Forniscimi una spiegazione molto dettagliata (come se dovessi creare un file pdf di descrizione del codice) del comportamento di tutti gli elementi di questo codice CodePen:

HTML

``` ``` ``` ``` Ready
Play with Effects
```
``` ```
```
Download WAV Mic Download WAV with Effects

CSS
```
body {
font-family: system-ui, sans-serif;
background: #020617;
color: #e5e7eb;
display: flex;
justify-content: center;
padding: 16px;
}

.app {
max-width: 780px;
width: 100%;
background: #020617;
border-radius: 12px;
padding: 16px;
box-shadow: 0 18px 30px rgba(0,0,0,0.6);
border: 1px solid #111827;
}

.section {
margin-bottom: 12px;
border-top: 1px solid #1f2937;
padding-top: 10px;
}
```

```
.waveform-row {
border-top: none;
padding-top: 0;
}

.rec-row {
display: flex;
justify-content: space-between;
gap: 12px;
align-items: center;
}

.block {
display: flex;
flex-wrap: wrap;
align-items: center;
gap: 6px;
}

/* Standard Buttons */
button {
background: #2563eb;
border: none;
color: white;
padding: 6px 10px;
border-radius: 6px;
cursor: pointer;
font-size: 13px;
}
button:disabled {
background: #4b5563;
cursor: not-allowed;
}

#status {
font-size: 12px;
opacity: 0.8;
}

/* Rec/Stop Buttons */
.btn-round-rec {
width: 30px;
height: 30px;
border-radius: 50%;
background: #dc2626;
position: relative;
padding: 0;
}
```

```
.btn-round-rec::after {
content: "";
position: absolute;
inset: 7px;
background: #fecaca;
border-radius: 50%;
}
.btn-square-stop {
width: 30px;
height: 30px;
border-radius: 8px;
background: #dc2626;
position: relative;
padding: 0;
}
.btn-square-stop::after {
content: "";
position: absolute;
inset: 7px;
background: #fecaca;
border-radius: 4px;
}

/* Knobs */
.knobs-grid {
display: grid;
grid-template-columns: repeat(3, minmax(0, 1fr));
gap: 12px;
justify-items: center;
}
.knob-wrapper {
display: flex;
flex-direction: column;
align-items: center;
background: #020617;
border-radius: 10px;
padding: 10px 4px;
border: 1px solid #1f2937;
width: 100%;
max-width: 140px;
}
.knob {
width: 60px;
height: 60px;
border-radius: 50%;
background: radial-gradient(circle at 30% 30%, #4b5563, #020617);
```

```
position: relative;
cursor: pointer;
transform: rotate(-135deg);
transition: box-shadow 0.1s ease;
box-shadow: 0 0 0 1px #111827, 0 4px 10px rgba(0,0,0,0.7);
}
.knob::after {
content: "";
position: absolute;
width: 4px;
height: 22px;
background: #e5e7eb;
border-radius: 4px;
top: 10px;
left: 50%;
transform: translateX(-50%);
}
.knob:active {
box-shadow: 0 0 0 1px #38bdf8, 0 4px 14px rgba(56,189,248,0.5);
}
.knob-label {
font-size: 12px;
margin-top: 6px;
}
.knob-value {
font-size: 11px;
opacity: 0.8;
}

/* Presets */
.preset-row {
display: grid;
grid-template-columns: repeat(4, minmax(0, 1fr));
gap: 8px;
border-top: none;
padding-top: 4px;
}
.preset {
background: #10b981;
text-align: center;
width: 100%;
padding: 8px 0;
}

/* Download Buttons */
.download-row {
display: flex;
```

```
justify-content: center;
gap: 12px;
}
.download-row button {
min-width: 160px;
}

/* Waveform Canva */
canvas#waveform {
width: 100%;
background: #000;
border-radius: 6px;
border: 1px solid #111827;
}
```

JS

```
//
// ========================================================================
// STATE & GLOBAL VARIABLES
//
// ========================================================================

// Base recording state
let mediaRecorder, recordedChunks = [], audioBlob = null, audioUrl = null;
let audioCtx, lowpassFilter, highpassFilter, delayNode, delayFeedback,
reverbConvolver, dryGain, wetGain, masterGain, analyser, dataArray,
animationId;

// Track of the active processed source to avoid overlapping playback
let activeProcessedSource = null;
let mediaElementSource = null;

//
// ========================================================================
// UI
//
// ========================================================================

// Fixed UI elements
const btnStartRec = document.getElementById("btnStartRec");
const btnStopRec = document.getElementById("btnStopRec");
const btnPlayProcessed = document.getElementById("btnPlayProcessed");
const btnDownloadWav = document.getElementById("btnDownloadWav");
const btnDownloadProcessedWav = document.getElementById("btnDownloadProcessedWav");
const statusEl = document.getElementById("status");
const player = document.getElementById("player");
```

```
// Waveform canva
const waveformCanvas = document.getElementById("waveform");
const wfCtx = waveformCanvas.getContext("2d");

// Dynamic UI containers
const knobsContainer = document.getElementById("knobsContainer");
const presetsContainer = document.getElementById("presetsContainer");

//
==========================================================================
// CONFIGURATION (KNOBS, PRESETS, PARAMETERS VALUES)
//
==========================================================================

// Knobs configuration
const knobsConfig = [
{ id: "gain", label: "Volume", min: 0, max: 4, step: 0.01, value: 1 },
{ id: "pitch", label: "Pitch", min: 0.5, max: 2, step: 0.01, value: 1 },
{ id: "lowpass", label: "Lowpass", min: 200, max: 20000, step: 1, value: 20000 },
{ id: "highpass", label: "Highpass",min: 10, max: 5000, step: 1, value: 10 },
{ id: "delayTime", label: "Delay", min: 0, max: 1, step: 0.01, value: 0 },
{ id: "reverbMix", label: "Reverb", min: 0, max: 1, step: 0.01, value: 0.3 }
];

// Presets configuration
const presetsConfig = {
clean: {
label: "Clean",
params: { gain: 1, lowpass: 20000, highpass: 10, delayTime: 0, reverbMix: 0.1, pitch: 1 }
},
phone: {
label: "Phone",
params: { gain: 1, lowpass: 3500, highpass: 400, delayTime: 0, reverbMix: 0.0, pitch: 1 }
},
hall: {
label: "Hall",
params: { gain: 1, lowpass: 18000, highpass: 80, delayTime: 0.25, reverbMix: 0.7, pitch: 1 }
},
lofi: {
label: "Lo-Fi",
params: { gain: 0.9, lowpass: 5000, highpass: 150, delayTime: 0.12, reverbMix: 0.4, pitch: 0.9 }
}
};

// Parameter values
const paramValues = {
gain: 1,
pitch: 1,
```

```
lowpass: 20000,
highpass: 10,
delayTime: 0,
reverbMix: 0.3
};

//
==========================================================================
// KNOB UTILITIES
//
==========================================================================

const lerp = (a, b, t) => a + (b - a) * t;
const clamp = (v, min, max) => Math.min(max, Math.max(min, v));
const valueToAngle = (v, min, max) => -135 + ((v - min) / (max - min)) * 270;

//
==========================================================================
// DYNAMIC UI CREATION (KNOBS & PRESETS)
//
==========================================================================

// Create knob elements dynamically
function createKnobs() {
knobsConfig.forEach(cfg => {
const wrapper = document.createElement("div");
wrapper.className = "knob-wrapper";
```

```
const knob = document.createElement("div");
knob.className = "knob";
knob.dataset.target = cfg.id;
knob.dataset.min = cfg.min;
knob.dataset.max = cfg.max;
knob.dataset.step = cfg.step;

const label = document.createElement("div");
label.className = "knob-label";
label.textContent = cfg.label;

const valueEl = document.createElement("div");
valueEl.className = "knob-value";
valueEl.id = cfg.id + "Val";
valueEl.textContent = (cfg.id === "lowpass" || cfg.id === "highpass")
  ? Math.round(cfg.value)
  : cfg.value.toFixed(2);

wrapper.appendChild(knob);
wrapper.appendChild(label);
wrapper.appendChild(valueEl);
knobsContainer.appendChild(wrapper);
```

```
});
}

// Create preset buttons dynamically
function createPresets() {
Object.entries(presetsConfig).forEach(([name, preset]) => {
const btn = document.createElement("button");
btn.className = "preset";
btn.dataset.preset = name;
btn.textContent = preset.label;
btn.addEventListener("click", () => applyPreset(name));
presetsContainer.appendChild(btn);
});
}

createKnobs();
createPresets();

//
// ============================================================================
// AUDIO GRAPH INITIALIZATION
//
// ============================================================================

function initAudioGraph() {
if (audioCtx) return;

audioCtx = new (window.AudioContext || window.webkitAudioContext)();

lowpassFilter = audioCtx.createBiquadFilter();
lowpassFilter.type = "lowpass";
lowpassFilter.frequency.value = paramValues.lowpass;

highpassFilter = audioCtx.createBiquadFilter();
highpassFilter.type = "highpass";
highpassFilter.frequency.value = paramValues.highpass;

delayNode = audioCtx.createDelay(5.0);
delayNode.delayTime.value = paramValues.delayTime;

delayFeedback = audioCtx.createGain();
delayFeedback.gain.value = 0.3;
delayNode.connect(delayFeedback);
delayFeedback.connect(delayNode);

reverbConvolver = audioCtx.createConvolver();
reverbConvolver.buffer = createReverbImpulse(audioCtx, 2.5, 2.0);

dryGain = audioCtx.createGain();
wetGain = audioCtx.createGain();
```

```
masterGain = audioCtx.createGain();
masterGain.gain.value = paramValues.gain;

const mix = paramValues.reverbMix;
dryGain.gain.value = 1 - mix;
wetGain.gain.value = mix;

analyser = audioCtx.createAnalyser();
analyser.fftSize = 2048;
dataArray = new Uint8Array(analyser.fftSize);

lowpassFilter.connect(highpassFilter);
highpassFilter.connect(delayNode);
delayNode.connect(dryGain);
delayNode.connect(reverbConvolver);
reverbConvolver.connect(wetGain);
dryGain.connect(masterGain);
wetGain.connect(masterGain);
masterGain.connect(analyser);
analyser.connect(audioCtx.destination);
}

// Connect HTML <audio> element to analyser to draw waveform on raw playback
function connectPlayerToAnalyser() {
if (!audioCtx || !player) return;
if (mediaElementSource) return;
mediaElementSource = audioCtx.createMediaElementSource(player);
mediaElementSource.connect(analyser);
}

// Generate an impulse response for the reverb
function createReverbImpulse(context, duration, decay) {
const rate = context.sampleRate;
const length = rate * duration;
const impulse = context.createBuffer(2, length, rate);
for (let c = 0; c < impulse.numberOfChannels; c++) {
const chData = impulse.getChannelData(c);
for (let i = 0; i < length; i++) {
const n = (length - i) / length;
chData[i] = (Math.random() * 2 - 1) * Math.pow(n, decay);
}
}
return impulse;
}

//
================================================================
// WAVEFORM
```

```
//
========================================================================

function drawWaveform() {
if (!analyser) return;
animationId = requestAnimationFrame(drawWaveform);

const w = waveformCanvas.width;
const h = waveformCanvas.height;

analyser.getByteTimeDomainData(dataArray);

wfCtx.fillStyle = "#000";
wfCtx.fillRect(0, 0, w, h);

wfCtx.lineWidth = 2;
wfCtx.strokeStyle = "#38bdf8";
wfCtx.beginPath();

const slice = w / dataArray.length;
let x = 0;

for (let i = 0; i < dataArray.length; i++) {
const v = dataArray[i] / 128.0;
const y = v * h / 2;
if (i === 0) wfCtx.moveTo(x, y);
else wfCtx.lineTo(x, y);
x += slice;
}

wfCtx.lineTo(w, h / 2);
wfCtx.stroke();
}

//
========================================================================
// PLAYER EVENTS (RAW PLAYBACK + WAVEFORM)
//
========================================================================

player.addEventListener("play", () => {
initAudioGraph();
connectPlayerToAnalyser();
if (!animationId) drawWaveform();

// Stop processed source if it is playing, to avoid overlaps
if (activeProcessedSource) {
try {
activeProcessedSource.stop();
} catch (e) {}
```

