Sample code - avoid promises

```
/*

    This example demonstrates real-time analysis (streaming) API.
    On a real-world situation, that audio will come from switchboard.
    On this example we read a file and stream is as it was an audio comming from the switchboard.

    The analysis algoritm uses 11025 samples per second, and 16 bits per sample, PCM, therefore this is the recommencded
    audio format.

    In case a different sample-rate streamed, the docker convert the original format to 11025, and this process slows the analysis process by about 9 times.
*/

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

const FILE_PATH = "[FILE PATH]"
const ANALYSIS_SERVER_URL = "ws://localhost:2259"; // assuming the docker is running on the local host
const AUDIO_PACKET_SIZE_MS = 300; // 300 ms is the remommended buffer size

streamFile(ANALYSIS_SERVER_URL, FILE_PATH, "json")

function streamFile(analysisServerUrl, inputFilePath, outputType = "json") {

    const wav = new wavefile.WaveFile();
    const buffer = fs.readFileSync(inputFilePath)

    wav.fromBuffer(buffer);

    if (wav.fmt.bitsPerSample % 8 !== 0) {
        throw new Error("invalid bitsPerSample");
    }

    const socket = io.connect(analysisServerUrl, {
        transports: ["websocket"]
    });


    socket
        .on("connect", () => sendHandshake(socket, wav, outputType))
        .on("connect_error", (err) => {
            throw err
        })
        .on("audio-analysis-error", (err) => {
            throw err
        })
        .on("handshake-done", (r) => sendSamples(socket, wav))
        .on('audio-analysis', (r) => processAnalysisData(socket, r))
        .on("analysis-report-ready", (r) => analysisReportReady(socket, r))
        .on("audio-analysis-completed", (r) => {
```

```
            if (!r.success) {
                throw new Error(r?.error || "Unexpected error occurred on finalize");
            }

            fetchAnalysisReport(socket);
        })
}

const fetchAnalysisReport = (socket) => {

    socket.emit("fetch-analysis-report", {
        // return report on json format
        outputFormat: "json",
        // true -> return on the report all the segments in the call.
        // in that case, the result will be similar to an offline analysis
        fetchSegments: true,
    });
}

const analysisReportReady = (socket, r) => {

    if (!r.success) {
        throw new Error(r?.error || "Unexpected error occurred on finalize");
    }

    console.log(JSON.stringify(r));

    socket.disconnect();
};

const processAnalysisData = (socket, {success, data}) => {

    if (!success) {
        throw new Error(data.error);
    }

    if (data.done) {
        console.log(JSON.stringify(data) + "\r\n");
    }
}

const sendHandshake = (socket, wav, outputType) => {

    socket.emit("handshake", {
        isPCM: wav.fmt.audioFormat === 1,
        channels: wav.fmt.numChannels,
        backgroundNoise: 1000,
        bitRate: wav.fmt.bitsPerSample,
        sampleRate: wav.fmt.sampleRate,
        outputType,
    });
}
```

```
function calcPacketSizeBytes(wav, audioLengthMS){

    // this function assumes bitsPerSample = 16 or 8 ==> the 2 supported values by the docker

    const singleSampleTimeMS = 1000 / wav.fmt.sampleRate;
    const singleSampleSizeBytes = wav.fmt.bitsPerSample / 8 * wav.fmt.numChannels;

    // Need to round to integer, as audioLengthMS / singleSAmpleTimeMS most likely will not be an integer
    const requiredSamplesCount = Math.round(audioLengthMS / singleSampleTimeMS);

    const packetSizeBytes = requiredSamplesCount * singleSampleSizeBytes;

    return packetSizeBytes;
}

const sendSamples = (socket, wav) => {

    let packetSize = calcPacketSizeBytes(wav, AUDIO_PACKET_SIZE_MS);
    let offset = 0;

    const samples = wav.data.samples;

    const interval = setInterval(() => {

        if (!socket.connected) {
            throw new Error("Socket is not connected");
        }

        const arraySize = Math.min(packetSize, samples.byteLength - offset);

        if (arraySize) {

            socket.emit("audio-stream", samples.slice(offset, offset + arraySize));
            offset += arraySize;

        } else {
            socket.emit("audio-stream",Buffer.alloc(0));
            clearInterval(interval);
        }

    }, AUDIO_PACKET_SIZE_MS);
```

