**Infosys Springboard Virtual Internship**

CrowdCount - People Counting Using Video Analysis

**Name: Keerthana S**

**Date : 01-01-2026**

**1. Introduction**

CrowdCount is a smart, real-time crowd analysis and monitoring solution designed to detect, track, and manage human presence through live video streams. By leveraging advanced computer vision and deep learning algorithms, the system accurately identifies individuals, studies their movement behavior, and determines current occupancy levels within user-defined areas. It is well suited for use in corporate offices, educational campuses, commercial complexes, public infrastructure, and other high-footfall locations where crowd regulation and safety are critical.

The system allows administrators to define and manage monitoring zones directly on the video feed, enabling precise analysis for specific regions. For each configured zone, CrowdCount continuously updates occupancy counts and produces visual heatmaps that illustrate crowd density and movement patterns. These visual insights enable rapid identification of congestion and support timely decision-making to prevent overcrowding or safety risks.

CrowdCount includes a secure, role-based web dashboard that offers real-time monitoring, historical data analysis, and detailed reporting features. Standard users can view live occupancy and visual analytics, while administrators have additional control to set capacity limits, modify zones, and export reports for operational assessment and regulatory compliance. Through the integration of AI-powered detection, live visualization, data storage, and controlled user access, CrowdCount provides a scalable and dependable platform for intelligent crowd management and informed decision-making.

**2. Key Features Implemented**

* An AI-driven deep learning model is used for real-time detection and continuous tracking of individuals across successive video frames.
* The system supports zone-oriented crowd surveillance, enabling administrators to define, modify, remove, and store multiple monitoring zones directly on the live video interface.
* CrowdCount performs continuous occupancy calculation, providing both overall people counts and individual zone-wise population data in real time.
* Visual heatmaps are generated dynamically to illustrate crowd concentration levels and movement activity within monitored regions.
* Secure role-based authentication is implemented to differentiate access privileges between standard users and administrators.
* Zone configuration and management are restricted to administrators through password-protected controls.
* Administrators can set and manage occupancy capacity limits for each monitoring zone.
* The system automatically flags alerts whenever the number of people in a zone exceeds the predefined capacity.
* A live video streaming interface delivers real-time camera feeds with heatmap overlays to the web dashboard.
* An interactive web-based dashboard presents live occupancy counts, zone-specific metrics, charts, and trend visualizations.
* Time-stamped occupancy data is stored in a database to support long-term monitoring and analysis.
* The analytics and reporting component provides insights through statistical evaluations and crowd trend analysis.
* Reports can be exported in CSV and PDF formats for documentation, auditing, and offline review.
* The platform follows a scalable, modular system design, separating video capture, AI inference, zone processing, APIs, and frontend modules to simplify maintenance and enable future enhancements.

**3. System Workflow**

1. Camera Setup  
The system begins by configuring the video input source, which may be a webcam, network camera, or recorded video file. Video frames are captured continuously to support real-time analysis.

2. Frame Capture and Preparation  
Each incoming frame is resized and preprocessed to optimize computational efficiency and maintain smooth processing performance.

3. AI-Driven Human Detection and Tracking  
The processed frames are analyzed by an AI module that uses a deep learning model to identify people. A tracking mechanism assigns persistent identifiers to each individual, ensuring consistent tracking across consecutive frames.

4. Zone Definition and Validation  
Previously defined monitoring zones are retrieved from storage. The system determines whether each tracked person’s center point falls within any of the configured zone boundaries.

5. Live Occupancy Computation  
The number of distinct individuals within each zone is calculated, and an overall crowd count is updated in real time.

6. Dynamic Heatmap Creation  
The movement paths of tracked individuals are accumulated over time to produce a heatmap that highlights crowd density and activity levels across the observed area.

7. Real-time Visual Overlay  
Zone outlines, individual tracking markers, occupancy statistics, and heatmap layers are superimposed on the live video stream to provide instant visual insights.

8. Backend API Updates  
Current occupancy values, zone-level metrics, and heatmap information are continuously synchronized with the backend through API endpoints.

9. Dashboard Display  
The web interface retrieves live data from the APIs and presents real-time statistics, visual charts, alerts, and the video feed to authenticated users.

10. Capacity Monitoring and Alerting  
The system constantly compares zone occupancy against predefined limits and generates on-screen alerts whenever capacity thresholds are breached.