activeProcessedSource = null;
}

// Re-enable "Play with Effects" (we are playing the dry version)
btnPlayProcessed.disabled = false;
});

player.addEventListener("ended", () ⇒ {
if (animationId) {
cancelAnimationFrame(animationId);
animationId = null;
}
});

//
// MIC RECORDING
//

btnStartRec.addEventListener("click", async () ⇒ {
try {
const stream = await navigator.mediaDevices.getUserMedia({ audio: true });
mediaRecorder = new MediaRecorder(stream);
recordedChunks = [];

```
  mediaRecorder.ondataavailable = e => {
    if (e.data.size) recordedChunks.push(e.data);
  };

  mediaRecorder.onstop = () => {
    audioBlob = new Blob(recordedChunks, { type: "audio/webm" });
    if (audioUrl) URL.revokeObjectURL(audioUrl);
    audioUrl = URL.createObjectURL(audioBlob);
    player.src = audioUrl;

    btnPlayProcessed.disabled = false;
    btnDownloadWav.disabled = false;
    btnDownloadProcessedWav.disabled = false;
    statusEl.textContent = "The Record Is Ready";
  };

  mediaRecorder.start();
  btnStartRec.disabled = true;
  btnStopRec.disabled = false;
  statusEl.textContent = "Recording...";
```

} catch (e) {
console.error(e);
statusEl.textContent = "Error: The Mic Is On";

```
}
});

// Stop microphone recording
btnStopRec.addEventListener("click", () => {
if (mediaRecorder && mediaRecorder.state === "recording") {
mediaRecorder.stop();
mediaRecorder.stream.getTracks().forEach(t => t.stop());
}
btnStartRec.disabled = false;
btnStopRec.disabled = true;
});

//
// ============================================================================
// DOWNLOAD RAW WAV
//
// ============================================================================

btnDownloadWav.addEventListener("click", async () => {
if (!audioBlob) return;
initAudioGraph();

const arrayBuffer = await audioBlob.arrayBuffer();
const audioBuffer = await audioCtx.decodeAudioData(arrayBuffer);
const wavBuffer = audioBufferToWav(audioBuffer);
const wavBlob = new Blob([wavBuffer], { type: "audio/wav" });

downloadBlob(wavBlob, "Recording.wav");
});

//
// ============================================================================
// PLAYBACK WITH EFFECTS
//
// ============================================================================

btnPlayProcessed.addEventListener("click", async () => {
if (!audioBlob) return;
initAudioGraph();

// Stop HTML player if it is playing
if (!player.paused) {
player.pause();
player.currentTime = 0;
}

// Stop previous processed source if any
if (activeProcessedSource) {
```

```
try {
activeProcessedSource.stop();
} catch (e) {}
activeProcessedSource = null;
}

const arrayBuffer = await audioBlob.arrayBuffer();
const audioBuffer = await audioCtx.decodeAudioData(arrayBuffer);
const source = audioCtx.createBufferSource();
source.buffer = audioBuffer;
source.playbackRate.value = paramValues.pitch;
source.connect(lowpassFilter);

activeProcessedSource = source;
btnPlayProcessed.disabled = true;

source.onended = () => {
if (activeProcessedSource === source) {
activeProcessedSource = null;
}
btnPlayProcessed.disabled = false;
};

source.start();
if (!animationId) drawWaveform();
});

//
// ========================================================================
// DOWNLOAD PROCESSED WAV
//
// ========================================================================

btnDownloadProcessedWav.addEventListener("click", async () => {
if (!audioBlob) return;

const arr = await audioBlob.arrayBuffer();

const probeCtx = new (window.AudioContext || window.webkitAudioContext)();
const decoded = await probeCtx.decodeAudioData(arr);
const duration = decoded.duration;
const sampleRate = decoded.sampleRate;
probeCtx.close();

const length = Math.ceil(duration * sampleRate);
const offlineCtx = new (window.OfflineAudioContext || window.webkitOfflineAudioContext)(
1,
length,
```

```
sampleRate
);

const source = offlineCtx.createBufferSource();
source.buffer = decoded;

const lp = offlineCtx.createBiquadFilter();
lp.type = "lowpass";
lp.frequency.value = paramValues.lowpass;

const hp = offlineCtx.createBiquadFilter();
hp.type = "highpass";
hp.frequency.value = paramValues.highpass;

const del = offlineCtx.createDelay(5.0);
del.delayTime.value = paramValues.delayTime;

const fb = offlineCtx.createGain();
fb.gain.value = 0.3;
del.connect(fb);
fb.connect(del);

const conv = offlineCtx.createConvolver();
conv.buffer = createReverbImpulse(offlineCtx, 2.5, 2.0);

const dry = offlineCtx.createGain();
const wet = offlineCtx.createGain();
const master = offlineCtx.createGain();

master.gain.value = paramValues.gain;
const mix = paramValues.reverbMix;
dry.gain.value = 1 - mix;
wet.gain.value = mix;

source.playbackRate.value = paramValues.pitch;

source.connect(lp);
lp.connect(hp);
hp.connect(del);
del.connect(dry);
del.connect(conv);
conv.connect(wet);
dry.connect(master);
wet.connect(master);
master.connect(offlineCtx.destination);

source.start(0);
const rendered = await offlineCtx.startRendering();
const wavBuffer = audioBufferToWav(rendered);
const wavBlob = new Blob([wavBuffer], { type: "audio/wav" });
```

```
downloadBlob(wavBlob, "Recording_with_effects.wav");
statusEl.textContent = "WAV with Effects Is Ready";
});

//
// ================================================================
// OTHER FUNCTIONS
//
// ================================================================

// Download a Blob as a file
function downloadBlob(blob, filename) {
const url = URL.createObjectURL(blob);
const a = document.createElement("a");
a.style.display = "none";
a.href = url;
a.download = filename;
document.body.appendChild(a);
a.click();
URL.revokeObjectURL(url);
document.body.removeChild(a);
}

// Parameters → update audio graph and local paramValues
function updateParam(id, v) {
paramValues[id] = v;

if (id === "gain" && masterGain) masterGain.gain.value = v;
else if (id === "lowpass" && lowpassFilter) lowpassFilter.frequency.value = v;
else if (id === "highpass" && highpassFilter) highpassFilter.frequency.value = v;
else if (id === "delayTime" && delayNode) delayNode.delayTime.value = v;
else if (id === "reverbMix" && dryGain && wetGain) {
wetGain.gain.value = v;
dryGain.gain.value = 1 - v;
}
}

// Update knob value label
function updateValLabel(id, v) {
const el = document.getElementById(id + "Val");
if (!el) return;
if (id === "lowpass" || id === "highpass") el.textContent = Math.round(v);
else el.textContent = v.toFixed(2);
}

// Knob behavior
const knobElems = document.querySelectorAll(".knob");
```

knobElems.forEach(knob ⇒ {
const id = knob.dataset.target;
const min = +knob.dataset.min;
const max = +knob.dataset.max;
const step = +knob.dataset.step || 0.01;

let value = paramValues[id];
let angle = valueToAngle(value, min, max);
let dragging = false;
let startY, startAngle;

knob.style.transform = `rotate(${angle}deg)`;

knob.addEventListener("mousedown", e ⇒ {
dragging = true;
startY = e.clientY;
startAngle = angle;
document.body.style.userSelect = "none";
});

window.addEventListener("mouseup", () ⇒ {
dragging = false;
document.body.style.userSelect = "";
});

window.addEventListener("mousemove", e ⇒ {
if (!dragging) return;
const dy = startY - e.clientY;
angle = clamp(startAngle + dy * 0.7, -135, 135);
knob.style.transform = `rotate(${angle}deg)`;

```
  const t = (angle + 135) / 270;
  const raw = lerp(min, max, t);
  const v = Math.round(raw / step) * step;

  value = v;
  updateParam(id, v);
  updateValLabel(id, v);
```