Sample code - Using promises

```
/*

    This example demonstrates real-time analysis (streaming) API.
    On a real-world situation, that audio will come from switchboard.
    On this example we read a file and stream is as it was an audio comming from the switchboard.

    The analysis algoritm uses 11025 samples per second, and 16 bits per sample, PCM, therefore this is the recommencded
    audio format.

    In case a different sample-rate streamed, the docker convert the original format to 11025, and this process slows the analysis process by about 9 times.
*/

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


const FILE_PATH = "[FILE PATH]"
const ANALYSIS_SERVER_URL = "ws://localhost:2259"; // assuming the docker is running on the local host
const AUDIO_PACKET_SIZE_MS = 300; // 300 ms is the remommended buffer size


streamFile(FILE_PATH, "json");

async function streamFile(filePath, count = 1, outputType) {

    try {

        await sendFile(ANALYSIS_SERVER_URL, filePath, outputType);

    } catch (err) {

        console.error(err);
    }
}

async function sendFile(analysisServerUrl, inputFilePath, outputType = "json") {

    const wav = new wavefile.WaveFile();
    const buffer = await fs.promises.readFile(inputFilePath)

    wav.fromBuffer(buffer);

    if (wav.fmt.bitsPerSample % 8 !== 0) {
        throw new Error("invalid bitsPerSample");
    }

    let socket = null;

    try {
        socket = await connect(analysisServerUrl)
```

```
        socket.on("audio-analysis-error", (err) => {
            throw err;
        });

    await handshake(socket, wav, outputType);

    await sendSamples(socket, wav);

    } catch (err) {

        throw err;

    } finally {
        socket?.disconnect();
    }
}

function connect(url) {

    return new Promise((resolve,reject) => {

        const socket = io.connect(url, {
            transports: ["websocket"]
        });

        socket.on("connect", () => {
            resolve(socket)
        });

        socket.on("connect_error", (err) => {
            reject(err);
        });
    });

}

function handshake(socket, wav, outputType) {

    return new Promise((resolve, reject) => {

        const onHandshakeDone = r => {

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

            if (r.success) {
                resolve(r.data);
            } else {
                reject(new Error(r?.error || "Unexpected error occurred on handshake"));
            }
        };

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

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

        socket.emit("handshake", {
            isPCM: wav.fmt.audioFormat === 1,
            channels: wav.fmt.numChannels,
            backgroundNoise: 1000,
            bitRate: wav.fmt.bitsPerSample,
            sampleRate: wav.fmt.sampleRate,
            sensitivity: "normal",
            outputType,
        });
    });
}


function fetchAnalysisReport(socket) {

    return new Promise((resolve, reject) => {

        const analysisReportReady = r => {

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

            if (r.success) {

                console.log(JSON.stringify(r));

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

            socket.disconnect();
        };

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

        socket.emit("fetch-analysis-report", {
            // return report on json format
            outputFormat: "json",
            // true -> return on the report all the segments in the call.
            // in that case, the result will be similar to an offline analysis
            fetchSegments: true,
        });

    });
}
```

```
function calcPacketSizeBytes(wav, audioLengthMS){

    // this function assumes bitsPerSample = 16 or 8 ==> the 2 supported values by the docker

    const singleSampleTimeMS = 1000 / wav.fmt.sampleRate;
    const singleSampleSizeBytes = wav.fmt.bitsPerSample / 8 * wav.fmt.numChannels;

    // Need to round to integer, as audioLengthMS / singleSAmpleTimeMS most likely will not be an integer
    const requiredSamplesCount = Math.round(audioLengthMS / singleSampleTimeMS);

    const packetSizeBytes = requiredSamplesCount * singleSampleSizeBytes;

    return packetSizeBytes;
}

async function sendSamples(socket, wav) {

    let packetSize = calcPacketSizeBytes(wav, AUDIO_PACKET_SIZE_MS);
    let offset = 0;

    return new Promise((resolve, reject) => {

        socket.on("audio-analysis-error", err => {

            // The docker will send this in case of an error.
            // The err param will hold the error details
            socket.disconnect();
            reject(err);
        });

        socket.on('audio-analysis', async (r) => {

            if (r.success) {

                if (r.data.done) {
                    console.log(JSON.stringify(r.data) + "\r\n");
                }
            }

            if (!r.success) {

                socket.disconnect();
                reject(new Error(r.error));
            }
        });

        socket.on('audio-analysis-completed', async (r) => {
```

```
        if (r.success) {

            await fetchAnalysisReport(socket);
        }
    });

    function send() {

        let arraySize = (wav.data.samples.byteLength - offset < packetSize)? wav.data.samples.byteLength - offset:packetSize;
        const array = wav.data.samples.slice(offset, offset + arraySize);

        offset += arraySize;

        socket.emit("audio-stream", array);

        if (offset < wav.data.samples.length && socket.connected) {

            setTimeout(send, AUDIO_PACKET_SIZE_MS);

        } else if(socket.connected){

            socket.emit("audio-stream", Buffer.alloc(0));

        } else {
            resolve();
        }
    }

    send();
});
```