11. Data Persistence  
Crowd and occupancy information is saved at regular intervals in a database along with timestamps, enabling future review and analysis.

12. Analytics and Report Export  
Authorized administrators can access analytical views to examine crowd trends and generate downloadable CSV or PDF reports for operational evaluation and regulatory compliance.

**4. Code Implementation**

**main.py**

def check\_admin\_access():

    """Check if user has admin access for zone management"""

    global ALLOW\_ZONE\_MANAGEMENT

    try:

        password = getpass.getpass("Admin Password (or press Enter): ")

    except:

        # Fallback if getpass doesn't work

        password = input("Admin Password (or press Enter): ")

    print("-"\*60)

    if password == "":

        ALLOW\_ZONE\_MANAGEMENT = False

    elif password == ADMIN\_PASSWORD:

        ALLOW\_ZONE\_MANAGEMENT = True

    else:

        ALLOW\_ZONE\_MANAGEMENT = False

    print("="\*60 + "\n")

    time.sleep(2)

def start\_api():

    """Start the FastAPI server"""

    import uvicorn

    uvicorn.run(api\_server.app, host="0.0.0.0", port=8000, log\_level="error")

# --- Simple Rectangular Zone Drawing ---

drawing = False

start\_x = start\_y = curr\_x = curr\_y = None

mode = None

selected\_zone\_id = None

new\_zone\_name = None

paused = False

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

    global drawing, start\_x, start\_y, curr\_x, curr\_y

    global mode, selected\_zone\_id, new\_zone\_name

    if event == cv2.EVENT\_LBUTTONDOWN:

        if mode in ["edit","delete"]:

            for z in zn.zones:

                if zn.is\_point\_inside\_zone(x, y, z):

                    selected\_zone\_id = z["id"]

                    if mode == "delete":

                        zn.delete\_zone\_by\_id(selected\_zone\_id)

                        zn.save\_zones()

                        print(f"Deleted zone {z['name']}")

                        mode = None

                        return

                    else:

                        drawing = True

                        start\_x, start\_y = x, y

                        return

        if mode == "new":

            drawing = True

            start\_x, start\_y = x, y

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

        curr\_x, curr\_y = x, y

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

        drawing = False

        curr\_x, curr\_y = x, y

        x1, y1 = min(start\_x, curr\_x), min(start\_y, curr\_y)

        x2, y2 = max(start\_x, curr\_x), max(start\_y, curr\_y)

        pts = [[x1,y1],[x2,y1],[x2,y2],[x1,y2]]

zn.load\_zones()

cam.start\_camera(0)

**tracking.py**

from ultralytics import YOLO

import supervision as sv

import os

# Get the correct path to the model

model\_path = os.path.join(os.path.dirname(\_\_file\_\_), "..", "models", "ai\_models", "yolov8s.pt")

model = YOLO(model\_path)

tracker = sv.ByteTrack()

def track\_people(frame):

    results = model(frame, imgsz=640, conf=0.5, verbose=False)[0]

    det = sv.Detections.from\_ultralytics(results)

    det = det[det.class\_id == 0]  # Only persons

    tracked = tracker.update\_with\_detections(det)

    people = []

    for xyxy, tid in zip(tracked.xyxy, tracked.tracker\_id):

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

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

        people.append({

            "id": int(tid),

            "bbox": (x1, y1, x2, y2),

            "centroid": (cx, cy)

        })

    return people

**Dashboard.html**

<body>

    <div class="dashboard-container">

        <header>

            <div>

                <span class="pulse"></span>

                <span class="fw-bold" style="letter-spacing: 0.5px;">Croud<span style="color:var(--accent)">Count</span>

                    - Infosys</span>

            </div>

            <div class="d-flex align-items-center gap-4">

                <div class="text-end">

                    <div class="label-upper">Live Occupancy</div>

                    <div class="h4 mb-0 text-info-custom fw-bold" id="totalCount">0</div>

                </div>

                <div class="text-end border-start border-secondary ps-4">

                    <div class="label-upper">Sync Time</div>

                    <div class="small text-muted" id="lastUpdated">--:--:--</div>

                </div>

                <div class="user-badge" id="userBadge">

                    <span id="userRole"></span>: <span id="userName"></span>

                </div>

                <button class="btn btn-primary btn-sm admin-only-btn" onclick="exportCSV()"

                    style="display: none;">Export CSV</button>

                <button class="btn btn-primary btn-sm admin-only-btn" onclick="exportPDF()"

                    style="display: none;">Export PDF</button>

                <button class="btn btn-danger btn-sm" onclick="logout()">Logout</button>

            </div>

        </header>

        <aside class="sidebar">

            <h6 class="label-upper mb-3" style="color: var(--accent)">● Area Breakdown</h6>

            <div id="zoneContainer" class="zone-list-scroll"></div>

            <div id="alertBox" class="mt-3 p-2 d-none rounded text-center small fw-bold"

                style="background: var(--danger); color: white; box-shadow: 0 0 20px rgba(255, 0, 85, 0.3);">

