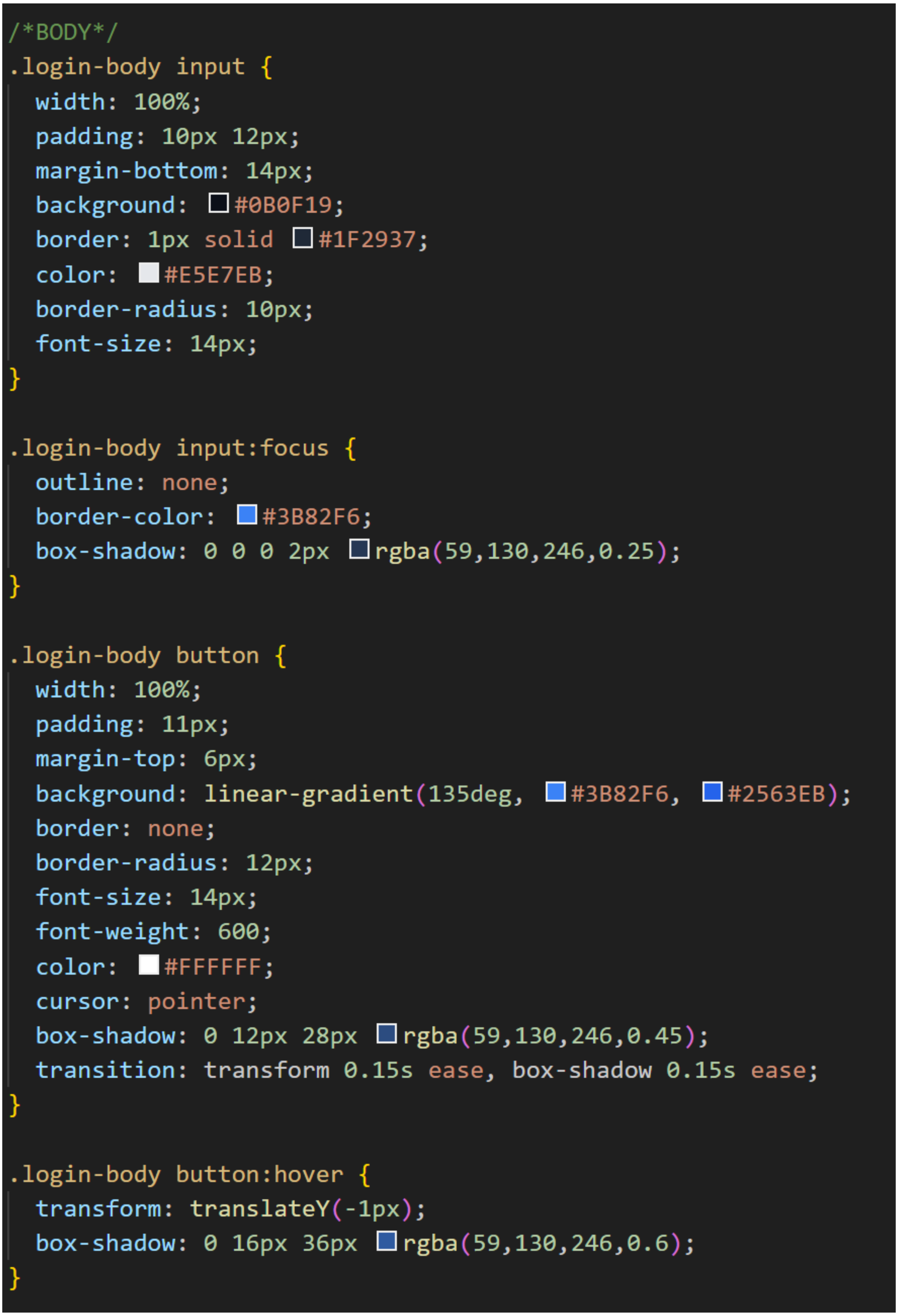
```javascript
//UPDATE DASHBOARD
async function update() {
    try {
        const res = await fetch("/get_count");
        if (!res.ok) return;

        const data = await res.json();
        const liveCount = Math.max(0, Math.floor(data.total_live));

        document.getElementById("live").innerText = liveCount;
        document.getElementById("total").innerText = data.total_cumulative;
        document.getElementById("last-update").innerText = data.timestamp;

        //MAIN GRAPH
        if (peopleChart) {
            peopleChart.data.labels.push(data.timestamp);
            peopleChart.data.datasets[0].data.push(liveCount);

            if (peopleChart.data.labels.length > 20) {
                peopleChart.data.labels.shift();
                peopleChart.data.datasets[0].data.shift();
            }
            peopleChart.update();
        }

        //ZONE TABLE
        const tbody = document.getElementById("zone-body");
        tbody.innerHTML = "";

        Object.entries(data.zones).forEach(([zone, values]) => {
            const row = document.createElement("tr");

            if (values.total > threshold) {
                row.style.background = "rgba(239,68,68,0.18)";
            }
```

style.css:



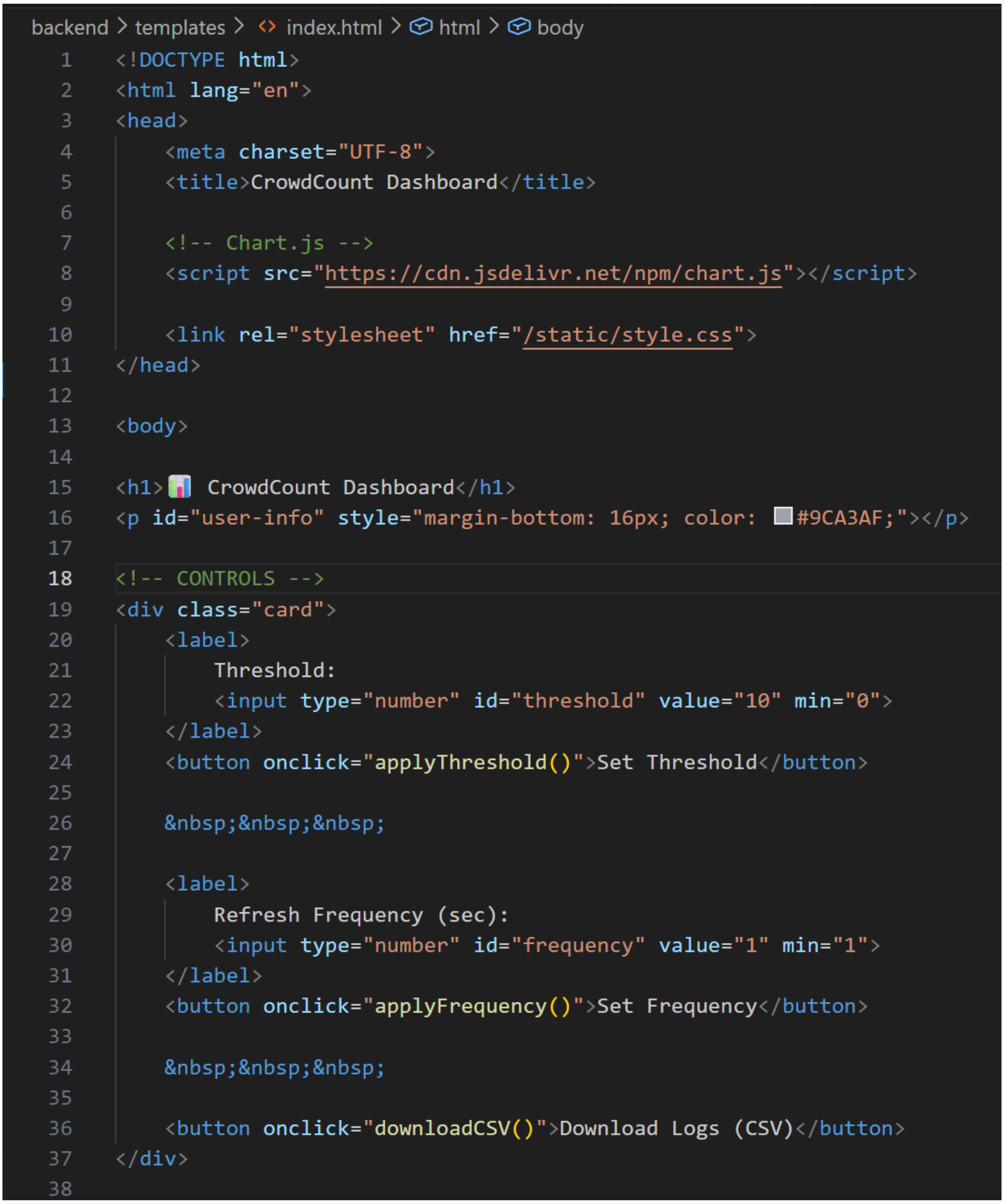
backend > static > # style.css > ...

```css
/*GLOBAL THEME*/
body {
  background: linear-gradient(135deg, #0b0e14, #0f1624);
  color: #ffffff;
  font-family: "Inter", Arial, sans-serif;
  padding: 24px;
  font-weight: 700; /* ✅ ALL TEXT BOLD */
}

h1, h2, h3, h4, h5, h6,
p, span, label, th, td, button, input {
  color: #ffffff !important;
  font-weight: 700 !important;
}

.hidden {
  display: none;
}

/*CARDS & LAYOUT*/
.stats {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  gap: 20px;
  margin-top: 20px;
}

.card {
  background: rgba(255,255,255,0.06);
  padding: 20px;
  border-radius: 18px;
  box-shadow: 0 10px 25px rgba(0,0,0,0.4);
}

/*CONTROLS*/
input[type="number"] {
  background: #0f1624;
  border: 2px solid #ffffff;
```

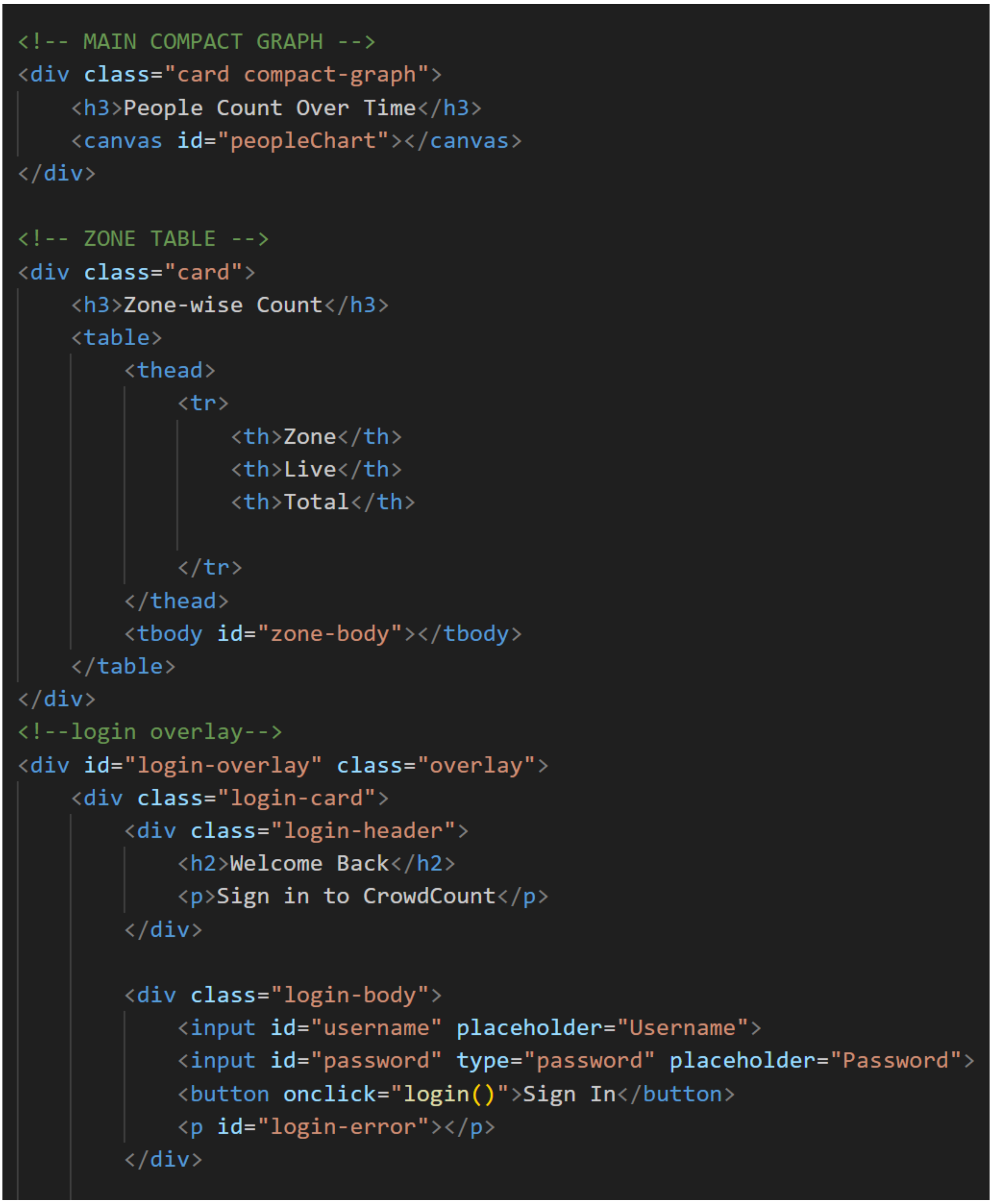


```css
/*BODY*/
.login-body input {
  width: 100%;
  padding: 10px 12px;
  margin-bottom: 14px;
  background: #0B0F19;
  border: 1px solid #1F2937;
  color: #E5E7EB;
  border-radius: 10px;
  font-size: 14px;
}

.login-body input:focus {
  outline: none;
  border-color: #3B82F6;
  box-shadow: 0 0 0 2px rgba(59,130,246,0.25);
}

.login-body button {
  width: 100%;
  padding: 11px;
  margin-top: 6px;
  background: linear-gradient(135deg, #3B82F6, #2563EB);
  border: none;
  border-radius: 12px;
  font-size: 14px;
  font-weight: 600;
  color: #FFFFFF;
  cursor: pointer;
  box-shadow: 0 12px 28px rgba(59,130,246,0.45);
  transition: transform 0.15s ease, box-shadow 0.15s ease;
}

.login-body button:hover {
  transform: translateY(-1px);
  box-shadow: 0 16px 36px rgba(59,130,246,0.6);
}
```