});

updateValLabel(id, value);
updateParam(id, value);
});

// Application of the presets
function applyPreset(name) {
initAudioGraph();
const preset = presetsConfig[name];

if (!preset) return;
const p = preset.params;

const set = (id, val) => {
paramValues[id] = val;
updateParam(id, val);
updateValLabel(id, val);

```
const k = document.querySelector(`.knob[data-target="${id}"]`);
if (k) {
  const min = +k.dataset.min;
  const max = +k.dataset.max;
  k.style.transform = `rotate(${valueToAngle(val, min, max)}deg)`;
}
```

};

set("gain", p.gain);
set("lowpass", p.lowpass);
set("highpass", p.highpass);
set("delayTime", p.delayTime);
set("reverbMix", p.reverbMix);
set("pitch", p.pitch);
}

// Stop waveform animation before leaving the page
window.addEventListener("beforeunload", () => {
if (animationId) cancelAnimationFrame(animationId);
});

Questo codice implementa una piccola DAW “one-track” in browser: registra dal microfono, riproduce l’audio, applica effetti in tempo reale (filtri, delay, riverbero, pitch) e permette di scaricare sia il segnale pulito sia quello processato in formato WAV. [1] [2]

Di seguito una spiegazione dettagliata, organizzata per sezioni, come in un documento di descrizione tecnica.

Struttura HTML

L’HTML definisce lo **scheletro** dell’applicazione: aree per waveform, controlli di registrazione, knob degli effetti, preset e pulsanti di download.

- `<script src="https://cdn.jsdelivr.net/.../audiobuffer-to-wav@1.0.0/index.min.js"></script>` carica la libreria `audiobuffer-to-wav`, che espone la funzione globale `audioBufferToWav(buffer, options?)` per convertire un `AudioBuffer` Web Audio in un ArrayBuffer WAV (16-bit PCM di default). [3] [4]
- `<canvas id="waveform"...>` è il canvas dove si disegna in tempo reale la forma d’onda usando un `AnalyserNode`. [5]

- Il blocco con `btnStartRec`, `btnStopRec` e `status` realizza l'interfaccia di registrazione microfono.
- L'elemento `<audio id="player" controls>` è il player HTML5 che riproduce il file registrato (versione "dry", senza effetti).
- `btnPlayProcessed` permette di riprodurre lo stesso audio ma attraverso la catena di effetti Web Audio.
- `knobsContainer` è un contenitore vuoto in cui i knob vengono creati dinamicamente via JavaScript, a partire da `knobsConfig`.
- `presetsContainer` ospita i pulsanti generati dinamicamente per richiamare configurazioni predefinite di parametri audio.
- `btnDownloadWav` e `btnDownloadProcessedWav` scaricano rispettivamente il file registrato "microfono puro" e quello processato (renderizzato offline con gli effetti).

Stili CSS

Il CSS definisce la **presentazione** dell'interfaccia: layout, estetica dei pulsanti, dei knob e del canvas.

- Il `body` ha font di sistema, sfondo scuro, centratura orizzontale e padding; questo crea un look "audio tool" moderno.
- `.app` è il contenitore principale, con larghezza massima, bordo arrotondato, bordo più chiaro e box-shadow per effetto "card".
- `.section` separa verticalmente le varie aree con un bordo superiore e margine inferiore; `.waveform-row` rimuove quel bordo nell'area superiore per un effetto più pulito.
- `.rec-row` usa `display:flex` per disporre in orizzontale blocco rec/stop e blocco player/processed, con allineamento centrale e gap.
- `button` definisce lo stile di default dei pulsanti: sfondo blu, testo bianco, bordo arrotondato, `cursor:pointer`; `button:disabled` ne cambia colore e cursore per comunicare lo stato inattivo.
- `.btn-round-rec` e `.btn-square-stop` creano i pulsanti di rec e stop con forme dedicate (cerchio e quadrato), usando pseudo-elementi `::after` per il simbolo interno chiaro, dando un feedback visivo immediato sul ruolo.
- `.knobs-grid` usa `display:grid` con tre colonne per disporre i knob in una griglia uniforme e centrata.
- `.knob-wrapper` è un piccolo pannello rettangolare che incapsula singolo knob, label e valore, con bordo e padding.
- `.knob` è un cerchio con gradiente radiale e `box-shadow` a imitare una manopola fisica; è inizialmente ruotato a -135°, in modo che la corsa di 270° (da -135° a +135°) rappresenti l'intero range del parametro.
- `.knob::after` disegna il "cursore" (indicator line) per mostrare la posizione angolare.
- `.preset-row` è una griglia a 4 colonne per i pulsanti preset, con ogni `.preset` in verde per evidenziare che sono "scelte rapide" più che comandi distruttivi.

- `.download-row` centra i pulsanti di download e impone una larghezza minima per aspetto uniforme.
- `canvas#waveform` imposta sfondo nero, bordo e border-radius per il riquadro della forma d'onda.

Stato globale e configurazione

La sezione iniziale JS definisce lo **stato applicativo** e le configurazioni per knob, preset e valori correnti.[5] [1]

- Variabili per la registrazione:
  - `mediaRecorder`: istanza di `MediaRecorder`, usata per registrare il flusso proveniente da `getUserMedia`.[2] [6]
  - `recordedChunks`: array di chunk `Blob` generati durante la registrazione.
  - `audioBlob`, `audioUrl`: rispettivamente il blob finale e il relativo URL (via `URL.createObjectURL`) da assegnare al player.
- Variabili Web Audio:
  - `audioCtx`: `AudioContext` principale usato per playback e effetti.[7] [5]
  - `lowpassFilter`, `highpassFilter`: `BiquadFilterNode` di tipo lowpass/highpass, che limitano lo spettro (p. es. simulazione telefono con passa-basso + passa-alto).[7] [1]
  - `delayNode`, `delayFeedback`: `DelayNode` più loop di feedback tramite `GainNode`, per ottenere un echo ripetuto.[8] [5]
  - `reverbConvolver`: `ConvolverNode` che usa un impulse response sintetico per il riverbero "hall".[9] [1]
  - `dryGain`, `wetGain`: gain separati per miscelare segnale secco e riverberato (controllo "reverbMix").
  - `masterGain`: gain finale che implementa il parametro "gain/volume".
  - `analyser`, `dataArray`, `animationId`: `AnalyserNode` per lettura del segnale in dominio tempo, `Uint8Array` per buffer di campioni e id di `requestAnimationFrame` per l'animazione waveform.[5]
- `activeProcessedSource` tiene il riferimento alla sorgente attualmente in riproduzione nella catena di effetti, per poterla fermare ed evitare sovrapposizioni.
- `mediaElementSource` è la sorgente Web Audio collegata al `<audio>` HTML per visualizzarne la waveform.

Le configurazioni:

- `knobsConfig` descrive ogni knob: id di parametro, etichetta, min/max, step e valore iniziale.
- `presetsConfig` definisce preset nominali ("Clean", "Phone", "Hall", "Lo-Fi") con set di parametri coerenti (es. "Phone" filtra bande alte e basse per ottenere un suono telefonico).
- `paramValues` è l'oggetto che contiene i valori runtime correnti dei parametri, aggiornato sia dai knob sia dai preset.

Creazione dinamica di knob e preset

Questa parte genera **dinamicamente** l'interfaccia dei controlli a partire dalle configurazioni.

Funzioni di utilità

- `lerp(a, b, t)` interpola linearmente tra `a` e `b` con fattore `t` (0→a, 1→b).
- `clamp(v, min, max)` limita un valore in un intervallo.
- `valueToAngle(v, min, max)` mappa un valore [min, max] all'intervallo angolare [-135°, +135°] per la rotazione grafica del knob.

`createKnobs()`

- Per ogni voce di `knobsConfig`:
  - Crea `div.knob-wrapper`.
  - Crea `div.knob` e imposta data-attribute `data-target` (id parametro), `data-min`, `data-max`, `data-step`.
  - Crea `div.knob-label` con il testo dell'etichetta ("Volume", "Pitch"...).
  - Crea `div.knob-value` con id `idVal` (es. `gainVal`) per mostrare il valore numerico; per lowpass/highpass viene arrotondato a intero, per gli altri con `toFixed(2)`.
  - Appende tutto al `knobsContainer`.

`createPresets()`

- Itera su `presetsConfig`:
  - Per ogni preset crea un `button.preset`, imposta `data-preset` e `textContent` con la label.
  - Aggancia un listener `click` che chiama `applyPreset(name)` per applicare i parametri relativi.
  - Appende al `presetsContainer`.