                ⚠️ CRITICAL CAPACITY REACHED

            </div>

            <!-- Admin Panel -->

            <div id="adminPanel" class="admin-panel" style="display: none;">

                <h6 class="label-upper mb-2" style="color: var(--secondary-accent)">⚙️ Admin Controls</h6>

                <div id="thresholdControls"></div>

            </div>

        </aside>

        <main class="main-content">

            <h6 class="label-upper mb-3">Live Video Feed with Heatmap</h6>

            <div class="video-container">

                <img id="videoFeed" src="http://localhost:8000/video\_feed" alt="Live Feed">

            </div>

        </main>

        <div class="bottom-shelf">

            <div class="metric-card">

                <h6 class="label-upper mb-2">📊 Activity Heat Profile</h6>

                <div style="height: 130px;">

                    <canvas id="heatIntensityChart"></canvas>

                </div>

            </div>

            <div class="metric-card">

                <h6 class="label-upper mb-2">👥 Population Trend</h6>

                <div style="height: 130px;">

                    <canvas id="populationChart"></canvas>

                </div>

            </div>

            <div class="metric-card">

                <h6 class="label-upper mb-2">📈 Zone Comparison</h6>

                <div style="height: 130px;">

                    <canvas id="zoneComparisonChart"></canvas>

                </div>

            </div>

        </div>

    </div>

    <script>

        const API\_BASE = 'http://localhost:8000';

        const MAX\_POINTS = 30;

        let heatChart, populationChart, zoneComparisonChart;

        let labels = [];

        let thresholds = {};

        let userRole = '';

        let token = '';

        let populationHistory = [];

        let populationTimestamps = [];

        // Check authentication

        function checkAuth() {

            token = localStorage.getItem('token');

            const username = localStorage.getItem('username');

            userRole = localStorage.getItem('role');

            if (!token) {

                window.location.href = 'login.html';

                return false;

            }

            // Display user info

            document.getElementById('userName').textContent = username;

            document.getElementById('userRole').textContent = userRole.toUpperCase();

            const userBadge = document.getElementById('userBadge');

            if (userRole === 'admin') {

                userBadge.classList.add('admin-badge');

                document.getElementById('adminPanel').style.display = 'block';

                // Show export buttons for admin only

                document.querySelectorAll('.admin-only-btn').forEach(btn => {

                    btn.style.display = 'inline-block';

                });

            }

            return true;

        }

        function logout() {

            localStorage.clear();

            window.location.href = 'login.html?logout=true';

        }

        async function apiCall(endpoint, options = {}) {

            const headers = {

                'Authorization': `Bearer ${token}`,

                'Content-Type': 'application/json',

                ...options.headers

            };

            const response = await fetch(`${API\_BASE}${endpoint}`, {

                ...options,

                headers

            });

            if (response.status === 401) {

                logout();

                return null;

            }

            return response;

        }

        async function exportCSV() {

            try {

                const response = await apiCall('/export\_csv');

                if (response && response.ok) {

                    const blob = await response.blob();

                    const url = window.URL.createObjectURL(blob);

                    const a = document.createElement('a');

                    a.href = url;

                    a.download = `crowd\_report\_${new Date().getTime()}.csv`;

                    document.body.appendChild(a);

                    a.click();

                    window.URL.revokeObjectURL(url);

                    document.body.removeChild(a);

                }

            } catch (error) {

                console.error('CSV export failed:', error);

                alert('Failed to export CSV');

            }

        }

        async function exportPDF() {

            try {

                const response = await apiCall('/export\_pdf');

                if (response && response.ok) {

                    const blob = await response.blob();

                    const url = window.URL.createObjectURL(blob);

                    const a = document.createElement('a');

                    a.href = url;

                    a.download = `crowd\_report\_${new Date().getTime()}.pdf`;

                    document.body.appendChild(a);

                    a.click();

                    window.URL.revokeObjectURL(url);

                    document.body.removeChild(a);