Emotion Logic docker supports integrations with 2 STT (Speech To Text) providers

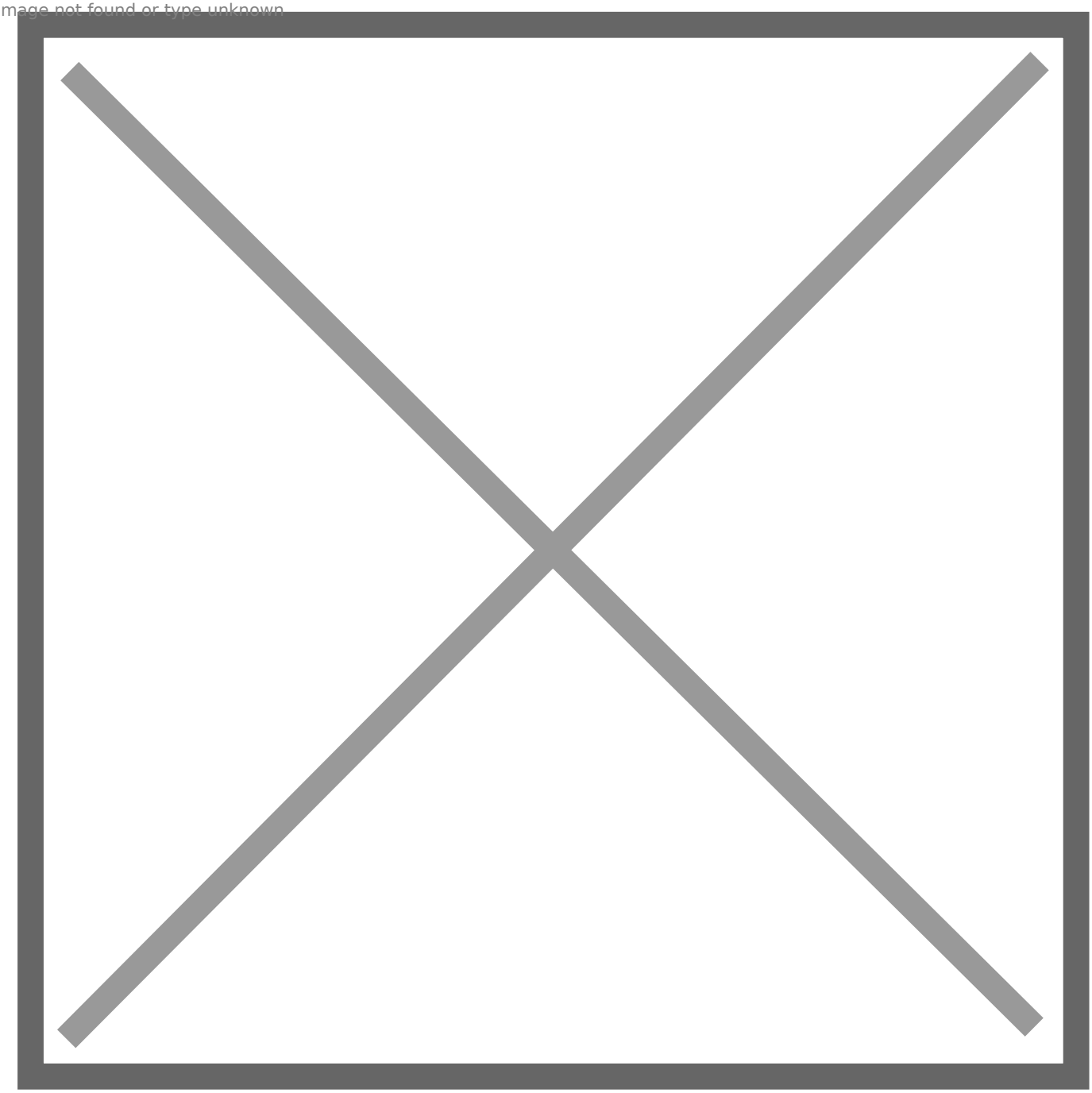
- Deepgram
- Speechmatics

By setting your STT provider API Key, the Emotion Logic anlysis docker will sync its analysis to the STT results.