`createKnobs(); createPresets();` vengono chiamate subito, dunque l'interfaccia è pronta appena eseguito lo script.

Inizializzazione della catena audio

`initAudioGraph()` costruisce la **rete di nodi Web Audio** e la prepara per l'uso sia durante la riproduzione sia per il disegno della waveform.[7] [5]

- Se `audioCtx` esiste già, esce: la catena viene creata una sola volta.
- Crea `audioCtx = new AudioContext()` (o `webkitAudioContext` per compatibilità).
- `lowpassFilter = audioCtx.createBiquadFilter()` con `type="lowpass"` e `frequency` iniziale da `paramValues.lowpass`.

- `highpassFilter = audioCtx.createBiquadFilter()` con `type="highpass"` e `frequency` da `paramValues.highpass`.
- `delayNode = audioCtx.createDelay(5.0)` con `delayTime` iniziale da `paramValues.delayTime` (max 5 secondi).[8]
- `delayFeedback = audioCtx.createGain()` impostato a 0.3 e collegato in feedback: `delayNode → delayFeedback → delayNode` per ripetizioni.
- `reverbConvolver = audioCtx.createConvolver()` con `buffer` ottenuto da `createReverbImpulse(audioCtx, 2.5, 2.0)`, che sintetizza un impulse response stereo di durata 2.5 s con decadimento esponenziale.[9] [1]
- `dryGain`, `wetGain`, `masterGain` creati con `createGain()`, con `masterGain.gain` impostato a `paramValues.gain`.
- Calcolo `mix = paramValues.reverbMix`: `dryGain.gain = 1 - mix`, `wetGain.gain = mix` per controllare il rapporto dry/wet.
- `analyser = audioCtx.createAnalyser()` con `fftSize=2048` e `dataArray = new Uint8Array(analyser.fftSize)`.

Connessioni della catena (per sorgenti “processed”):

- `lowpassFilter → highpassFilter → delayNode`.
- Dal `delayNode`:
  - `delayNode → dryGain` (segnale diretto).
  - `delayNode → reverbConvolver → wetGain` (segnale riverberato).
- `dryGain` e `wetGain` confluiscono in `masterGain`.
- `masterGain → analyser → audioCtx.destination` per permettere sia ascolto sia visualizzazione waveform.

`connectPlayerToAnalyser()` serve solo a collegare `player` (segnale “dry” HTML5) al `analyser`, così che la waveform mostri la forma d’onda della riproduzione da `<audio>`:

- Se `audioCtx` e `player` esistono e `mediaElementSource` non è stato creato:
  - `mediaElementSource = audioCtx.createMediaElementSource(player);`
  - `mediaElementSource.connect(analyser);`

Riverbero: `createReverbImpulse`

Questa funzione genera **offline** un impulse response sintetico per il convolver.

- Usa `context.sampleRate` per calcolare `length = rate * duration`.
- `createBuffer(2, length, rate)` crea un buffer stereo della durata desiderata.
- Per ciascun canale:
  - `chData[i] = (Math.random() * 2 - 1) * Math.pow((length - i)/length, decay);`

    - Questo crea rumore bianco con ampiezza modulata da una funzione $(n)^{decay}$ decrescente, simulando un decadimento naturale di riverbero.
- Il buffer ottenuto viene assegnato come `convolver.buffer`.

Waveform: lettura e disegno

`drawWaveform()` aggiorna continuamente il canvas per visualizzare la forma d'onda del segnale passato all'`AnalyserNode`.[5]

- Se `analyser` non esiste, ritorna.
- Imposta `animationId = requestAnimationFrame(drawWaveform)` per avere un loop continuo.
- Preleva dimensioni del canvas (`w`, `h`) e chiama `analyser.getByteTimeDomainData(dataArray)`, che riempie `dataArray` con valori 8-bit (0-255) rappresentanti il segnale nel tempo.
- Pulisce il canvas (`fillRect` nero) e imposta stile linea ciano.
- Calcola `slice = w / dataArray.length` per mappare ogni campione a una posizione x.
- Per ogni campione, normalizza `dataArray[i]/128.0`, scala a metà altezza e disegna una polilinea attraverso tutti i punti, poi chiude verso il centro verticale.

Eventi del player HTML5

Questa parte sincronizza il player con la catena audio e la waveform.

- On `player.play`:
    - Chiama `initAudioGraph()` (se non già fatto).
    - Chiama `connectPlayerToAnalyser()` per connettere l'HTMLAudio all'`AnalyserNode`.
    - Avvia `drawWaveform()` se non è già in esecuzione.
    - Se esiste `activeProcessedSource`, chiama `stop()` su di esso per evitare che playback "processed" e "raw" si sovrappongano.
    - Riabilita `btnPlayProcessed` (si sta riproducendo la versione dry).
- On `player.ended`:
    - Se c'è un `animationId` attivo, chiama `cancelAnimationFrame` e azzera `animationId` per fermare l'animazione.

Registrazione microfono

Questa parte usa **MediaCapture** e **MediaRecorder** per acquisire l'audio del microfono.[6] [2]

Avvio registrazione: `btnStartRec.click`

- `getUserMedia({ audio: true })` chiede permesso e stream audio.
- Crea `mediaRecorder = new MediaRecorder(stream)` con il stream audio come input.[6]
- In `mediaRecorder.ondataavailable`:
  - I chunk vengono pushati in `recordedChunks` ogni volta che c'è nuovo dato.
- In `mediaRecorder.onstop`:
  - Crea `audioBlob = new Blob(recordedChunks, { type: "audio/webm" });`
  - Se esiste `audioUrl` precedente, lo revoca con `URL.revokeObjectURL`.
  - `audioUrl = URL.createObjectURL(audioBlob);`
  - `player.src = audioUrl;` collega l'audio registrato al player HTML5.
  - Abilita `btnPlayProcessed`, `btnDownloadWav`, `btnDownloadProcessedWav` e aggiorna `statusEl` a "The Record Is Ready".
- Subito dopo:
  - `mediaRecorder.start();` avvia la registrazione.
  - Disabilita `btnStartRec` e abilita `btnStopRec`.
  - Aggiorna lo status a "Recording...".
- In caso di eccezione:
  - Logga su console e imposta status a "Error: The Mic Is On".

Stop registrazione: `btnStopRec.click`

- Se `mediaRecorder` esiste e `state === "recording"`, chiama `mediaRecorder.stop()` e ferma tutte le tracce audio (`mediaRecorder.stream.getTracks().forEach(t => t.stop())`), rilasciando il microfono.
- Riabilita il pulsante start e disabilita lo stop.

Download WAV "raw"

`btnDownloadWav.click` permette di scaricare la registrazione **senza effetti** ma convertita in WAV tramite `audiobuffer-to-wav`.[4] [3]

- Se `audioBlob` non esiste, esce.
- Chiama `initAudioGraph()` per assicurarsi di avere un `AudioContext`.
- Con `audioBlob.arrayBuffer()` ottiene un `ArrayBuffer` dal Blob.
- `audioCtx.decodeAudioData(arrayBuffer)` decodifica il WebM in `AudioBuffer` Web Audio.[9]
- `audioBufferToWav(audioBuffer)` converte l'`AudioBuffer` in un ArrayBuffer con codifica WAV (16-bit PCM per default).[4]
- Crea `wavBlob = new Blob([wavBuffer], { type: "audio/wav" });` e usa `downloadBlob(wavBlob, "Recording.wav");` per forzare il download.

Playback con effetti

`btnPlayProcessed.click` riproduce la stessa registrazione ma **attraverso la catena effetti**.

- Se `audioBlob` non esiste, esce.
- `initAudioGraph()` assicura l'esistenza della catena.
- Se il player HTML `<audio>` sta suonando (`!player.paused`):
  - Chiama `player.pause()` e resetta `player.currentTime = 0;` per evitare sovrapposizioni con il processed playback.
- Se esiste un `activeProcessedSource` precedente:
  - Tenta `activeProcessedSource.stop()` nel blocco try/catch, poi azzera la variabile.
- Converte di nuovo `audioBlob` in `ArrayBuffer` e `AudioBuffer` con `decodeAudioData`.
- Crea `source = audioCtx.createBufferSource();`
  - `source.buffer = audioBuffer;`
  - `source.playbackRate.value = paramValues.pitch;` per implementare variazione di pitch tramite playback speed.
  - `source.connect(lowpassFilter);` inietta questa sorgente nella catena effetti creata in `initAudioGraph`.
- Imposta `activeProcessedSource = source` e disabilita `btnPlayProcessed` per evitare click multipli durante il playback.
- `source.onended`:
  - Se `activeProcessedSource === source`, azzera questa variabile.
  - Riabilita `btnPlayProcessed`.
- `source.start();` avvia il playback.
- Se la waveform non è in esecuzione (`!animationId`), chiama `drawWaveform()` per visualizzare il segnale in uscita da `masterGain` verso `analyser`.

Download WAV con effetti (render offline)

`btnDownloadProcessedWav.click` rende possibile scaricare un file WAV **con gli effetti applicati**, usando `OfflineAudioContext`.[1] [5]

1. Se `audioBlob` non esiste, esce.
2. Converte `audioBlob` in `ArrayBuffer` e crea un `AudioContext` "di probe" (`probeCtx`) solo per decodificare e leggere `duration` e `sampleRate` dell'audio, poi chiude `probeCtx`.
3. Calcola `length = Math.ceil(duration * sampleRate)` e crea `offlineCtx = new OfflineAudioContext(1, length, sampleRate)`:
   - Un solo canale, lunghezza in frame sufficiente a contenere l'intera registrazione, stesso sample rate.