                }

            } catch (error) {

                console.error('PDF export failed:', error);

                alert('Failed to export PDF');

            }

        }

        async function loadThresholds() {

            if (userRole !== 'admin') return;

            try {

                const response = await apiCall('/thresholds');

                if (response && response.ok) {

                    thresholds = await response.json();

                    console.log('Loaded thresholds:', thresholds);

                    // If no thresholds exist, create default ones from current zones

                    if (Object.keys(thresholds).length === 0) {

                        console.log('No thresholds found, will create from zones');

                        // Get current zones from the latest data

                        const countResponse = await fetch(`${API\_BASE}/get\_count`);

                        if (countResponse.ok) {

                            const countData = await countResponse.json();

                            // Create default thresholds for each zone

                            for (const zoneName of Object.keys(countData.zones)) {

                                thresholds[zoneName] = { max\_capacity: 30, alert\_enabled: true };

                                // Save to backend

                                await apiCall(`/set\_threshold?zone\_name=${encodeURIComponent(zoneName)}&max\_capacity=30`, {

                                    method: 'POST'

                                });

                            }

                            console.log('Created default thresholds:', thresholds);

                        }

                    }

                    renderThresholdControls();

                }

            } catch (error) {

                console.error('Failed to load thresholds:', error);

            }

        }

        function renderThresholdControls() {

            const container = document.getElementById('thresholdControls');

            if (!container) {

                console.error('thresholdControls container not found!');

                return;

            }

            container.innerHTML = '';

            console.log('Rendering thresholds:', thresholds);

            if (Object.keys(thresholds).length === 0) {

                container.innerHTML = '<div style="color: #64748b; font-size: 0.85rem; padding: 10px;">No zones configured yet. Create zones in the video window.</div>';

                return;

            }

            for (const [zoneName, data] of Object.entries(thresholds)) {

                const item = document.createElement('div');

                item.className = 'threshold-item';

                item.setAttribute('data-zone', zoneName);

                item.innerHTML = `

            <span style="font-size: 0.85rem;">${zoneName}</span>

            <div>

                <input type="number" value="${data.max\_capacity}"

                       onchange="updateThreshold('${zoneName}', this.value)"

                       min="1" max="1000">

            </div>

        `;

                container.appendChild(item);

            }

            console.log(`Rendered ${Object.keys(thresholds).length} threshold controls`);

        }

        async function updateThreshold(zoneName, value) {

            try {

                const response = await apiCall(`/set\_threshold?zone\_name=${zoneName}&max\_capacity=${value}`, {

                    method: 'POST'

                });

                if (response && response.ok) {

                    thresholds[zoneName].max\_capacity = parseInt(value);

                    console.log(`Threshold updated for ${zoneName}: ${value}`);

                }

            } catch (error) {

                console.error('Failed to update threshold:', error);

            }

        }

        function init() {

            if (!checkAuth()) return;

        init();

    </script>

</body>

</html>

**5. Code Explanation**

1. main.py – Core Execution & System Coordination

* Acts as the central controller for the entire CrowdCount platform
* Initializes video sources and loads stored zone configurations
* Starts the AI inference loop for real-time person detection and tracking
* Handles administrator actions for zone creation, modification, and deletion
* Captures frames, calculates FPS, and overlays bounding boxes, zones, counts, and heatmaps
* Updates live occupancy statistics and shares them with backend APIs
* Periodically stores occupancy data in the database
* Runs the FastAPI server in a parallel thread to support live streaming and dashboard access
* Integrates all backend modules for continuous real-time operation

2. camera\_feed.py – Video Input Management

* Manages video input from webcams, IP cameras, or prerecorded files
* Configures camera properties such as resolution and frame rate
* Continuously captures frames for downstream processing
* Safely releases camera resources during system shutdown
* Isolates hardware-specific logic to keep the system device-independent

3. tracking.py – AI Detection & Tracking Engine

* Loads a YOLOv8 deep learning model for human detection
* Filters detections to track only person-class objects
* Uses ByteTrack to assign consistent IDs across frames
* Extracts bounding box coordinates and centroid positions
* Outputs structured tracking data for zone and heatmap processing
* Serves as the intelligence core of the system