index.html:

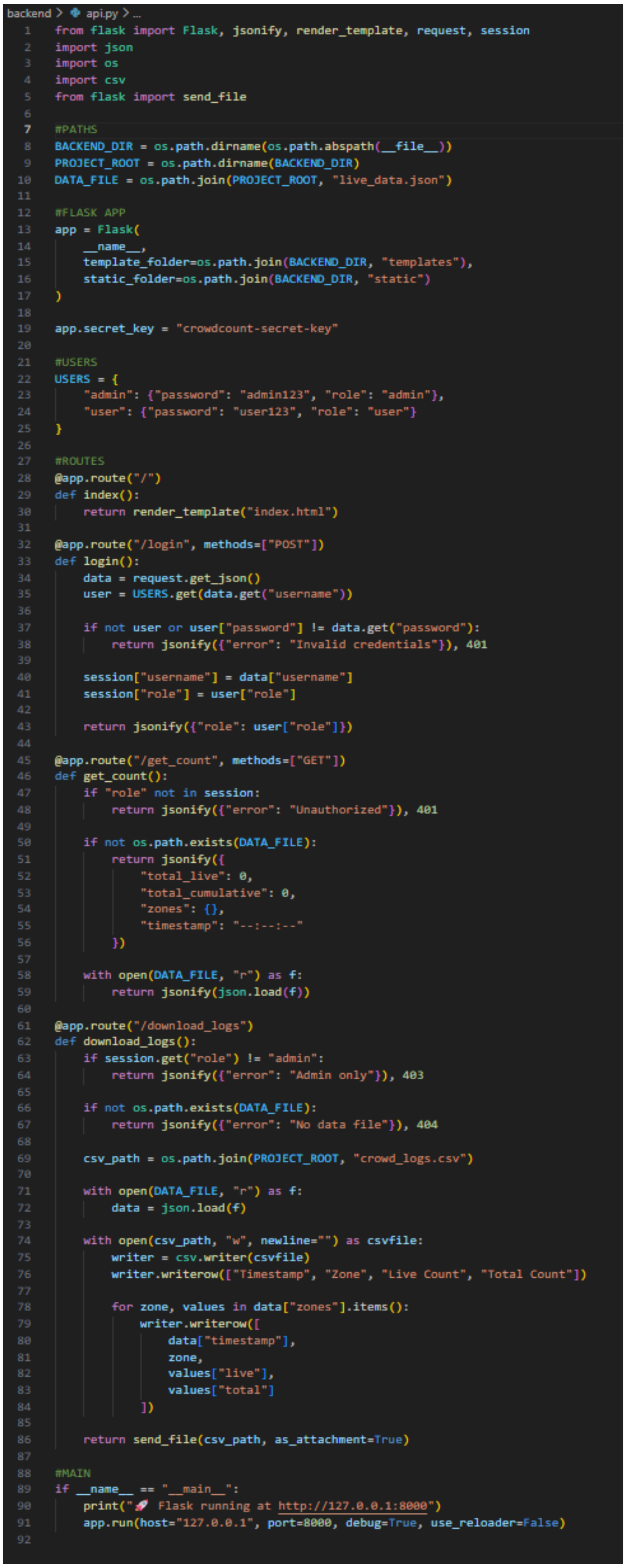


```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>CrowdCount Dashboard</title>

    <!-- Chart.js -->
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>

    <link rel="stylesheet" href="/static/style.css">
</head>

<body>

<h1>📊 CrowdCount Dashboard</h1>
<p id="user-info" style="margin-bottom: 16px; color: #9CA3AF;"></p>

<!-- CONTROLS -->
<div class="card">
    <label>
        Threshold:
        <input type="number" id="threshold" value="10" min="0">
    </label>
    <button onclick="applyThreshold()">Set Threshold</button>

    &nbsp;&nbsp;&nbsp;

    <label>
        Refresh Frequency (sec):
        <input type="number" id="frequency" value="1" min="1">
    </label>
    <button onclick="applyFrequency()">Set Frequency</button>

    &nbsp;&nbsp;&nbsp;

    <button onclick="downloadCSV()">Download Logs (CSV)</button>
</div>