4. Crea `source = offlineCtx.createBufferSource()` e la inizializza con `decoded`.
5. Ricostruisce una catena effetti *equivalente* a quella di runtime:
    - `lp` e `hp` come BiquadFilter lowpass/highpass con `frequency = paramValues.lowpass/highpass`.
    - `del` come `createDelay(5.0)` con `delayTime = paramValues.delayTime`.
    - `fb` come `GainNode` impostato a 0.3 in feedback con `del`.
    - `conv` come convolver con `createReverbImpulse(offlineCtx, 2.5, 2.0)`.
    - `dry`, `wet`, `master` come gain nodes; `master.gain = paramValues.gain`, `dry.gain`/`wet.gain` calcolati da `paramValues.reverbMix`.
    - `source.playbackRate.value = paramValues.pitch`.
    - Connessioni: `source → lp → hp → del → dry & conv → wet; dry & wet → master → offlineCtx.destination`.
6. `source.start(0);` avvia la riproduzione nel contesto offline.
7. `const rendered = await offlineCtx.startRendering();` produce un `AudioBuffer` contenente il risultato della catena.
8. `audioBufferToWav(rendered)` converte in WAV; si crea `wavBlob` e si scarica con `downloadBlob(wavBlob, "Recording_with_effects.wav");`.
9. Aggiorna lo status a “WAV with Effects Is Ready”.

Funzioni di supporto: download e aggiornamento parametri

`downloadBlob(blob, filename)`

- Crea un oggetto URL via `URL.createObjectURL(blob)`.
- Crea un `<a>` nascosto, imposta `href` e `download`, lo aggiunge al DOM, invoca `click()` per avviare il download, poi pulisce revocando l’URL e rimuovendo il link.

`updateParam(id, v)`

- Aggiorna `paramValues[id] = v`.
- Se la catena audio è inizializzata:
    - `gain`: aggiorna `masterGain.gain.value`.
    - `lowpass`: aggiorna `lowpassFilter.frequency.value`.
    - `highpass`: aggiorna `highpassFilter.frequency.value`.
    - `delayTime`: aggiorna `delayNode.delayTime.value`.
    - `reverbMix`: aggiorna `wetGain.gain` e `dryGain.gain` come `v` e `1-v`.
- In questo modo i knob agiscono *live* anche durante il playback.

`updateValLabel(id, v)`

- Trova l'elemento `id + "Val"` (es. `gainVal`).
- Per `lowpass`/`highpass` mostra `Math.round(v)`, per gli altri `v.toFixed(2)`.
- Mantiene la UI allineata con i valori effettivi.

Comportamento dei knob (interazione mouse)

La porzione successiva gestisce il **drag** dei knob e traduce il movimento verticale del mouse in variazione di parametro.

- Selezione: `const knobElems = document.querySelectorAll(".knob");` recupera tutti i knob creati dinamicamente.
- Per ogni `knob`:
  - Legge `id`, `min`, `max`, `step` dagli attributi dataset.
  - Inizializza `value = paramValues[id]` e calcola `angle = valueToAngle(value, min, max)`.
  - Variabili locali: `dragging`, `startY`, `startAngle`.
  - Imposta la trasformazione CSS iniziale `knob.style.transform = rotate(angle deg)`.
  - `mousedown`:
    - `dragging = true`, memorizza `startY` (posizione verticale iniziale del mouse) e `startAngle`.
    - Imposta `document.body.style.userSelect = "none"` per evitare selezioni di testo durante il drag.
  - `window.mouseup`:
    - `dragging = false` e ripristina `userSelect`.
  - `window.mousemove`:
    - Se non è in drag, ritorna.
    - Calcola `dy = startY - e.clientY` (spostamento verso l'alto → valore positivo).
    - Aggiorna `angle = clamp(startAngle + dy * 0.7, -135, 135)`.
    - Aggiorna `knob.style.transform` con il nuovo angolo.
    - Calcola `t = (angle + 135) / 270` come posizione relativa 0–1.
    - `raw = lerp(min, max, t)` e `v = Math.round(raw / step) * step` per quantizzare al passo desiderato.
    - Aggiorna `value`, chiama `updateParam(id, v)` per aggiornare catena audio e `updateValLabel(id, v)` per aggiornare display numerico.
  - Alla fine:
    - Chiama `updateValLabel(id, value)` e `updateParam(id, value)` per assicurare che lo stato iniziale dei parametri sia applicato alla catena quando viene creata.

Applicazione dei preset

`applyPreset(name)` permette di impostare rapidamente i parametri a valori predefiniti coerenti.

- Chiama `initAudioGraph()` per assicurarsi che i nodi audio siano disponibili.
- Recupera `preset = presetsConfig[name]`; se non esiste, esce.
- Definisce funzione interna `set(id, val)`:
  - Aggiorna `paramValues[id]`, `updateParam(id, val)` e `updateValLabel(id, val)`.
  - Trova il knob corrispondente via `document.querySelector('.knob[data-target="id"]')`.
  - Calcola l'angolo corretto con `valueToAngle(val, min, max)` e aggiorna `k.style.transform`.
- Chiama `set` per `gain`, `lowpass`, `highpass`, `delayTime`, `reverbMix`, `pitch` usando `preset.params`.
- Effetto: tutti i knob (grafica + valori + catena audio) scattano immediatamente al preset scelto.

Pulizia finale: beforeunload

- `window.addEventListener("beforeunload", ...)` cancella `requestAnimationFrame(animationId)` se attivo, evitando lasciare animazioni pendenti quando la pagina viene chiusa o ricaricata.

Se vuoi, nella risposta successiva si può aggiungere una sezione specifica su "come mappare questo design in una documentazione PDF" (schema moduli, diagramma dei nodi Web Audio, pseudo-codice per i flussi) per facilitare la scrittura della relazione.

⁂

non puoi crearmelo tu?

Sì, certo: posso crearti io la "bozza" di contenuto per il PDF, strutturata già come documentazione tecnica (se vuoi poi la incolli in LaTeX/Word).

Di seguito trovi un testo pronto da copiare, con sezioni e sottosezioni, pensato proprio come descrizione del codice.

1. Panoramica dell'applicazione

Questa applicazione Web realizza un piccolo registratore con effetti audio in tempo reale, basato su HTML5, CSS e Web Audio API. Consente di registrare dal microfono, riprodurre l'audio registrato, applicare una catena di effetti (filtri, delay, riverbero, pitch-shift) e scaricare l'audio sia in versione non processata (raw) sia con effetti in formato WAV grazie alla libreria `audiobuffer-to-wav`.[21] [22] [23] [24]

L'interfaccia utente è composta da:

- Un canvas per la visualizzazione della waveform.
- Pulsanti per avviare/fermare la registrazione e per controllare la riproduzione.
- Manopole (knob) per controllare i parametri degli effetti.
- Preset di effetti preconfigurati.
- Pulsanti di download per i file WAV.

2. Struttura HTML

2.1 Inclusione libreria di conversione WAV

Lo `<script>` iniziale include la libreria `audiobuffer-to-wav` da CDN:

- L'URL punta alla versione minificata su jsDelivr.[25]
- La libreria espone una funzione che accetta un `AudioBuffer` Web Audio e restituisce un ArrayBuffer codificato in formato WAV (16-bit PCM di default).[22] [23]

Questa funzione viene utilizzata sia per il download del file "raw" sia per quello processato.

2.2 Contenitore principale `.app`

L'elemento `<div class="app">` incapsula l'intera UI:

- Definisce la larghezza massima del layout.
- Contiene tutte le sezioni: waveform, controlli di registrazione, knob, preset, pulsanti di download.

Ogni sezione è organizzata tramite `<div class="section ...">` con classi specifiche per il tipo di contenuto.

2.3 Waveform

La sezione `waveform-row` contiene:

```
- `<canvas id="waveform" width="560" height="120"></canvas>`
```

- Questo canvas viene utilizzato dal codice JavaScript per disegnare la forma d'onda del segnale audio usando un `AnalyserNode`.[21]
- Le dimensioni in pixel vengono interpretate dal contesto 2D (wfCtx) per il rendering.

2.4 Controlli di registrazione e playback

La sezione `rec-row` contiene due blocchi:

1. Blocco registrazione:
    - `button#btnStartRec.btn-round-rec`: pulsante circolare per avviare la registrazione dal microfono.

- `button#btnStopRec.btn-square-stop`: pulsante quadrato per fermare la registrazione; inizialmente disabilitato.
- `span#status`: piccolo indicatore testuale dello stato (es. “Ready”, “Recording…”, “The Record Is Ready”).

2. Blocco riproduzione:
   - `<audio id="player" controls>`: elemento HTML5 che riproduce il file audio registrato in forma “dry”, senza catena di effetti Web Audio.
   - `button#btnPlayProcessed`: pulsante che avvia la riproduzione dell’audio registrato attraverso la catena di effetti; inizialmente disabilitato.

2.5 Container per knob e preset

- `div#knobsContainer.knobs-grid`: inizialmente vuoto, viene popolato dinamicamente via JS usando `knobsConfig`, creando per ogni parametro una manopola grafica con label e valore numerico.
- `div#presetsContainer.preset-row`: anch’esso vuoto in HTML, viene popolato con pulsanti preset generati a partire da `presetsConfig`.