When activating STT on the docker, each analysis sigment will contain the spoken text at the time of the segment.

How to set STT provider API Key

Image not found or type unknown



1. Open the Docker dashboard and navigate to the “Integrations” tab.

2. If you do not have an account with one of the supported Speech-to-Text (STT) providers, please visit:

• Deepgram

• Speechmatics

3. Create an API Key with your chosen STT provider.

4. Enter the STT API Key in the appropriate field.

5. Save your changes.

6. Ensure that you include "useSpeechToText: true" in your analysis requests.

Release Notes: Version 7.32.1

New Features: • LOVE Values: Added all LOVE values to enhance the emotional analysis capabilities.

Improvements: • MostFanatic Function: Optimization of the MostFanatic function for better performance and accuracy.
• Passion Detection: Modified the SAF value function to improve the detection of passion.
• Strengths and Challenges: Function updated to relate to averages as a baseline, providing relative strengths and weaknesses. The function now includes “uneasy” and “arousal” metrics to keep the assessment relative.

Bug Fixes: • Channel Similarity: Fixed a bug related to similarity calculations between channels.

Updates:

• Excitement and Uncertainty: Updated the functions for Excitement and Uncertainty to align with new norms.
• BG Auto Test: Modified the BG auto test functionality. Tests are now disabled for segments shorter than 5 seconds. Users should utilize FIX BG or STT for segmentation in such cases.

Release Notes for LVA7 Tech. 7.30.1

Version Update:

Optimization: Improved CallPriority scores and call classifications tailored for call center scenarios.
Bug Fix: Resolved issues with time pointer shifts in lengthy files.
Modification: Updated FeelGPT protocol terminology to clarify message meanings (changed "Passion" to "arousal" and "passion peak" to "arousal peak").

Release Notes for LVA7 Tech. 7.29.3

We are excited to announce the release of LVA7, a significant update to our analytics platform. This version introduces several enhancements and fixes aimed at improving accuracy, usability, and comprehensiveness of risk assessments and personality insights. Here's what's new:

Enhancements:

Objective Risk Formula Optimization:

1. We've fine-tuned the Objective (OZ) risk formulas to better incorporate inaccuracy indicators, resulting in more nuanced risk assessments.

2. Users can expect a modest recalibration of risk scores, with a greater number of risk indicators and inaccuracies now being flagged.

3. For those preferring the previous version's risk evaluation, the option to revert is available by setting sensitivity: bwc1 for backward compatibility.

Introduction of Final Risk Score:

A new "Final Risk" score has been added to the risk summaries, amalgamating objective and subjective risk evaluations for a comprehensive overview.

When only one type of risk is assessed, the Final Risk score will reflect that singular assessment.

The calculation method for the Final Risk score in the Topics and Questions sections has been updated for enhanced accuracy.

Personality Assessment Enhancement: (In supported applications)

The questionnaire API now supports personality assessments at the question level.

Use isPersonality: true to designate a question for personality evaluation.

Use isPersonality: false to designate a question for risk assessment only.

Questions with a non-zero weight parameter will contribute to both personality and risk assessments. Set weight: 0 to exclude a question from risk evaluation.

Important Update Regarding isPersonality Setting:

To ensure a seamless transition and maintain backward compatibility, the isPersonality option will default to True in the current release. Be aware that this behavior is slated for a future change. We strongly recommend that users review and adjust their questionnaire settings accordingly to ensure accurate core competencies values analysis. Remember, only questions explicitly marked with isPersonality: true are factored into this analysis.

Bug Fixes:

Emotion Diamond Real-Time Values Correction:

An issue affecting the real-time values displayed on Emotion Diamond for channel 1 has been addressed, ensuring accurate emotional insight representation.