```



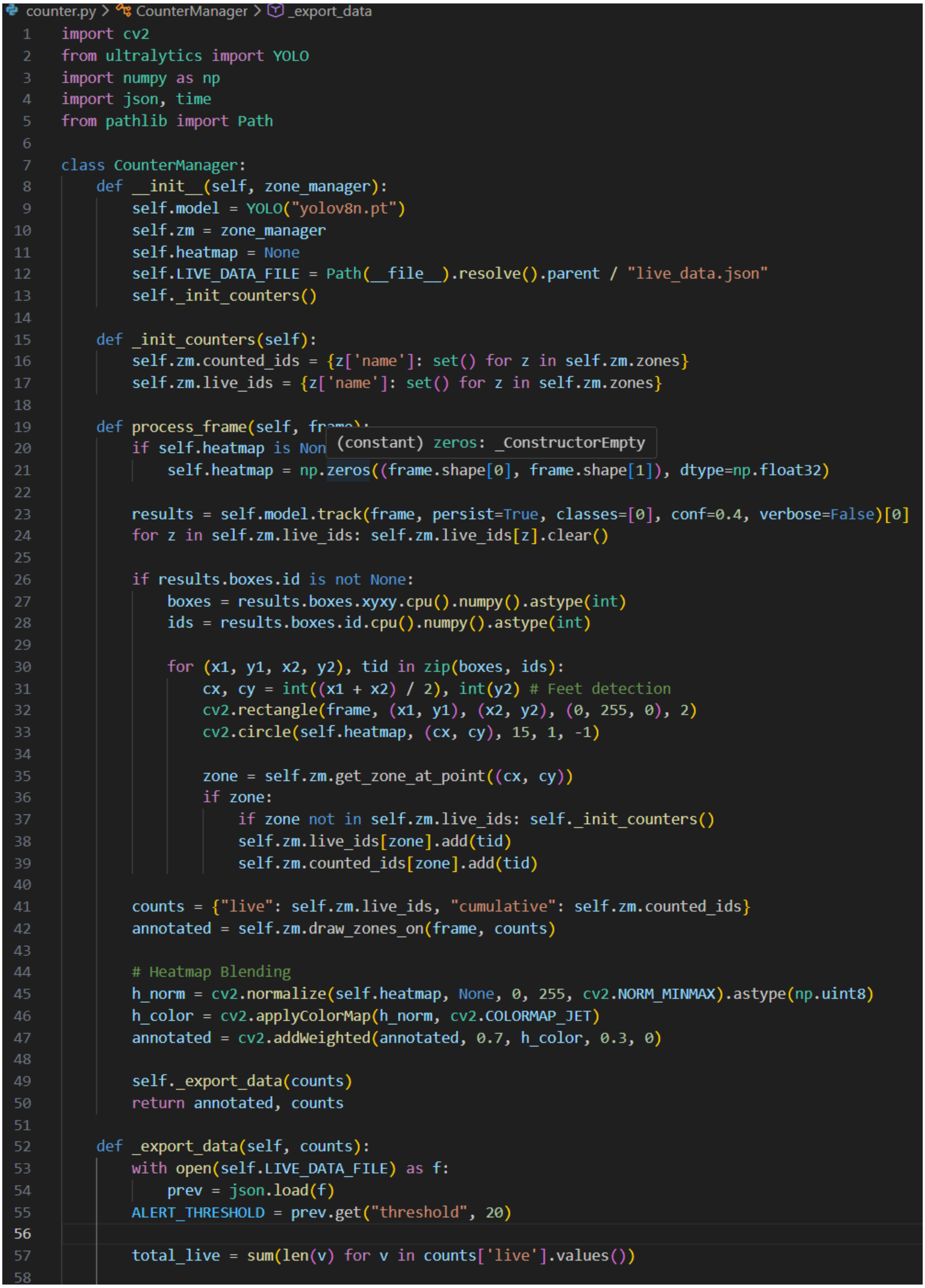
```html
<!-- MAIN COMPACT GRAPH -->
<div class="card compact-graph">
    <h3>People Count Over Time</h3>
    <canvas id="peopleChart"></canvas>
</div>

<!-- ZONE TABLE -->
<div class="card">
    <h3>Zone-wise Count</h3>
    <table>
        <thead>
            <tr>
                <th>Zone</th>
                <th>Live</th>
                <th>Total</th>

            </tr>
        </thead>
        <tbody id="zone-body"></tbody>
    </table>
</div>
<!--login overlay-->
<div id="login-overlay" class="overlay">
    <div class="login-card">
        <div class="login-header">
            <h2>Welcome Back</h2>
            <p>Sign in to CrowdCount</p>
        </div>

        <div class="login-body">
            <input id="username" placeholder="Username">
            <input id="password" type="password" placeholder="Password">
            <button onclick="login()">Sign In</button>
            <p id="login-error"></p>
        </div>
```

api.py:



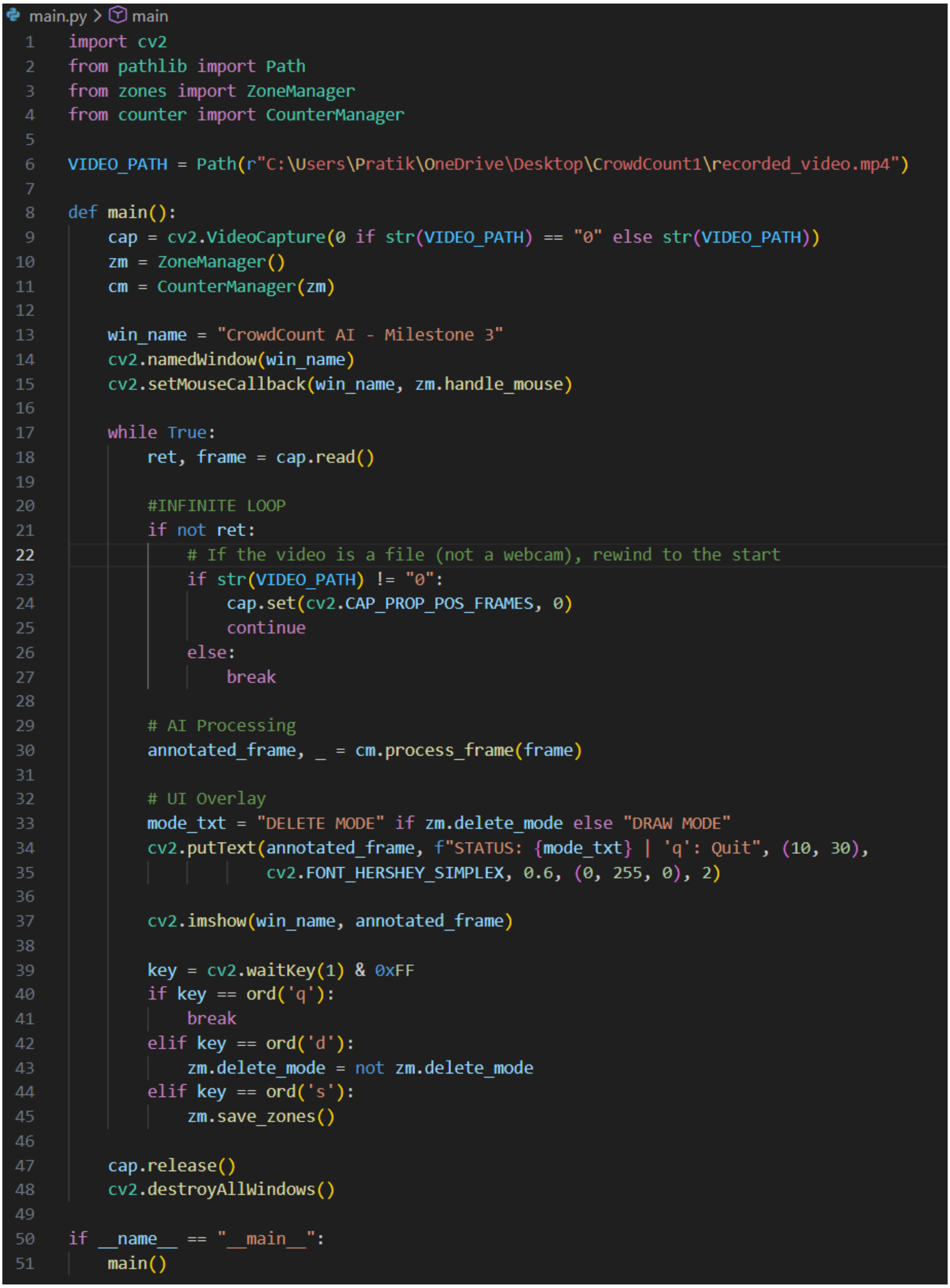
```python
from flask import Flask, jsonify, render_template, request, session
import json
import os
import csv
from flask import send_file

#PATHS
BACKEND_DIR = os.path.dirname(os.path.abspath(__file__))
PROJECT_ROOT = os.path.dirname(BACKEND_DIR)
DATA_FILE = os.path.join(PROJECT_ROOT, "live_data.json")

#FLASK APP
app = Flask(
    __name__,
    template_folder=os.path.join(BACKEND_DIR, "templates"),
    static_folder=os.path.join(BACKEND_DIR, "static")
)

app.secret_key = "crowdcount-secret-key"

#USERS
USERS = {
    "admin": {"password": "admin123", "role": "admin"},
    "user": {"password": "user123", "role": "user"}
}

#ROUTES
@app.route("/")
def index():
    return render_template("index.html")

@app.route("/login", methods=["POST"])
def login():
    data = request.get_json()
    user = USERS.get(data.get("username"))

    if not user or user["password"] != data.get("password"):
        return jsonify({"error": "Invalid credentials"}), 401

    session["username"] = data["username"]
    session["role"] = user["role"]

    return jsonify({"role": user["role"]})

@app.route("/get_count", methods=["GET"])
def get_count():
    if "role" not in session:
        return jsonify({"error": "Unauthorized"}), 401

    if not os.path.exists(DATA_FILE):
        return jsonify({
            "total_live": 0,
            "total_cumulative": 0,
            "zones": {},
            "timestamp": "--:--:--"
        })

    with open(DATA_FILE, "r") as f:
        return jsonify(json.load(f))

@app.route("/download_logs")
def download_logs():
    if session.get("role") != "admin":
        return jsonify({"error": "Admin only"}), 403

    if not os.path.exists(DATA_FILE):
        return jsonify({"error": "No data file"}), 404

    csv_path = os.path.join(PROJECT_ROOT, "crowd_logs.csv")

    with open(DATA_FILE, "r") as f:
        data = json.load(f)

    with open(csv_path, "w", newline="") as csvfile:
        writer = csv.writer(csvfile)
        writer.writerow(["Timestamp", "Zone", "Live Count", "Total Count"])

        for zone, values in data["zones"].items():
            writer.writerow([
                data["timestamp"],
                zone,
                values["live"],
                values["total"]
            ])

    return send_file(csv_path, as_attachment=True)

#MAIN
if __name__ == "__main__":
    print("🚀 Flask running at http://127.0.0.1:8000")
    app.run(host="127.0.0.1", port=8000, debug=True, use_reloader=False)
```

counter.py:


counter.py > CounterManager > _export_data
(constant) zeros: _ConstructorEmpty

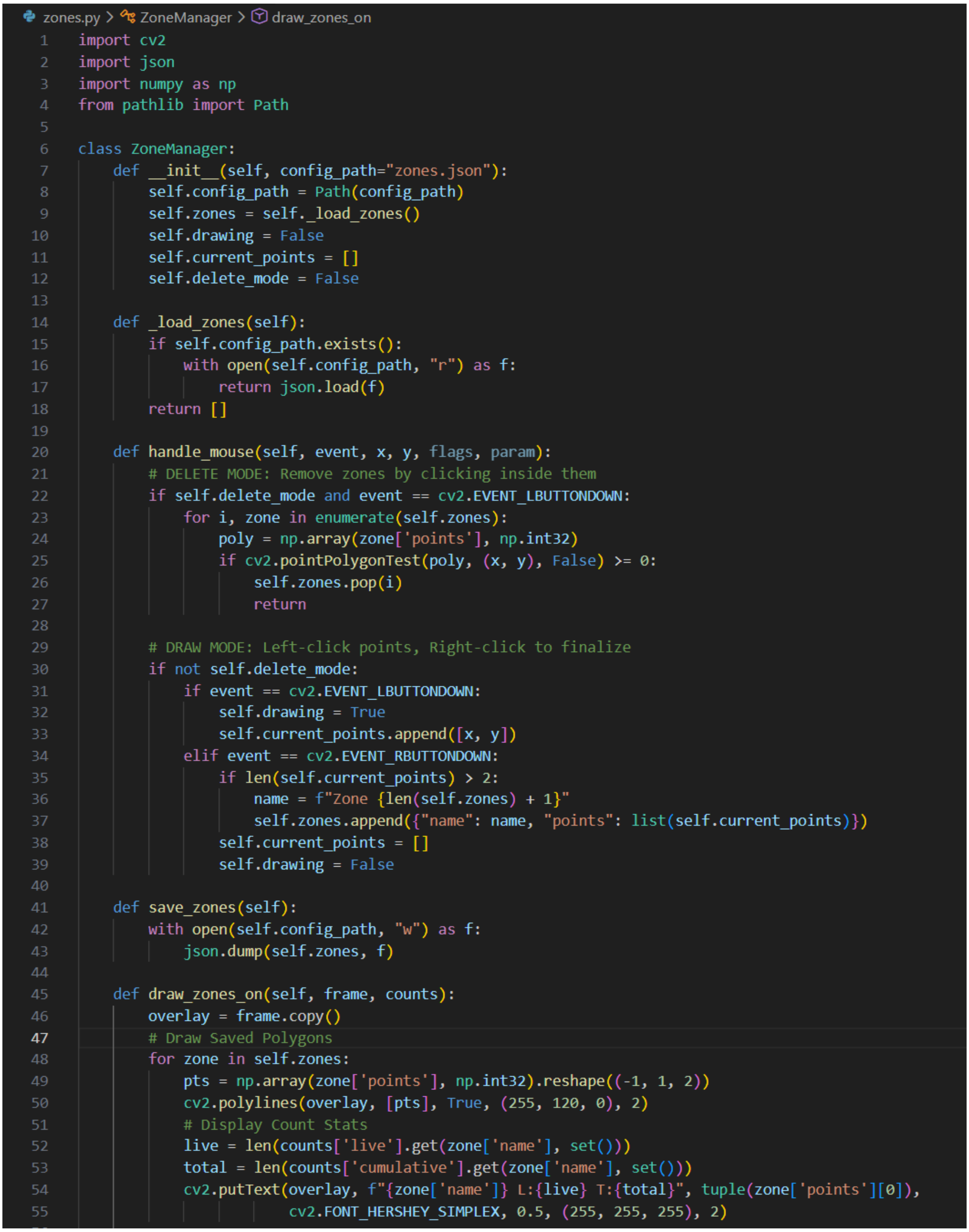
```python
import cv2
from ultralytics import YOLO
import numpy as np
import json, time
from pathlib import Path

class CounterManager:
    def __init__(self, zone_manager):
        self.model = YOLO("yolov8n.pt")
        self.zm = zone_manager
        self.heatmap = None
        self.LIVE_DATA_FILE = Path(__file__).resolve().parent / "live_data.json"
        self._init_counters()

    def _init_counters(self):
        self.zm.counted_ids = {z['name']: set() for z in self.zm.zones}
        self.zm.live_ids = {z['name']: set() for z in self.zm.zones}

    def process_frame(self, frame):
        if self.heatmap is Non
            self.heatmap = np.zeros((frame.shape[0], frame.shape[1]), dtype=np.float32)

        results = self.model.track(frame, persist=True, classes=[0], conf=0.4, verbose=False)[0]
        for z in self.zm.live_ids: self.zm.live_ids[z].clear()

        if results.boxes.id is not None:
            boxes = results.boxes.xyxy.cpu().numpy().astype(int)
            ids = results.boxes.id.cpu().numpy().astype(int)

            for (x1, y1, x2, y2), tid in zip(boxes, ids):
                cx, cy = int((x1 + x2) / 2), int(y2) # Feet detection
                cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
                cv2.circle(self.heatmap, (cx, cy), 15, 1, -1)

                zone = self.zm.get_zone_at_point((cx, cy))
                if zone:
                    if zone not in self.zm.live_ids: self._init_counters()
                    self.zm.live_ids[zone].add(tid)
                    self.zm.counted_ids[zone].add(tid)

        counts = {"live": self.zm.live_ids, "cumulative": self.zm.counted_ids}
        annotated = self.zm.draw_zones_on(frame, counts)

        # Heatmap Blending
        h_norm = cv2.normalize(self.heatmap, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)
        h_color = cv2.applyColorMap(h_norm, cv2.COLORMAP_JET)
        annotated = cv2.addWeighted(annotated, 0.7, h_color, 0.3, 0)

        self._export_data(counts)
        return annotated, counts

    def _export_data(self, counts):
        with open(self.LIVE_DATA_FILE) as f:
            prev = json.load(f)
        ALERT_THRESHOLD = prev.get("threshold", 20)

        total_live = sum(len(v) for v in counts['live'].values())

