Real-time analysis (streaming)

Emotion-Logic's real-time API offers instant emotion detection for live interactions, making it ideal for voice-controlled devices, customer support, or any situation requiring immediate emotional understanding. With the real-time API, you can process streaming audio data and receive emotion detection results as events occur, enhancing responsiveness and user engagement.

Streaming (real-time) analysis is based on socket.io (Web Socket) and consists of several events that are sent from the client to the Docker container and vice versa.
Socket.io clients are supported by many programming languages.

Please refer to the full client implementation in the "stream-analysis-sample.js" file (NodeJS).

The analysis flow for a single call is as follows:

1. The client connects to the Docker container.
2. The client sends a "handshake" event containing audio metadata.
3. The Docker container sends a "handshake-done" event, indicating that it is ready to start receiving the audio stream, or provides an error indication related to the "handshake" event.
4. The client begins sending "audio-stream" events with audio buffers.
5. The Docker container sends an "audio-analysis" event whenever it completes a new analysis.
6. The client disconnects when the stream (call) is finished.

All code samples in this document are in NodeJS, but any socket.io client library should work for this purpose.

Connecting the analysis server

Connecting the analysis server is a standard client-side websockets connection

```
const io = require("socket.io-client");

function connect(url) {
    return new Promise((y,n) => {
        const socket = io.connect(url, {
            transports: ["websocket"]
        });

        socket.on("connect", () => {
            console.log("connected to analysis server");
        });

        socket.on("connect_error", (err) => {
            console.log("error connecting to analysis server");
        });
    });
}
```

Handshake Event

Sent by: client

Event payload

Parameter	Is Mandatory	Comments
isPCM	Yes	Boolean, "true" if the stream is PCM format. Currently, this param must be true
channels	Yes	A number, to indicate the number of channels. May be "1" or "2"
backgroundNoise	Yes	A number represents the background noise in the recording. The higher the number the higher the background noise. Standard recording should have value of 1000
bitRate	Yes	A number represents the audio bit-rate. Currently 8 and 16 are supported
sampleRate	Yes	The audio sample rate. Supported values are: 6000, 8000, 11025, 16000, 22050, 44100, 48000
outputType	No	Can be "json" ot "text". Default is "json"

Handshake Done

The docker sends this event as a response to a "handshake" event. On success, the payload will contain the streamId, on error it will hold the error data.

Event name: handshake-done

Sent by: analysis server

Event payload:

Parameter	Comments
success	Boolean, "true" handshake succeed
errorCode	an error code, in case the handshake failed (success == false)
error	an error message, in case the handshake failed (success == false)

```javascript
function handshake(socket, isPCM, channels, backgroundNoise, bitRate, sampleRate , outputType) {

    return new Promise((y, n) => {

        const onHandshakeDone = r => {

            socket.off("handshake-done", onHandshakeDone);

            if (r.success) {
                console.log("handshake ok");
            } else {
                console.logr?.error || "Unexpected error occurred on handshake");
            }
        };

        socket.on("handshake-done", onHandshakeDone);

        socket.emit("handshake", {
            isPCM: isPCM,
            channels: channels,
            backgroundNoise: backgroundNoise,
            bitRate: bitRate,
            sampleRate: sampleRate,
            outputType: outputType,
        });
    });
}
```

Audio Stream

After a successful handshake, the client starts sending audio-buffers to the docker. The docker will asynchronously send the analysis results to the client.

Event: audio-stream

Sent by: client

Event payload: An audio buffer

```
function streamAudio(socket, buffer) {
    socket.emit("audio-stream", buffer);
}
```

Audio Analysis

As the client sends audio buffers, the docker starts analyzing it. Whenever the docker build a new segment, it pushes the segment analysis to the client using the "audio-analysis" event.

Event: audio-analysis

Sent by: docker

Event payload: Segment analysis data. Please refer to API Response for analysis details.

```
function streamAudio(socket, buffer) {
    socket.emit("audio-stream", buffer);
}
```

Fetch analysis report

At the end on the call, it is possible to send a "fetch-analysis-call" event to the docker.

The docker will respond with an "analysis-report-ready" event containing the call report (same report as accepted on a file-analysis call).

Event: fetch-analysis-call

Event parameters

Parameter	Is Mandatory	
outputFormat	No	May be "json" (default) or "text"
fetchSegments	No	May be true (default) or false

Analysis report ready

After sending a "fetch analysis report" event, the analysis server respond and "analysis report ready" event.

The response will contain the same analysis report as provided by a regular file analysis.

Event: analysis-report-ready

Sent by: analysis server

```javascript
function fetchAnalysisReport(socket, outputFormat, fetchSegments) {
    return new Promise((y, n) => {
        const analysisReportReady = r => {

            socket.off("analysis-report-ready", analysisReportReady);

            if (r.success) {

                y(r.data);
            } else {
                n(new Error(r?.error || "Unexpected error occurred on finalize"));
            }

            socket.disconnect();
        };

        socket.on("analysis-report-ready", analysisReportReady);

        socket.emit("fetch-analysis-report", {
            outputFormat: outputFormat,
            fetchSegments: fetchSegments,
        });
    });
}
```

Revision #1
Created 12 March 2025 21:01:13 by mauricio olarte