4. zones.py – Zone Management & Heatmap Processing

* Loads and saves zone configurations using a shared JSON file
* Supports creation, modification, and deletion of polygon-based zones
* Determines whether tracked individuals fall within defined zones
* Counts unique individuals per zone using tracking IDs
* Generates dynamic heatmaps based on movement intensity
* Maintains historical heatmap data for analytics
* Renders zone boundaries, labels, heatmaps, and occupancy text on video frames

5. database.py – Data Storage & Threshold Handling

* Manages SQLite database connections and schema
* Creates and maintains tables for occupancy logs and zone thresholds
* Stores time-stamped crowd data for historical analysis
* Retrieves recent records for analytics and reporting
* Manages per-zone capacity limits and alert settings
* Ensures reliable session handling and error management

6. auth\_routes.py – Authentication & Authorization APIs

* Implements secure user authentication using JWT tokens
* Generates access tokens containing role-based information
* Validates tokens for protected API endpoints
* Returns authenticated user details to the frontend
* Enforces role-based access for administrators and standard users

7. admin\_routes.py – Administrative Backend APIs

* Provides admin-only endpoints for system management
* Handles user creation, modification, deletion, and viewing
* Manages camera source configurations and metadata
* Supports zone management through secured APIs
* Retrieves real-time zone-wise occupancy data
* Allows administrators to configure capacity thresholds
* Restricts access to authorized administrative users only

8. analytics\_routes.py – Analytics & Reporting

* Fetches historical occupancy data from the database
* Computes statistical metrics such as average, minimum, and maximum counts
* Exports historical data in CSV format
* Generates professional PDF reports with zone and activity details
* Supports auditing, compliance, and operational analysis

9. public\_routes.py – Public APIs & Live Streaming

* Exposes system health and status endpoints
* Provides live total and zone-wise occupancy counts
* Streams real-time video with heatmap overlays using MJPEG
* Maintains shared global state for live frames and statistics
* Enables dashboard updates without exposing sensitive controls

10. dashboard.html – Authenticated Web Dashboard

* Displays live video feed with heatmap and zone overlays
* Shows real-time total and zone-wise occupancy counts
* Visualizes data using charts for trends and comparisons
* Provides threshold configuration controls for administrators
* Triggers alerts when occupancy limits are exceeded
* Supports CSV and PDF report exports
* Adjusts UI elements based on user roles

