Zone Division Endpoint

Purpose:
Accepts an uploaded location file or data, processes the mapped area into spatial polygon zones, and returns zone data for use by other system modules.

Recommended HTTP Method:
POST (used when uploading files or sending data to be processed).

Route Example:
`POST /api/zones/divide`

Authorization:

- For most internal, sensitive, or potentially misused endpoints (such as those modifying/creating area data), require an API key passed in an Authorization header. This prevents anyone from abusing or flooding your service.
- Use simple API keys for internal use or when no user-level identity is needed. For public APIs or user-level security, consider OAuth/JWT, but for your MVP/project, an API key is usually enough.

Inputs:

- Multipart form upload:
  - `file`: (optional) .geojson, .json, or supported format representing the location's coordinates or shape
  - or direct JSON body containing location/polygon coordinates for manual entry

Note on Input Coordinates:

The Zone Division API expects spatial polygon input data with accurate coordinates.

Coordinates must be in [longitude, latitude] format, as per GeoJSON standards.

For polygons, the coordinates array should represent closed shapes (first and last points must be identical).

This is crucial for spatial polygon division and area calculations.

1. Input Format Must Include Lat/Lon Pairs

Example polygon in JSON input or GeoJSON format:

json

```
{
  "type": "Feature",
  "geometry": {
    "type": "Polygon",
    "coordinates": [
      [
        [lon1, lat1],
        [lon2, lat2],
        [lon3, lat3],
        [lon4, lat4],
        [lon1, lat1]  // Polygon closed by repeating first coordinate
      ]
    ]
  },
  "properties": {
    "name": "User Selected Area"
  }
}
```

Example Request Using curl:

```
text

curl -X POST http://yourapi.com/api/zones/divide \
  -H "Authorization: Bearer <YOUR_API_KEY>" \
  -F "file=@/path/to/location.geojson"
```

Frontend:
- Provides interactive map for users to draw/select areas or upload location files
- Sends processed polygon data to backend API

Success Response Example:

```json
{
  "status": "success",
  "zones": [
    {
      "zone_id": "zone1",
      "polygon": [[lat1, lon1], [lat2, lon2], ...],
      "area_sqft": 5320.5
      "camera_id": "camera_01"
    },
    {
      "zone_id": "zone2",
      "polygon": [[lat1, lon1], [lat2, lon2], ...],
      "area_sqft": 4883.13
      "camera_id": "camera_01"
    }
  ]
}
```

- *Output should be a list/array of zones with unique IDs and their polygon coordinates. Add extra metadata (name, camera_id, etc.) as your system evolves.*

Camera Video Upload Endpoint

Purpose:

Allow users to upload video footage for a specific camera associated with a zone. This video will then be consumable by other components (crowd density, anomaly detection, missing person search) without ambiguity in mapping camera, video, and zone data.

HTTP Method:

POST

Route:

`/api/cameras/upload-video`

Authorization:

API key or JWT token required.

Inputs:

- camera_id (string, required)
  Identifier of the camera (from zone division metadata, e.g., "camera_01")
- zone_id (string, optional but recommended)
  Identifier of zone this camera belongs to, ensuring clear video-zone mapping.
- file (multipart form-data, required)
  The MP4 video file uploaded via frontend by user.

Outputs:

```
json

{
  "status": "success",
  "message": "Video uploaded successfully",
  "camera_id": "camera_01",
  "zone_id": "Z1",
  "video_url":
"https://storage.example.com/videos/camera_01-20251116.mp4",
  "upload_timestamp": "2025-11-16T01:40:00Z"
}
```

- `video_url` is the accessible location of the uploaded video, which downstream components will use as input.

Integration with Downstream Components:

- Use the `video_url` output as input to Crowd Density Prediction, Anomaly Detection, and Missing Person Search APIs where video or stream input is required.
- The `camera_id` and `zone_id` fields in the output provide unambiguous mapping for cameras, zones, and videos throughout your system.
- This structured approach ensures consistent data flow and minimal confusion.