```

main.py:



```python
import cv2
from pathlib import Path
from zones import ZoneManager
from counter import CounterManager

VIDEO_PATH = Path(r"C:\Users\Pratik\OneDrive\Desktop\CrowdCount1\recorded_video.mp4")

def main():
    cap = cv2.VideoCapture(0 if str(VIDEO_PATH) == "0" else str(VIDEO_PATH))
    zm = ZoneManager()
    cm = CounterManager(zm)

    win_name = "CrowdCount AI - Milestone 3"
    cv2.namedWindow(win_name)
    cv2.setMouseCallback(win_name, zm.handle_mouse)

    while True:
        ret, frame = cap.read()

        #INFINITE LOOP
        if not ret:
            # If the video is a file (not a webcam), rewind to the start
            if str(VIDEO_PATH) != "0":
                cap.set(cv2.CAP_PROP_POS_FRAMES, 0)
                continue
            else:
                break

        # AI Processing
        annotated_frame, _ = cm.process_frame(frame)

        # UI Overlay
        mode_txt = "DELETE MODE" if zm.delete_mode else "DRAW MODE"
        cv2.putText(annotated_frame, f"STATUS: {mode_txt} | 'q': Quit", (10, 30),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)

        cv2.imshow(win_name, annotated_frame)

        key = cv2.waitKey(1) & 0xFF
        if key == ord('q'):
            break
        elif key == ord('d'):
            zm.delete_mode = not zm.delete_mode
        elif key == ord('s'):
            zm.save_zones()

    cap.release()
    cv2.destroyAllWindows()

if __name__ == "__main__":
    main()
```

zones.py:



```python
import cv2
import json
import numpy as np
from pathlib import Path

class ZoneManager:
    def __init__(self, config_path="zones.json"):
        self.config_path = Path(config_path)
        self.zones = self._load_zones()
        self.drawing = False
        self.current_points = []
        self.delete_mode = False

    def _load_zones(self):
        if self.config_path.exists():
            with open(self.config_path, "r") as f:
                return json.load(f)
        return []

    def handle_mouse(self, event, x, y, flags, param):
        # DELETE MODE: Remove zones by clicking inside them
        if self.delete_mode and event == cv2.EVENT_LBUTTONDOWN:
            for i, zone in enumerate(self.zones):
                poly = np.array(zone['points'], np.int32)
                if cv2.pointPolygonTest(poly, (x, y), False) >= 0:
                    self.zones.pop(i)
                    return

        # DRAW MODE: Left-click points, Right-click to finalize
        if not self.delete_mode:
            if event == cv2.EVENT_LBUTTONDOWN:
                self.drawing = True
                self.current_points.append([x, y])
            elif event == cv2.EVENT_RBUTTONDOWN:
                if len(self.current_points) > 2:
                    name = f"Zone {len(self.zones) + 1}"
                    self.zones.append({"name": name, "points": list(self.current_points)})
                self.current_points = []
                self.drawing = False

    def save_zones(self):
        with open(self.config_path, "w") as f:
            json.dump(self.zones, f)

    def draw_zones_on(self, frame, counts):
        overlay = frame.copy()
        # Draw Saved Polygons
        for zone in self.zones:
            pts = np.array(zone['points'], np.int32).reshape((-1, 1, 2))
            cv2.polylines(overlay, [pts], True, (255, 120, 0), 2)
            # Display Count Stats
            live = len(counts['live'].get(zone['name'], set()))
            total = len(counts['cumulative'].get(zone['name'], set()))
            cv2.putText(overlay, f"{zone['name']} L:{live} T:{total}", tuple(zone['points'][0]),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2)
```

Code Explanation

- **Backend (Flask – api.py)**
  Handles user authentication, reads live crowd data from a JSON file, provides APIs to fetch crowd statistics, and allows admin users to download data logs.

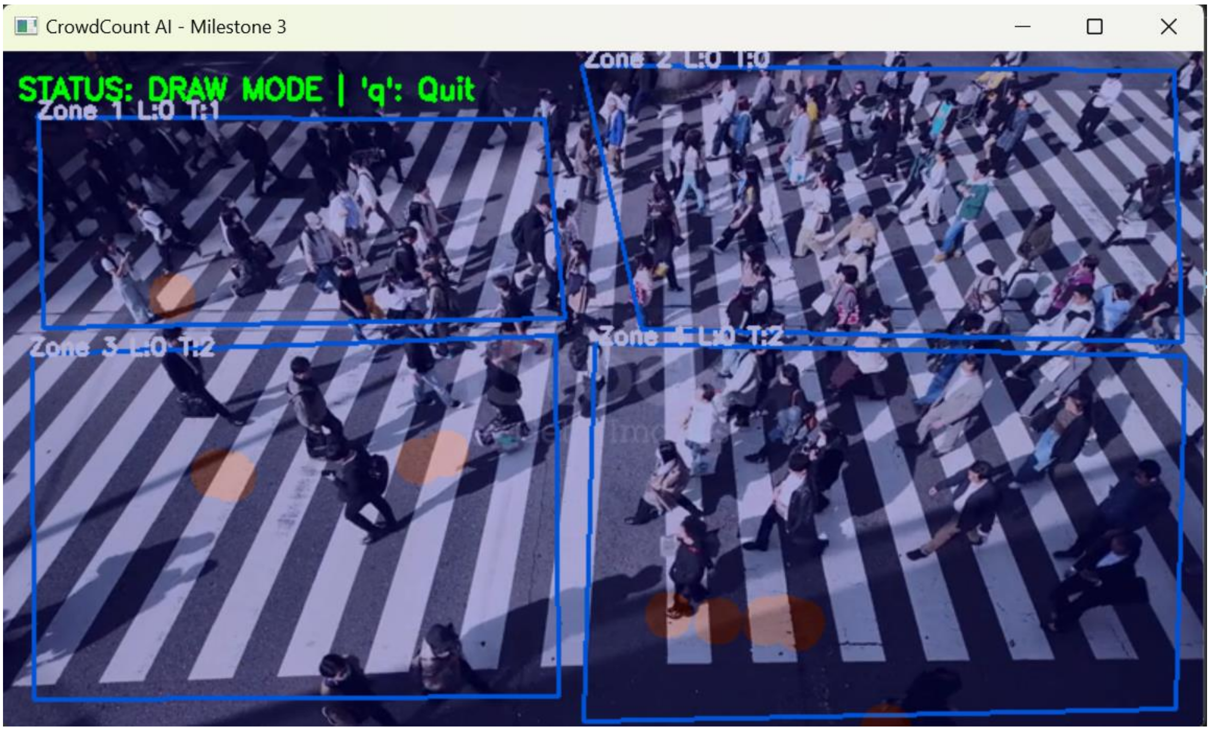
- **Frontend Structure (index.html)**
  Defines the dashboard layout, login overlay, statistics cards, zone-wise table, and the graph container.