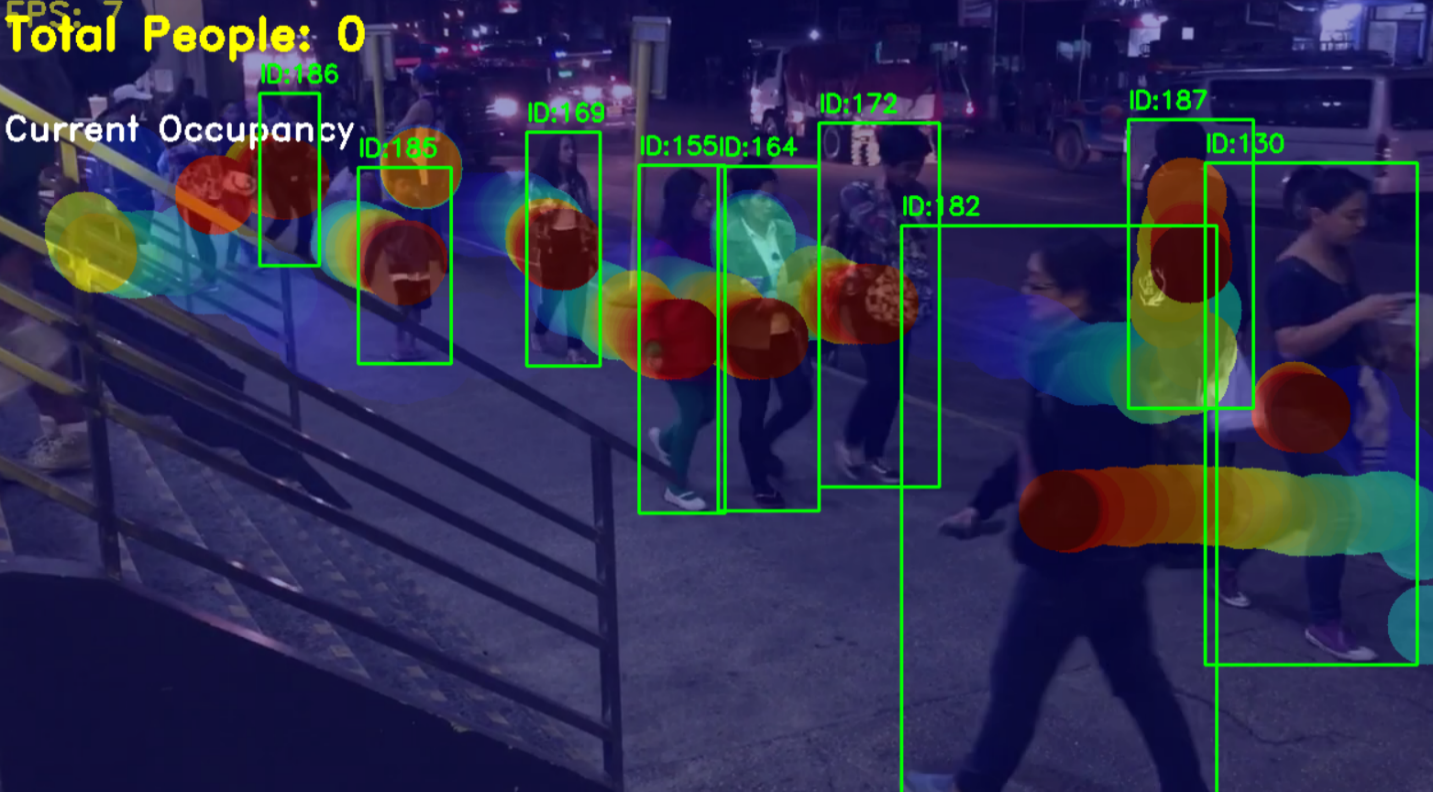
11. login.html – Secure Login Interface

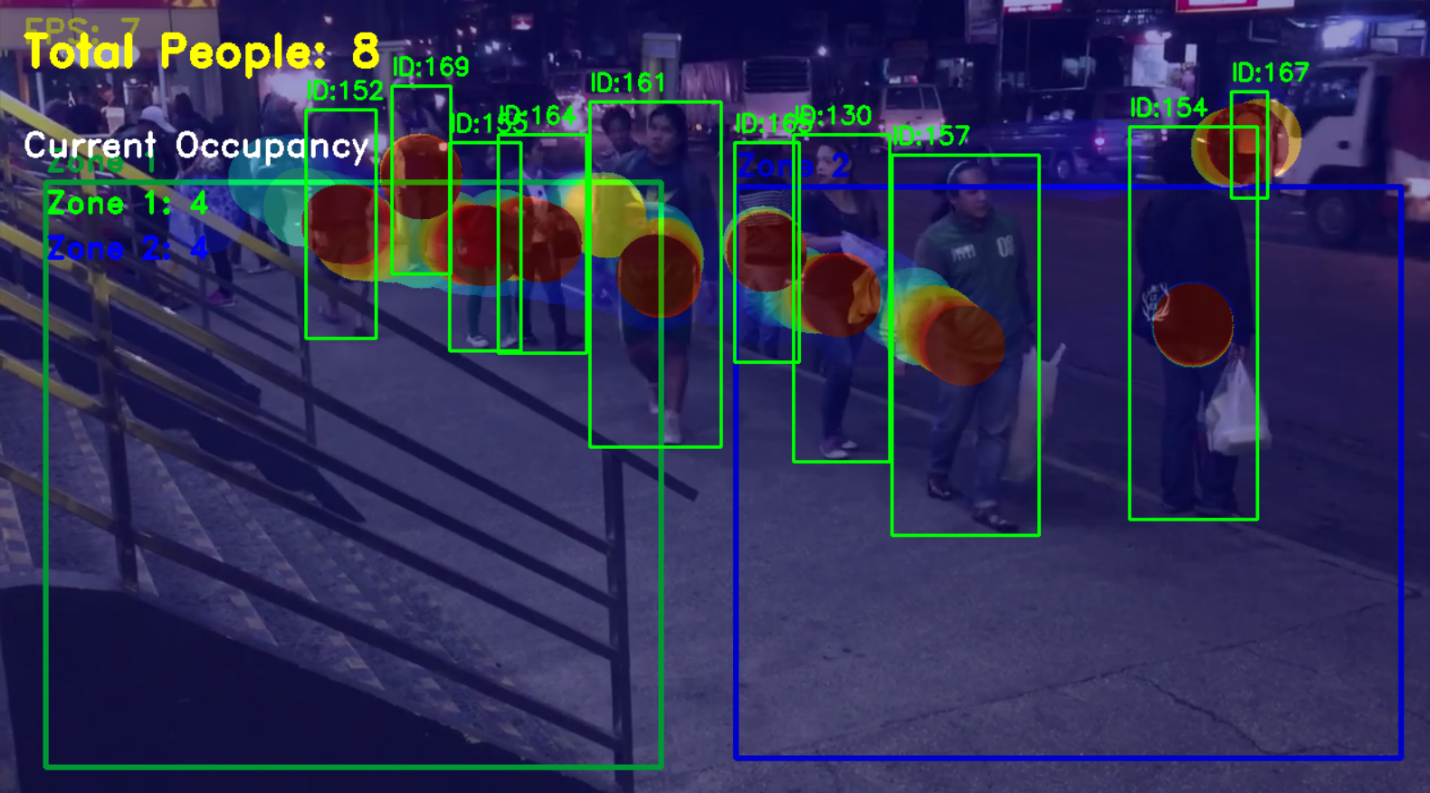
* Collects user credentials securely
* Communicates with backend authentication services
* Stores JWT tokens and role data in browser storage
* Redirects users based on authentication status
* Displays loading indicators and error messages