2.6 Pulsanti di download

La `download-row` contiene:

- `button#btnDownloadWav`: consente il download del file registrato (microfono) in formato WAV, senza effetti.
- `button#btnDownloadProcessedWav`: consente il download del file processato con la catena effetti, anch’esso in WAV.

Entrambi i pulsanti sono inizialmente disabilitati e vengono attivati solo dopo che una registrazione è stata completata.

3. Stili CSS principali

3.1 Layout generale

- Il `body` imposta:
  - Font di sistema.
  - Sfondo scuro.
  - Layout con `display:flex` e `justify-content:center` per centrare l’app.
- `.app` definisce:
  - Larghezza massima a 780px.
  - Sfondo, bordi arrotondati, bordo più chiaro e ombra per un aspetto “card” moderno.

Ogni `.section` aggiunge il separatore visivo (border-top) e il padding verticale, tranne la `waveform-row` che rimuove il bordo superiore per allinearsi all’inizio dell’app.

3.2 Controlli di registrazione

- `.rec-row` utilizza `display:flex` con `justify-content:space-between` per disporre il blocco di registrazione e quello di playback su una sola riga.
- `.block` è un contenitore flessibile per elementi ravvicinati (pulsanti + testo), con wrapping e gap.

I pulsanti generici:

- `button` definisce colori, padding, bordi arrotondati, e `cursor:pointer`.
- `button:disabled` scurisce il pulsante e cambia il cursore a `not-allowed` per comunicare lo stato disabilitato.

I pulsanti rec/stop:

- `.btn-round-rec`: pulsante circolare rosso; pseudo-elemento `::after` disegna un cerchio interno chiaro che richiama il classico "rec".
- `.btn-square-stop`: pulsante quadrato rosso; `::after` disegna un quadrato chiaro interno come simbolo di stop.

3.3 Manopole e preset

- `.knobs-grid` definisce una griglia a 3 colonne per i knob, centrati e con spaziatura costante.
- `.knob-wrapper` crea un riquadro per ogni parametro, con bordo, padding e allineamento verticale (knob, label, valore).

Il singolo knob:

- `.knob` è un cerchio con gradiente radiale per imitare la tridimensionalità di una manopola fisica.
- `transform: rotate(-135deg)` imposta la posizione angolare minima (0%) a -135°, lasciando una corsa totale di 270°.
- `.knob::after` disegna la "tacca" (indicatore) che mostra l'orientamento della manopola.
- `.knob:active` modifica l'ombra per dare feedback di pressione.

Preset:

- `.preset-row` usa una griglia a 4 colonne per i pulsanti preset.
- `.preset` definisce pulsanti verdi, a larghezza piena, per indicare che sono configurazioni "positive" e veloci da provare.

4. Logica JavaScript: stato e configurazione

4.1 Stato della registrazione

Le variabili globali:

- `mediaRecorder` è l'istanza di `MediaRecorder` associata allo stream del microfono.[26]
- `recordedChunks` è l'array di chunk audio prodotti durante la registrazione.
- `audioBlob` è il Blob finale della registrazione (tipicamente `audio/webm`).
- `audioUrl` è l'URL oggetto di `audioBlob`, assegnato al `src` del player.

4.2 Nodi Web Audio e visualizzazione

Variabili per la catena audio:

- `audioCtx` è l'`AudioContext` principale.[21]
- `lowpassFilter` e `highpassFilter` sono BiquadFilterNodes `lowpass` e `highpass`, che implementano il controllo delle bande di frequenza.[27]
- `delayNode` e `delayFeedback` realizzano un delay con feedback.
- `reverbConvolver` è un `ConvolverNode` utilizzato per il riverbero hall.[24] [28]
- `dryGain` e `wetGain` sono i nodi di gain per il mix dry/wet.
- `masterGain` è il volume globale dell'uscita.
- `analyser` è un `AnalyserNode` per estrarre i dati di dominio tempo e disegnare la waveform.[21]
- `dataArray` è il buffer di byte usato per leggere i valori dal `analyser`.
- `animationId` è l'id del ciclo `requestAnimationFrame` per il disegno continuo.

`activeProcessedSource` tiene traccia della sorgente "processed" attualmente in playback, per poterla fermare prima di crearne una nuova.
`mediaElementSource` è la sorgente derivata dall'elemento `<audio>` HTML, usata per collegare il player all'`AnalyserNode`.

4.3 Configurazione knob e preset

- `knobsConfig` descrive ogni manopola: id parametro (gain, pitch, lowpass…), label, min, max, step, valore di default.
- `presetsConfig` contiene vari preset:
  - `clean`: quasi senza effetti, banda completa, leggero riverbero.
  - `phone`: banda limitata (telefonico), riverbero nullo, simula un parlato telefonico.
  - `hall`: riverbero marcato, leggero delay, filtri adatti a una sala ampia.
  - `lofi`: banda ridotta, delay e riverbero moderati, pitch leggermente abbassato (0.9) per effetto lo-fi.
- `paramValues` contiene lo stato corrente dei parametri; viene aggiornato dai knob e dai preset.

5. Costruzione dinamica della UI (knob e preset)

5.1 Funzioni di utilità

- `lerp(a, b, t)` calcola l'interpolazione lineare tra `a` e `b`.
- `clamp(v, min, max)` limita `v` all'intervallo [min, max].
- `valueToAngle(v, min, max)` mappa il valore del parametro `v` all'intervallo angolare [-135°, +135°] per l'orientamento del knob.

5.2 `createKnobs()`

Per ciascun elemento di `knobsConfig`:

- Crea un wrapper `.knob-wrapper`.
- Crea il `div.knob` con data-attributi per id, min, max, step.
- Crea `div.knob-label` con la label testuale.
- Crea `div.knob-value` con id `idVal` e valore iniziale formattato (arrotondato per low/high-pass, con due decimali per gli altri).
- Appende tutto a `knobsContainer`.

5.3 `createPresets()`

Per ogni preset in `presetsConfig`:

- Crea un `button.preset`.
- Imposta `data-preset` con il nome logico del preset.
- Imposta il testo con la label da mostrare.
- Aggiunge un listener `click` che invoca `applyPreset(name)`.

Alla fine, `createKnobs()` e `createPresets()` vengono chiamate, rendendo l'interfaccia operativa.

6. Inizializzazione della catena Web Audio

6.1 `initAudioGraph()`

Questa funzione crea e connette tutti i nodi audio se non esistono già:

- Crea `audioCtx` come `AudioContext`.
- Crea `lowpassFilter`, `highpassFilter`, `delayNode`, `delayFeedback`.
- Configura il feedback del delay: `delayNode → delayFeedback → delayNode`.
- Crea `reverbConvolver` e genera l'impulse response con `createReverbImpulse(audioCtx, 2.5, 2.0)`.[24]
- Crea i tre GainNode (`dryGain`, `wetGain`, `masterGain`) e imposta il valore iniziale di `masterGain.gain` da `paramValues.gain`.

- Imposta il mix dry/wet in base a `paramValues.reverbMix`.
- Crea l'`analyser`, definisce `fftSize=2048` e alloca `dataArray`.

Collega i nodi nella seguente catena:

- `lowpassFilter → highpassFilter → delayNode`.
- `delayNode → dryGain` e `delayNode → reverbConvolver → wetGain`.
- `dryGain` e `wetGain` confluiscono in `masterGain`.
- `masterGain → analyser → audioCtx.destination`.

6.2 `connectPlayerToAnalyser()`

- Se `mediaElementSource` non esiste:
  - Crea `audioCtx.createMediaElementSource(player)`.
  - Collega la sorgente direttamente all'`analyser` in parallelo alla catena degli effetti, così la waveform visualizza anche l'audio riprodotto dall'elemento `<audio>`.[21]

7. Riverbero e waveform

7.1 `createReverbImpulse(context, duration, decay)`

- Calcola la lunghezza del buffer come `sampleRate * duration`.
- Crea un buffer stereo (`createBuffer(2, length, rate)`).
- Per ogni canale e campione:
  - Calcola un fattore di decadimento $(length - i)/length$ elevato a `decay`.
  - Moltiplica un valore di rumore uniforme in [-1, +1] per questo fattore, producendo un rumore che decresce nel tempo.
- Restituisce il buffer, che viene usato per `reverbConvolver.buffer`.

7.2 `drawWaveform()`

- Usa `requestAnimationFrame(drawWaveform)` per aggiornare continuamente il canvas.
- Recupera le dimensioni del canvas.
- Chiama `analyser.getByteTimeDomainData(dataArray)` per ottenere il segnale nel dominio del tempo come valori 0-255.[21]
- Pulisce il canvas e imposta stile della linea.
- Calcola `slice = width / dataArray.length` per mappare ogni campione su una posizione x.
- Converte ogni valore in coordinata y e disegna una curva continua che rappresenta la waveform.

8. Gestione del player HTML e registrazione microfono

8.1 Eventi del player

- `player.play`:
  - Inizializza la catena audio.
  - Collega `player` all'`analyser`.
  - Avvia il disegno della waveform se non già attivo.
  - Ferma eventuale sorgente processed (`activeProcessedSource`) per evitare playback simultanei.
  - Riabilita il pulsante "Play with Effects".
- `player.ended`:
  - Se l'animazione è attiva, cancella `requestAnimationFrame` e azzera `animationId`.