The old Nemesysco's cloud response and the new EmotionLogic response

Nemesysco's cloud response	New Emotion-Logic response	Remarks
"RISKREPT":["Topic1;C0;6;90;95", "Topic2;C0;6;95;100"]	{ "data": { "reports": { "risk": { "topics": [{ "_id": "question1", "averageJQ": 26, "averageVol1": 892, "averageVol2": 73, "maxSOS": 103, "riskObjective": 43, "riskSubjective": 85, "segmentsCount": 34 }] } } } }	The Topics Risk report is now more detailed and contains more items. The old response structure was: *Topic Name;Channel ID;Segment Count; Risk;Max SOS* Topic Name is now "_id" "***C0***" - old Channel ID - this param was dropped from the new version ***Segment count*** maps to the new ***segmentsCount*** The old ***RISK*** maps to the new "***riskObjective***" and uses the same scale and values. "***SOS***" maps to the new "***maxSOS***" and have the same meaning and scales.

"RISKREPQ":["Topic1;Question1;C0;1;22;75;10", "Topic1;Question2;C0;1;12;93;20", "Topic2;Question3;C0;2;84;100;30", "Topic2;Question4;C0;2;55;92;40"],	"reports": { "risk": { "questions": [{ "_id": "topic1", "averageJQ": 26, "averageVol1": 892, "averageVol2": 73, "maxSOS": 103, "riskObjective": 43, "riskSubjective": 85, "segmentsCount": 34 }] } }	The Questions Risk report is now more detailed and contains more items. The old repsonse structure was: *Topic Name;Question Id;Channel ID;Segment Count; Risk;Max SOS* Question Name is now "_id" "***C0***" - old Channel ID - this param was dropped from the new version ***Segment count*** maps to the new ***segmentsCount*** The old ***RISK*** maps to the new "***riskObjective***" and uses the same scale and values. "***SOS***" maps to the new "***maxSOS***" and have the same meaning and scales.
"EDPREPT":["Leadership;Leading by example;C0;1;25;1;38;1;20;13;83;100;100;41", "Leadership;Approach toward difficulties;C0;1;19;1;31;1;60;25;68;67;100;57", "Leadership;Leadership skills;C0;2;25;1;23;1;32;22;81;100;100;60", "Leadership;Influencing others;C0;2;38;1;24;1;34;23;81;68;98;42"]		Emotional Diamond data by question
"SEG":["TotalSeg#;Seg#;TOPIC;QUESTION;Channel;StartPos;EndPos;OnlineLVA;OfflineLVA; Risk1;Risk2;RiskOZ;OZ1/OZ2/OZ3;Energy;Content;Upset;Angry;Stressed;COGLevel; EMOLevel;Concentration;Anticipation; Hesitation;EmoBalance;IThink;Imagin; SAF;OCA; EmoCogRatio;ExtremeEmotion;CogHighLowBalance;VoiceEnergy;LVARiskStress; LVAGLBStress;LVAEmoStress;LVACOGStress;LVAENRStress", "SEG1;0001;Leadership;Leading by example;C0;0.90;1.40;Calibrating... (-2);<OFFC01>;0;0; 145;4/3/1232;4;0;0;0;0;15;30;30;30;14;51;0;0;0;551;100;11;58;1356 / 66;0;0;0;0;0"]		Segments data by the selected application structure

Initializing Docker with Environment Variables

In scenarios where Docker containers need to be initialized automatically—such as when deployed by Kubernetes—manual initiation through the Docker dashboard is not possible. Instead, the container can be configured to initialize itself automatically by passing the necessary environment variables.

Mandatory Environment Variables

To ensure proper authentication and functionality, the following environment variables must be provided:

• **PLATFORM_APIKEY** – API key for emlo.cloud

• **PLATFORM_APIKEY_PASSWORD** – Password for the emlo.cloud API key

To run the container with these variables, use the following command:

```
docker run --rm -p 8080:8080 -p 2259:2259 \
    -e "PLATFORM_APIKEY=test" \
    -e "PLATFORM_APIKEY_PASSWORD=test" \
    --name nms-server nemesysco/on_premises
```

Optional Environment Variables

The following optional environment variables can be used to integrate with third-party services or modify the container's behavior:

• **DEEPGRAM_URL** – Base URL for the Deepgram Speech-to-Text (STT) API

• **STT_KEY** – API key for Deepgram's STT service

• **SPEECHMATICS_KEY** – API key for Speechmatics STT API

• **WHISPER_BASE_URL** – Base URL for Whisper STT API

• **DISABLE_UI** – A flag to disable the Docker UI. Assigning any value to this variable will disable the UI.

By configuring these variables appropriately, the container can be tailored to meet specific deployment needs.

Revision #2
Created 12 March 2025 21:01:13
Updated 13 June 2025 13:55:34