12. index.html – Public / Lightweight Dashboard

* Displays live occupancy counts and trend visuals
* Shows zone-wise information without admin controls
* Requires no authentication
* Suitable for demonstrations and basic monitoring use cases

**6. Output**

****



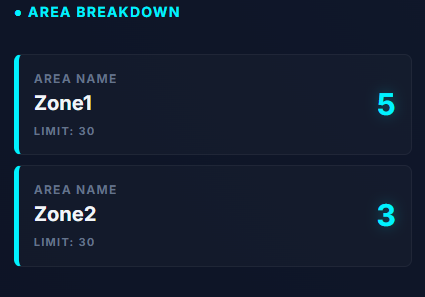
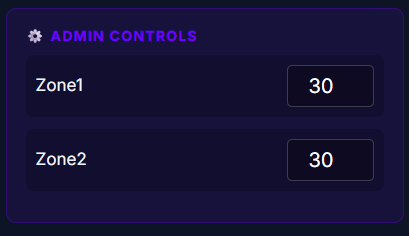


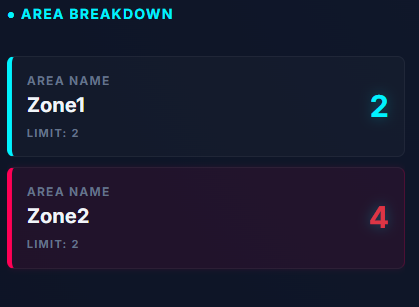
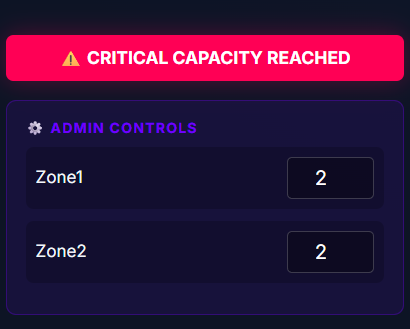
















**7. Conclusion**

The CrowdCount platform effectively showcases the seamless integration of artificial intelligence, computer vision techniques, and modern web frameworks to create a reliable real-time crowd analysis and monitoring system. Through the use of AI-powered human detection and tracking combined with zone-level occupancy evaluation, the system delivers precise and meaningful insights into crowd movement patterns and space usage.

The solution incorporates flexible zone configuration, live visual representation through dashboards and heatmaps, secure role-based authentication, and continuous data storage for long-term analysis. Safety and operational control are strengthened by administrative capabilities such as occupancy limit configuration and automated alert mechanisms, enabling timely and informed responses. Additionally, the ability to generate analytical reports in both CSV and PDF formats supports auditing, documentation, and compliance needs.

In summary, CrowdCount offers a modular, scalable, and deployment-ready solution suitable for environments including corporate workplaces, public infrastructure, and high-density areas. Its organized system design and extensible framework allow for future improvements such as enhanced analytics, multi-camera integration, and intelligent alert systems, positioning it as a solid base for next-generation smart crowd management applications.