- **Frontend Logic (app.js)**
  Manages login actions, fetches real-time data from the backend, updates dashboard values dynamically, controls refresh frequency, and applies role-based access restrictions.

- **Styling (style.css)**
  Provides a modern dark-themed user interface with responsive cards, tables, and a visually appealing login screen.

Dashboard

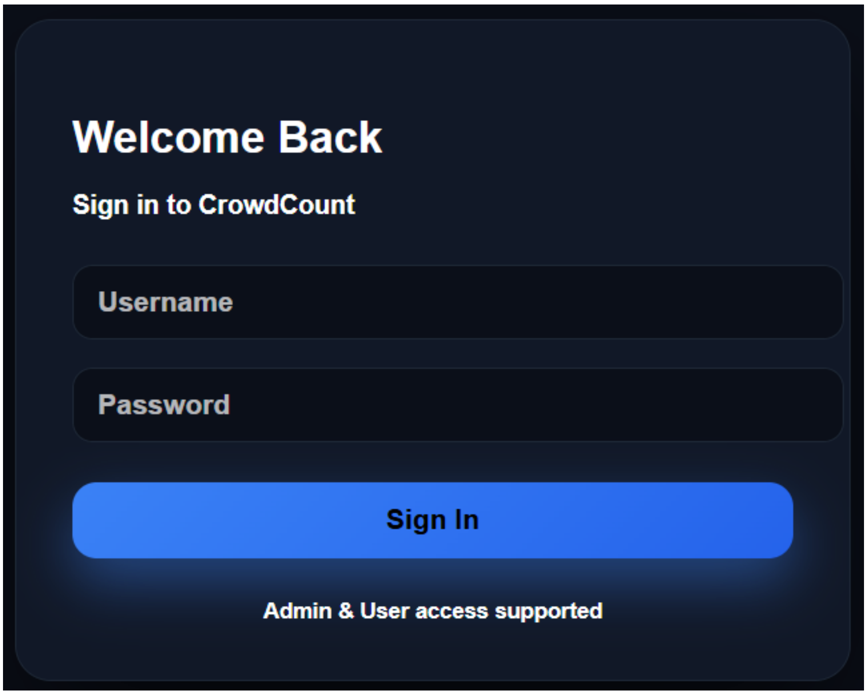
Video Analysis and zone creation



CrowdCount AI - Milestone 3

STATUS: DRAW MODE | 'q': Quit

Login authentication

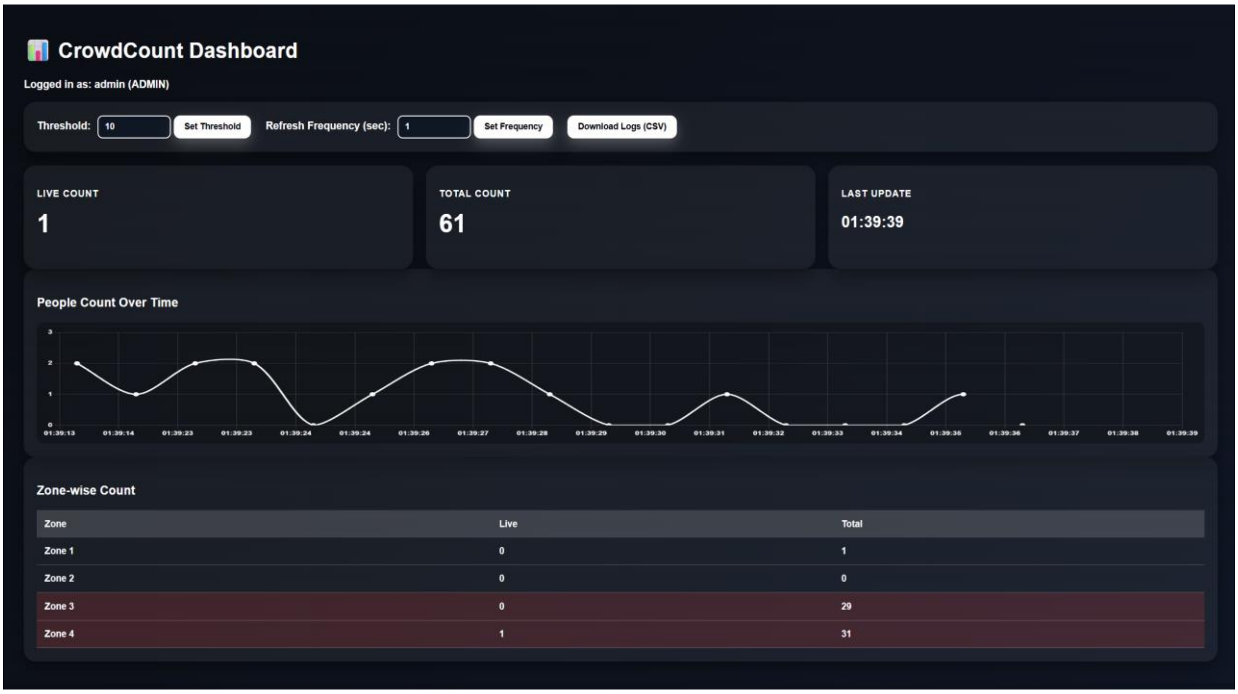


Welcome Back

Sign in to CrowdCount

Admin & User access supported

Main Dashboard



📊 CrowdCount Dashboard

Logged in as: admin (ADMIN)

LIVE COUNT

1

TOTAL COUNT

61

LAST UPDATE

01:39:39

People Count Over Time

Zone-wise Count

Zone	Live	Total
Zone 1	0	1
Zone 2	0	0
Zone 3	0	29
Zone 4	1	31

Conclusion

The CrowdCount Dashboard System successfully provides a real-time, role-based solution for monitoring crowd density across multiple zones. By integrating a Flask-based backend with a responsive web dashboard, the system enables live data visualization, threshold-based alerts, and secure access control for different user roles. The project demonstrates an effective approach to real-time data monitoring using modern web technologies and offers a scalable foundation for future enhancements such as AI-based crowd prediction, video analytics, and database integration.

Links

Github: https://github.com/andewadi/CrowdCount