Example cURL request:

bash

```
curl -X POST http://localhost:8000/api/cameras/upload-video \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -F "camera_id=camera_01" \
  -F "zone_id=Z1" \
  -F "file=@/path/to/video.mp4"
```

Crowd Density Prediction Endpoint

Purpose:
Processes video footage/stream for a mapped zone and estimates people density, reporting results in persons per unit area.

Recommended HTTP Method:
POST (used for uploading video files or sending data to be analyzed).

Route Example:
`POST /api/zones/{zone_id}/density`

Authorization:

- Also recommended to use API keys here, especially since videos are sensitive and you want to prevent unauthorized or automated abuse.

Inputs:

- Path parameter: `zone_id` (from the zone division output)
- Multipart form upload or JSON:
    - `file`: .mp4 video footage or current frame/image
    - or stream URL (for future support)

Example Request Using curl:

```
text

curl -X POST http://yourapi.com/api/zones/zone1/density \
  -H "Authorization: Bearer <YOUR_API_KEY>" \
  -F "file=@/path/to/video.mp4"
```

or with JSON (if you support video streams later):

```
json

{
  "video_url": "http://camera-feed/link/zone1"
}
```

Success Response Example:

Outputs:

```
json

{
  "zone_id": "Z1",
  "people_count": 100,
  "density": 0.047,       // people per sqft
  "area_sqft": 5320.5,
  "entry_rate": 5,        // number of people entering the zone
per minute
  "exit_rate": 3,         // number of people exiting the zone
per minute
  "camera_id": "camera_01",
  "timestamp": "2025-11-16T06:00:00Z"
}
```

- Here, people_count, entry_rate, and exit_rate are output metrics produced by crowd density analysis.
- *Include the raw count, physical zone size (from the first endpoint), density value, and optionally a label/category for UI display.*

Anomaly Detection Endpoint Documentation

POST /api/anomaly/detect — Inputs

- request_id (string, required)
  Unique identifier for the detection request.
- timestamp (string, optional)
  Time at which the data/frame was captured.
- source (string, optional)
  Camera or sensor identifier, e.g., `"camera_01"`.
- location (array, optional)
  Geographical coordinates `[latitude, longitude]`, typically from Zone Division output.
- payload (object, required) containing:
  - mode (string, required)
    Either `"frame"` or `"features"`, denoting the type of input.
  - If `mode` is `"frame"`:
    - frame (object, required) containing either:
      - `image_base64` (string): Base64-encoded image string OR
      - `image_url` (string): Publicly accessible URL to an image frame
  - If `mode` is `"features"`:
    - features (object, required) containing:
      - `people_count` (integer, required)
      - `entry_rate` (integer, optional): Number of people entering the zone per minute
      - `exit_rate` (integer, optional): Number of people exiting the zone per minute

---

POST /api/anomaly/detect — Outputs

- Common fields:
  - `request_id` (string)
  - `zone_id` (string)
  - `is_anomaly` (boolean)
  - `anomaly_score` (float between 0 and 1)
  - `anomaly_type` (string or null)
  - `affected_zones` (array of strings): Zones impacted by the anomaly
  - `explanation` (string): Natural language description suitable for user display, e.g., "Fire detected in zone Z1 near camera_01."
  - `meta` (object): Model metadata (version, inference time, thresholds)
- New outputs for downstream components:
  - `alert` (object): Concise alert summary for routing and early warning systems
  - `json`

```
{
  "anomaly_type": "fire",
  "zone_id": "Z1",
  "severity": "high",            // Derived from anomaly_score &
thresholds
  "timestamp": "2025-11-16T06:00:00Z",
  "location": [12.9721, 77.5946] // Latitude, Longitude coordinates of
the anomaly
}
```

-
- This `alert` can be passed to the Shortest Path component for emergency routing.
- The same `alert` is used by the 15 Minutes Prior Prediction system for proactive warnings.

