WebSocket API for People Counter Application

Endpoint: `ws://<server_ip>:<port>` (Default: `ws://<server_ip>:8765`)

Overview

The WebSocket server provides real-time data streams for video frames, application statistics, event notifications, and allows clients to request saved face images. All messages are JSON formatted.

A. Server-to-Client Messages (Pushed Data)

These messages are broadcast by the server to all connected clients periodically or when events occur.

1. Frame Update

Purpose: Sends a compressed JPEG image of the latest processed video frame.

JSON Structure:

```json
{
    "type": "frame",
    "data": "<base64_encoded_jpeg_string>",
    "timestamp": "YYYY-MM-DD HH:MM:SS.mmm"
}
```

Fields:

- `type`: (string) Always "frame"
- `data`: (string) Base64 encoded string of the JPEG image. To display, prefix with ``
- `timestamp`: (string) Server timestamp when the frame message was created

Frequency: Approximately 10-30 times per second, depending on processing speed and `WEBSOCKET_STREAM_QUALITY`.

2. Statistics Update

Purpose: Sends current application statistics.

JSON Structure:

```json
{
    "type": "stats",
    "data": {
        "count_in": 0,
        "count_out": 0,
        "face_count": 0,
        "daily_event_count": 0,
        "event_active": false,
        "current_event_count_in": 0
    },
    "timestamp": "YYYY-MM-DD HH:MM:SS.mmm"
}
```

Fields:

- `type`: (string) Always "stats"
- `data`: (object) Contains the statistical values:
  - `count_in`: (integer) Total people counted entering since the last daily reset
  - `count_out`: (integer) Total people counted exiting since the last daily reset
  - `face_count`: (integer) Total unique faces detected and saved since the last daily reset
  - `daily_event_count`: (integer) Number of "events" (high traffic periods) detected today
  - `event_active`: (boolean) `true` if an event is currently active, `false` otherwise
  - `current_event_count_in`: (integer) Number of people counted IN during the current active event (if `event_active` is true)
- `timestamp`: (string) Server timestamp when the stats message was created

Frequency: Approximately twice per second.

3. Event Notification

Purpose: Notifies clients about the start or end of a significant "event" (e.g., a period of high foot traffic).

JSON Structure (Event Started):

json

```
{
    "type": "event",
    "data": {
        "status": "started",
        "start_time": "HH:MM:SS"
    },
    "timestamp": "YYYY-MM-DD HH:MM:SS.mmm"
}
```

JSON Structure (Event Ended):

json

```
{
    "type": "event",
    "data": {
        "status": "ended",
        "start_time": "HH:MM:SS",
        "end_time": "HH:MM:SS",
        "count_in": 0
    },
    "timestamp": "YYYY-MM-DD HH:MM:SS.mmm"
}
```

Fields:

- `type`: (string) Always "event"
- `data`: (object) Contains event details:
  - `status`: (string) Either "started" or "ended"
  - `start_time`: (string) Time (HH:MM:SS) when the event started
  - `end_time`: (string, only for "ended" status) Time (HH:MM:SS) when the event ended
  - `count_in`: (integer, only for "ended" status) Total people counted IN during this specific event
- `timestamp`: (string) Server timestamp when the event message was created

Frequency: Sent when an event starts or ends.

B. Client-to-Server Messages (Requests)

Clients can send these messages to request specific data from the server.

1. Request Single Face Image

Purpose: Request the image data for a specific detected face using its ID.

Client Request:

```json
{
    "type": "get_face",
    "face_id": 123
}
```

Fields:

- `type`: (string) Always "get_face"
- `face_id`: (integer) The unique ID of the face to retrieve. Face IDs are typically assigned sequentially or are track IDs from the YOLO model

Server Response (Success):

```json
{
    "type": "face_image",
    "face_id": 123,
    "image_data": "<base64_encoded_jpeg_string>",
    "filename": "face_123.jpg"
}
```

Response Fields:

- `image_data`: Base64 encoded string of the JPEG face image
- `filename`: Original filename of the saved face image on the server

Server Response (Error/Not Found):

```json
{
    "type": "face_image",
    "face_id": 123,
    "error": "Face ID not found"
}
```

Error Field:

- `error`: A message describing why the face image could not be retrieved (e.g., "Face ID not found", "Face image file not found")

2. Request All Recent Face Images

Purpose: Request a list of the most recently saved face images, with their data.

Client Request:

```json
{
    "type": "get_all_faces",
    "limit": 50
}
```

Fields:

- `type`: (string) Always "get_all_faces"
- `limit`: (integer, optional) The maximum number of recent face images to return. Defaults to 50 on the server if not provided

Server Response (Success):

```json
{
    "type": "all_faces",
    "faces": [
        {
            "face_id": 123,
            "filename": "face_123.jpg",
            "image_data": "<base64_encoded_jpeg_string>"
        },
        {
            "face_id": 122,
            "filename": "face_122.jpg",
            "image_data": "<base64_encoded_jpeg_string>"
        }
    ]
}
```

Response Fields:

- `faces`: (array) An array of face objects, sorted from most recent to oldest. Each object contains `face_id`, `filename`, and `image_data`

Server Response (Error):

```json
{
    "type": "all_faces",
    "error": "Face data not available on server"
}
```

Error Field:

- `error`: A message describing an error (e.g., "Face data not available on server")

C. General Error Message (Server-to-Client)

If the client sends an invalid JSON message or an unrecognized request type, the server may respond with a generic error.

JSON Structure:

```json
{
    "error": "Invalid JSON message"
}
```

Example Errors:

- "Invalid JSON message"
- "Internal server error processing request"
- "Invalid face_id or limit format"

Client Implementation Notes

Connection

Establish a WebSocket connection to the server's endpoint.

Message Parsing

All messages are JSON. Parse incoming messages accordingly.

Image Display

Base64 encoded image data can be directly used in an `<img>` tag's `src` attribute by prepending ``.

Error Handling

Implement logic to handle connection errors, timeouts, and error messages from the server.

Reconnection

Consider implementing a reconnection strategy if the WebSocket connection is lost.

Note: Remember to replace placeholders like `<server_ip>` with actual values when implementing your client application.