8.2 Avvio registrazione: `btnStartRec.click`

- Richiede accesso al microfono con `navigator.mediaDevices.getUserMedia({ audio:true })`. [29] [26]
- Crea `mediaRecorder = new MediaRecorder(stream)`.[26]
- In `mediaRecorder.ondataavailable` accumula i chunk audio in `recordedChunks`.
- In `mediaRecorder.onstop`:
  - Crea `audioBlob` come `new Blob(recordedChunks, { type:"audio/webm" })`.
  - Crea `audioUrl` con `URL.createObjectURL(audioBlob)` e lo assegna al `src` del player.
  - Abilita pulsanti di playback processed e di download.
  - Aggiorna `statusEl` a "The Record Is Ready".
- Avvia `mediaRecorder.start()`, disabilita il pulsante di start, abilita quello di stop e aggiorna lo stato a "Recording...".

8.3 Stop registrazione: `btnStopRec.click`

- Se `mediaRecorder` è in stato `recording`, chiama `mediaRecorder.stop()` e ferma tutte le tracce dello stream per rilasciare il microfono.
- Riabilita il pulsante di avvio e disabilita quello di stop.

9. Funzionalità di download

9.1 Download WAV raw: `btnDownloadWav.click`

- Se `audioBlob` non esiste, non fa nulla.
- Inizializza l'audio context (se necessario).
- Converte `audioBlob` in `ArrayBuffer` e lo decodifica in `AudioBuffer` (Web Audio).
- Usa `audioBufferToWav(audioBuffer)` per ottenere un ArrayBuffer WAV.[30] [23] [22]
- Crea un Blob `audio/wav` e chiama `downloadBlob(wavBlob, "Recording.wav")` per forzare il download.

9.2 Download WAV processed: `btnDownloadProcessedWav.click`

Questa procedura usa `OfflineAudioContext` per renderizzare offline l'audio con la catena di effetti.

- Decodifica `audioBlob` in un `AudioBuffer` usando un `AudioContext` temporaneo.
- Calcola durata e sample rate; chiude il context temporaneo.
- Crea `offlineCtx = new OfflineAudioContext(1, length, sampleRate)` con:
  - 1 canale.
  - `length` pari al numero di frame necessari a contenere tutta la registrazione.
- Crea un `BufferSource`, lo collega a una catena di nodi che replica quella online:
  - BiquadFilter lowpass/highpass.
  - Delay con feedback.
  - Convolver con impulse response generata da `createReverbImpulse`.
  - Gain per mix dry/wet.
  - Master gain con `paramValues.gain`.
  - PlaybackRate per il pitch.
- Collega il master all'output dell'OfflineAudioContext.
- Avvia la sorgente e chiama `offlineCtx.startRendering()`, ottenendo un `AudioBuffer` con l'audio processato.[31] [21]
- Converte l'AudioBuffer in WAV tramite `audioBufferToWav` e scarica con `downloadBlob(...)`.
- Aggiorna lo stato a "WAV with Effects Is Ready".

10. Aggiornamento parametri e preset

10.1 `downloadBlob(blob, filename)`

- Crea un URL oggetto per il Blob.
- Crea un link `<a>` nascosto con attributo `download`.
- Simula un click per avviare il download.

- Revoca l'URL e rimuove l'elemento link.

10.2 `updateParam(id, v)` e `updateValLabel(id, v)`

- `updateParam`:
  - Aggiorna `paramValues[id]`.
  - Se i nodi audio sono già creati, aggiorna i campi corrispondenti del grafo:
    - `masterGain.gain` per `gain`.
    - `lowpassFilter.frequency` e `highpassFilter.frequency`.
    - `delayNode.delayTime`.
    - `dryGain`/`wetGain` per il mix riverbero.
- `updateValLabel`:
  - Aggiorna l'elemento `id + "Val"` con il nuovo valore formattato, mantenendo la UI coerente con lo stato interno.

10.3 Gestione interazione knob

Per ogni knob:

- Legge config (id, min, max, step) dai data-attributi.
- Inizializza l'angolo in base al valore corrente del parametro.
- Gestisce tre eventi:
  - `mousedown`: inizia il drag, registra posizione iniziale del mouse e angolo.
  - `mousemove` (su `window`): se in drag, calcola delta verticale, aggiorna l'angolo, ricava il valore relativo (interpolazione tra min e max), quantizza con `step` e chiama `updateParam` + `updateValLabel`.
  - `mouseup` (su `window`): termina il drag e ripristina la selezione del testo.
- All'inizializzazione chiama `updateParam` e `updateValLabel` per applicare i valori di default alla catena audio.

10.4 `applyPreset(name)`

- Inizializza il grafo audio.
- Recupera i parametri del preset.
- Per ogni parametro (`gain`, `lowpass`, `highpass`, `delayTime`, `reverbMix`, `pitch`):
  - Aggiorna `paramValues`.
  - Chiama `updateParam` e `updateValLabel`.
  - Calcola il nuovo angolo del knob con `valueToAngle` e aggiorna la trasformazione CSS del knob.

11. Pulizia alla chiusura pagina

- `beforeunload`:
  - Se l'animazione della waveform è attiva, chiama `cancelAnimationFrame(animationId)` per evitare cicli di disegno pendenti al momento dell'uscita dalla pagina.

Se vuoi, nel prossimo passo si può:

- Aggiungere uno schema tipo "diagramma a blocchi" della catena audio da inserire nel PDF.
- Oppure trasformare questa struttura direttamente in markup LaTeX (se stai usando LaTeX per generare il PDF).

⁂

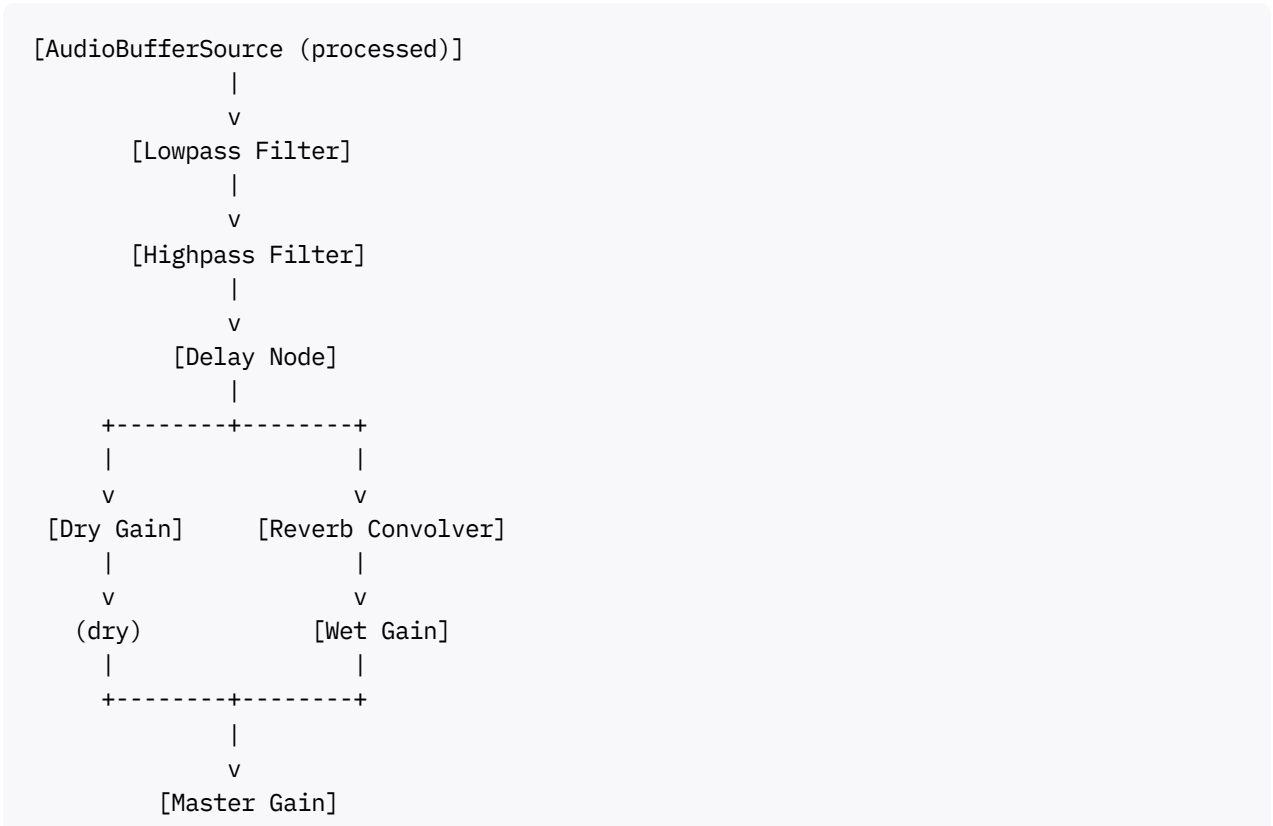
generami il diagramma a blocchi

Ti propongo un diagramma a blocchi testuale (facile da trasformare in figura con draw.io, PowerPoint, LaTeX TikZ, ecc.) che rappresenta:

- la catena **processed** (usata per "Play with Effects" e per il rendering offline),
- la catena **raw** (player HTML),
- il ramo di visualizzazione waveform (Analyser).