Example cURL Usage

Frame Mode:

bash

```
curl -X POST http://localhost:8000/api/anomaly/detect \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -d '{
    "request_id": "req_001",
    "timestamp": "2025-02-10T10:23:00Z",
    "source": "camera_04",
    "location": [12.9721, 77.5946],
    "payload": {
      "mode": "frame",
      "frame": {
        "image_base64": "<BASE64_ENCODED_IMAGE_STRING>"
      }
    }
  }'
```

Features Mode:

bash

```
curl -X POST http://localhost:8000/api/anomaly/detect \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -d '{
    "request_id": "req_002",
    "timestamp": "2025-02-10T10:23:00Z",
    "source": "camera_07",
    "location": [12.9721, 77.5946],
    "payload": {
      "mode": "features",
      "features": {
        "people_count": 35,
        "entry_rate": 4,
        "exit_rate": 1
      }
    }
  }'
```

15-Minutes Crowd Surge Prediction Endpoint

Purpose:

- Predicts crowd behavior for the next 15 minutes in each zone based on current crowd density data.
- Reacts dynamically to alerts from the Anomaly Detection component by altering the prediction output, indicating crowd movement patterns specific to emergency scenarios.

HTTP Method:

- POST `/api/crowd/predict`

Authorization:

- Token-based authorization (API key/JWT), required to access prediction services.

Inputs:

- zone_data (array of objects, required):
  List of current crowd densities for each zone, derived from the Crowding Density Prediction component:
- `json`

```
[
  {
    "zone_id": "Z1",
    "people_count": 100,
    "density": 0.047,
    "area_sqft": 5320.5
  },
  {
    "zone_id": "Z2",
    "people_count": 80,
    "density": 0.045,
    "area_sqft": 5320.5
  },
  ...
]
```

-
- alerts (array of objects, optional):
  Recent critical alerts from the Anomaly Detection component, including zone of occurrence.
  Example:
- `json`

```
[
  {
    "zone_id": "Z1",
    "anomaly_type": "fire",
    "severity": "high",
    "timestamp": "2025-11-16T06:00:00Z",
    "location": [12.9721, 77.5946]
  }
]
```

-
- If present, the model alters its prediction based on the anomaly to reflect possible crowd surges or dispersals.

Responses:

Normal Prediction (No Anomaly Active):

json

```
{
  "prediction": "Crowd in Zone Z1 expected to increase slightly, stabilizing at moderate levels.",
  "confidence": 72  // percentage
}
```

If an anomaly has occurred (from alert):

json

```
{
  "prediction": "Zone Z1 will experience a rapid crowd dispersal, reducing pressure. Meanwhile, Zone Z2 will likely see a surge at exit gate.",
  "confidence": 85  // percentage
}
```

Note: The prediction text dynamically changes if an alert is received, reflecting anticipated crowd movement under emergency conditions.

Example cURL Command:

bash

```
curl -X POST http://localhost:8000/api/crowd/predict \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -d '{
```

```
  "zone_data": [
    {
      "zone_id": "Z1",
      "people_count": 100,
      "density": 0.047,
      "area_sqft": 5320.5
    },
    {
      "zone_id": "Z2",
      "people_count": 80,
      "density": 0.045,
      "area_sqft": 5320.5
    }
  ],
  "alerts": [
    {
      "zone_id": "Z1",
      "anomaly_type": "fire",
      "severity": "high",
      "timestamp": "2025-11-16T06:00:00Z",
      "location": [12.9721, 77.5946]
    }
  ]
  }'
```

Outputs:

- Prediction Text: A human-readable forecast about crowd trends, intended for user display.
- Confidence Percentage: Likelihood (percent) of the predicted event's accuracy.
- Alert Impact: If a critical alert exists, the prediction will include tailored messages indicating crowd dispersal or surge zones.

Summary:

This API acts as a real-time predictive tool, dynamically altering predictions based on alerts, and providing essential data to emergency and planning systems.

Shortest Path Finder Endpoint

Purpose:

Calculates the shortest/optimal path for an emergency responder from their current location to the target anomaly zone, considering current crowd density per zone to avoid congestion.

HTTP Method:

POST

Route:

`/api/path/find`

Authorization:

Requires API key or JWT token for access.

Inputs:

- current_location (array, required)
  Responder's current GPS coordinates: `[latitude, longitude]`.
- alert (object, required)
  Active anomaly alert (from anomaly detection component) containing:
- `json`

```
{
  "zone_id": "Z1",
  "anomaly_type": "fire",
  "severity": "high",
  "timestamp": "2025-11-16T06:00:00Z",
  "location": [12.9721, 77.5946]
}
```

- crowd_density_data (array of objects, required)
  Current crowd densities per zone from Crowd Density Prediction component:
- `json`

```
[
  {
    "zone_id": "Z1",
    "people_count": 100,
    "density": 0.047,
    "area_sqft": 5320.5
  },
  {
    "zone_id": "Z2",
    "people_count": 80,
    "density": 0.045,
    "area_sqft": 4800
  }
]
```

Outputs:

- GeoJSON Feature object describing the shortest path polyline:

json

```
{
  "type": "Feature",
  "geometry": {
    "type": "LineString",
    "coordinates": [
      [77.5900, 12.9700],
      [77.5915, 12.9715],
      [77.5946, 12.9721]
    ]
  },
  "properties": {
    "distance_meters": 450,
    "estimated_time_seconds": 300
  }
}
```

- This output can be directly rendered on Leaflet/OpenStreetMap to show the route visually.

Example cURL Request:

bash

```bash
curl -X POST http://localhost:8000/api/path/find \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -d '{
    "current_location": [12.9710, 77.5900],
    "alert": {
      "zone_id": "Z1",
      "anomaly_type": "fire",
      "severity": "high",
      "timestamp": "2025-11-16T06:00:00Z",
      "location": [12.9721, 77.5946]
    },
    "crowd_density_data": [
      {
        "zone_id": "Z1",
        "people_count": 100,
        "density": 0.047,
        "area_sqft": 5320.5
      },
      {
        "zone_id": "Z2",
        "people_count": 80,
        "density": 0.045,
        "area_sqft": 4800
      }
    ]
  }'
```

Missing Person Search Component API Documentation

Purpose:

- Allow users to register a missing person case by uploading the person’s image, height, hair description, etc.
- Use multi-step filtering (clothes color, hair, height) on video frames to narrow down candidates.
- Employ face vector embeddings for matching candidates to the missing person.
- Notify the user when a match is found.

Endpoints

---

1. Register Missing Person Case

HTTP Method:
POST

Route:
`/api/missing/register`

Authorization:
API key required.

Inputs (multipart/form-data):

- `image` (file): Photo of the missing person.
- `height` (float): Height of the person in cm or feet (specify unit).
- `hair_description` (string): Short textual description (e.g., "curly brown hair").
- `clothes_color` (string): Predominant clothing color.
- `reporter_contact` (string): Optional, for notifications.

Output:

```
json

{

  "case_id": "case123",

  "status": "registered",

  "message": "Missing person case registered successfully."
}
```

---

2. Search Missing Person In Video Footage

HTTP Method:
POST

Route:
`/api/missing/search`

Authorization:
API key required.

Inputs:

- JSON payload or multipart upload (depending on implementation) with:
  - `case_id` (string): ID from register endpoint.
  - `video_url` (string): URL to video footage to search in.
  - `skip_frames` (integer, optional): Number of frames to skip between processing to speed up search.
  - Optional advanced filters override like clothing color, hair color, height (if user wants to narrow the search).

Output:

- If match found:

json

```
{
  "case_id": "case123",
  "matched": true,
  "matching_frames": [
    {
      "timestamp": "2025-11-16T10:00:00Z",
      "frame_url": "https://example.com/frame123.jpg",
      "confidence": 0.85
    },

    ...

  ],

  "message": "Matching person found in video frames. Notification sent."

}
```

- If no match:

```
json

{

  "case_id": "case123",

  "matched": false,

  "message": "No matching person found in the provided video."

}
```

Notes:

- The search uses a multi-step matching pipeline:
  1. Filter candidates using clothes color, hair description, height.
  2. Use vector embeddings of face images for final matching.
- Notifications sent to reporter when matches occur (mechanism out of scope).
- Designed as a standalone component independent of other crowd and anomaly modules but can be integrated if desired later.