1. Diagramma a blocchi – catena effetti (runtime)

Percorso usato dal pulsante **Play with Effects** (catena definita in `initAudioGraph`):



```
              |
              v
          [Analyser]
              |
              v
        [AudioContext
          Destination]
```

Note principali (in linea con il modello di audio graph della Web Audio API).[48] [49]

- L'ingresso è un `AudioBufferSourceNode` con `playbackRate` controllato dal parametro **Pitch**.
- I filtri `Lowpass` e `Highpass` definiscono la banda passante.
- `Delay Node` ha un loop interno: in codice `delayNode → delayFeedback → delayNode`.
- L'uscita del delay viene sdoppiata: un ramo passa diretto nel `Dry Gain`, l'altro nel `Reverb Convolver` (riverbero convoutivo).[50] [51]
- `Dry Gain` e `Wet Gain` sono controllati dal parametro **ReverbMix**.
- `Master Gain` applica il parametro **Volume** (gain globale).
- `Analyser` serve per la visualizzazione waveform; da qui il segnale va al `destination` (uscita audio del dispositivo).[52] [49]

Se vuoi tradurlo in TikZ, basta mappare ogni blocco a un nodo rettangolare e le frecce a `\draw[->]`.

2. Diagramma a blocchi – catena effetti (OfflineAudioContext)

La catena per il rendering offline (download WAV con effetti) è concettualmente identica, ma costruita dentro un `OfflineAudioContext`:

```
[Decoded AudioBuffer]
         |
         v
 [BufferSource (offline)]
         |
         v
   [Lowpass Filter]
         |
         v
   [Highpass Filter]
         |
         v
      [Delay Node]
         |
   +-----+-----+
   |           |
   v           v
[Dry Gain] [Reverb Convolver]
   |           |
   v           v
 (dry)     [Wet Gain]
```

```
   |           |
   +-----+-----+
         |
         v
    [Master Gain]
         |
         v
 [OfflineAudioContext
     Destination]
          |
          v
  [Rendered AudioBuffer]
          |
          v
  [WAV encoder
 (audioBufferToWav)]
```

- L'ingresso è il `BufferSource` alimentato con l'`AudioBuffer` decodificato dal blob registrato.
- L'uscita dell'OfflineAudioContext (`Rendered AudioBuffer`) viene passato ad `audioBufferToWav` che produce il file WAV scaricabile.[53] [54] [55]

3. Diagramma a blocchi – percorso raw e visualizzazione

Percorsi quando usi il player HTML `<audio>` (senza effetti) e la waveform.

3.1 Playback raw (HTMLAudioElement)

```
[HTML <audio> element]
          |
    (audio hardware path
     gestito dal browser)
```

- L'elemento `<audio>` riproduce direttamente il file collegato via `src`.
- Il browser instrada il segnale alla scheda audio; la catena Web Audio non modifica questo segnale.

3.2 Visualizzazione waveform da player HTML

Quando il player suona, il codice chiama `createMediaElementSource(player)` e collega la sorgente all'`Analyser`:

```
[HTML <audio> element]
          |
          v
[MediaElementSourceNode]
          |
          v
      [Analyser]
          |
          v
```

```
    [Canvas renderer
     (drawWaveform)]
```

- Questo ramo esiste in parallelo alla catena effetti; serve solo a leggere i campioni per disegnarli, non per modificarne l'ascolto.[48] [52]

4. Diagramma a blocchi – pannello di controllo (knob/preset)

Per completare la documentazione, puoi includere anche un piccolo schema logico per la parte di controllo:

```
        +----------------+
        |  UI Controls   |
        | (Knobs,Preset) |
        +----------------+
           |        |
           |        |
           v        v
[Param Values]   [Knob Angles/
(gain, lp, hp,    Labels UI]
 delay, rev, pitch)]
          |
          v
[Update Audio Graph]
(updateParam(...):
 - masterGain.gain
 - lp.frequency
 - hp.frequency
 - delayTime
 - dry/wet gains
 - playbackRate)
```

- I knob modificano i `paramValues` e, tramite `updateParam`, aggiornano i corrispondenti `AudioParam` dei nodi Web Audio.[56] [49]
- I preset impostano in blocco i parametri e aggiornano grafica dei knob + stato del grafo.

Se vuoi, nel messaggio successivo posso darti direttamente il codice TikZ/LaTeX per il diagramma principale (catena effetti) pronto da incollare in un documento.

⁂

1. https://web.dev/patterns/media/audio-effects
2. https://web.dev/patterns/media/microphone-record
3. https://www.npmjs.com/package/audiobuffer-to-wav
4. https://app.unpkg.com/audiobuffer-to-wav@1.0.0/files/README.md
5. http://padenot.github.io/web-audio-demojs14-paris/web-audio-api-cheatsheet.html
6. https://developer.mozilla.org/en-US/docs/Web/API/MediaStream_Recording_API/Using_the_MediaStream_Recording_API

7. https://developer.mozilla.org/en-US/docs/Web/API/BaseAudioContext/createBiquadFilter
8. https://developer.mozilla.org/en-US/docs/Web/API/BaseAudioContext/createDelay
9. https://webdocs.dev/en-us/docs/web/api/baseaudiocontext/createbiquadfilter
10. https://www.npmjs.com/package/@types/audiobuffer-to-wav
11. https://stackoverflow.com/questions/43717841/how-to-convert-audiobuffer-to-wav-file
12. https://cdn.jsdelivr.net/npm/audiobuffer-to-wav/
13. https://24ways.org/2019/building-a-dictaphone-using-media-recorder-and-getusermedia/
14. https://snyk.io/advisor/npm-package/audiobuffer-to-wav
15. https://addpipe.com/getusermedia-examples/
16. https://www.npmjs.com/search?q=WAV
17. https://stackoverflow.com/questions/62348047/getusermedia-and-mediarecorder-in-html
18. https://github.com/cstoquer/audio-encoder
19. https://www.npmjs.com/search?q=keywords%3Awav+to+mp3
20. https://www.jsdelivr.com/package/npm/wav-file-encoder
21. https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API/Visualizations_with_Web_Audio_API
22. https://app.unpkg.com/audiobuffer-to-wav@1.0.0/files/README.md
23. https://www.npmjs.com/package/audiobuffer-to-wav
24. https://web.dev/patterns/media/audio-effects
25. https://cdn.jsdelivr.net/npm/audiobuffer-to-wav/
26. https://developer.mozilla.org/en-US/docs/Web/API/MediaStream_Recording_API/Using_the_MediaStream_Recording_API
27. https://developer.mozilla.org/en-US/docs/Web/API/BaseAudioContext/createBiquadFilter
28. https://webdocs.dev/en-us/docs/web/api/baseaudiocontext/createbiquadfilter
29. https://web.dev/patterns/media/microphone-record
30. https://stackoverflow.com/questions/43717841/how-to-convert-audiobuffer-to-wav-file
31. http://padenot.github.io/web-audio-demojs14-paris/web-audio-api-cheatsheet.html
32. https://stackoverflow.com/questions/14535829/web-audio-api-how-to-inspect-the-audio-graph
33. https://www.overleaf.com/learn/latex/CircuiTikz_package
34. https://tex2speech-website.vercel.app/docs.html
35. https://info.arxiv.org/help/texlive_package_list.html
36. https://www.docuwriter.ai/posts/technical-documentation-templates
37. https://www.reddit.com/r/LaTeX/comments/1ejr9s0/automated_pdf_generation_of_latex_files_via_api/
38. https://www.proprofskb.com/blog/technical-documentation-software/
39. https://github.com/notthetup/awesome-webaudio
40. https://wedocs.co/best-technical-documentation-tools/
41. https://project-archive.inf.ed.ac.uk/ug4/20244200/ug4_proj.pdf
42. https://kodesage.ai/blog/9-best-software-documentation-tools
43. https://forum.juce.com/t/method-for-writing-audiobuffer-to-a-wav-file/61187

44. https://www.reddit.com/r/ControlTheory/comments/nvyzhw/best_latex_packages_for_control_theory_diagrams/
45. https://document360.com/blog/software-documentation-tools/
46. https://codesandbox.io/examples/package/audiobuffer-to-wav
47. https://www.atlassian.com/blog/loom/software-documentation-tools
48. https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API/Using_Web_Audio_API
49. https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API
50. https://blog.mi.hdm-stuttgart.de/index.php/2021/02/24/web-audio-api-tips-for-performance/
51. https://www.briankoponen.com/web-audio-introductory-tutorial/
52. https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API/Visualizations_with_Web_Audio_API
53. https://www.npmjs.com/package/audiobuffer-to-wav
54. https://stackoverflow.com/questions/43717841/how-to-convert-audiobuffer-to-wav-file
55. http://padenot.github.io/web-audio-demojs14-paris/web-audio-api-cheatsheet.html
56. https://web.dev/articles/webaudio-intro
57. https://stackoverflow.com/questions/14535829/web-audio-api-how-to-inspect-the-audio-graph
58. https://www.w3.org/TR/webaudio-1.1/
59. https://www.telerik.com/blogs/adding-audio-visualization-react-app-using-web-audio-api
60. https://www.dsprelated.com/freebooks/pasp/System_Block_Diagram.html
61. https://padenot.github.io/web-audio-scotlandjs15/
62. https://syntherjack.net/synth-block-diagram-guidelines/
63. https://www.nickwritesablog.com/sound-design-in-web-audio-neurofunk-bass-part-1/
64. https://www.ti.com/lit/pdf/spry298
65. https://webaudioapi.com/samples/
66. https://arxiv.org/pdf/2010.10291.pdf
67. https://stackoverflow.com/questions/34482319/web-audio-api-how-do-i-add-a-working-convolver
68. https://udn.realityripple.com/docs/Web/API/Web_Audio_API/Visualizations_with_Web_Audio_API
69. https://sarti.faculty.polimi.it/Augusto_Sarti/CV_and_publications_files/2007_IEEE_SPMag_object-based_sound_synthesis.pdf
70. https://github.com/CaesuraCo/web-audio-graph