Ora ti fornirò un codice formato da tre parti (HTML, CSS, Javascript) che svolge le funzionalità di un recorder web con knobs, presets, funzionalità di download e collegamento ad un drive.

HTML

``` ``` ``` ``` Ready
Play with Effects
```
``` ```
```
Download WAV Mic Download WAV with Effects
Autorizza Google Drive Carica WAV Mic su Drive Carica WAV Effects su Drive Drive: non autorizzato

CSS
```
body {
font-family: system-ui, sans-serif;
background: #f6f1e7;
color: #111827;
display: flex;
justify-content: center;
padding: 16px;
margin: 0;
}

.app {
max-width: 780px;
width: 100%;
background: #fbf7f0;
border-radius: 12px;
padding: 16px;
box-shadow: 0 18px 30px rgba(0,0,0,0.25);
border: 1px solid #e5dccb;
}

.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;
}

.muted {
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: #f3ede2; / panna scuro */
border-radius: 10px;
padding: 10px 4px;
```

```
border: 1px solid #e0d7c6;
width: 100%;
max-width: 140px;
}

.knob {
width: 60px;
height: 60px;
border-radius: 50%;
background: radial-gradient(
circle at 30% 30%,
#ffffff,
#d9cfbf
); /* avorio */
position: relative;
cursor: pointer;
transform: rotate(-135deg);
transition: box-shadow 0.1s ease;
box-shadow:
0 0 0 1px #cfc5b3,
0 6px 14px rgba(0,0,0,0.25);
}

.knob::after {
content: "";
position: absolute;
width: 4px;
height: 22px;
background: #6b5f4a; /* marrone caldo */
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;
}

/* Drive buttons */
.drive-row {
display: flex;
justify-content: center;
gap: 10px;
align-items: center;
flex-wrap: wrap;
}
.drive-row button {
min-width: 180px;
}

/* Waveform Canva */
canvas#waveform {
width: 100%;
background: #000;
border-radius: 6px;
border: 1px solid #111827;
}
.logo-container {
display: flex;
justify-content: center;
```

```
margin-bottom: 10px;
}

.logo {
height: 450px; /* piccolo */
width: auto;
opacity: 1.0;
}
.waveform-wrapper{
display: flex;
align-items: center;
gap: 24px;
}

.logo-side{
display: flex;
align-items: center;
justify-content: center;
}

.logo{
height: 250px; /* grande */
width: auto;
transform: translateY(-23px);
}

canvas#waveform{
flex: 1; /* la waveform prende lo spazio che resta */
}

.waveform-container {
position: relative;
width: 100%;
}

.logo-overlay {
position: absolute;
top: -60px; /* regola questa per alzare/abbassare /
left: 50%;
transform: translateX(-50%);
height: 220px; / dimensione logo /
pointer-events: none; / non blocca click sul canvas */
z-index: 10;
}

canvas#waveform {
width: 100%;
background: #000;
```

```
border-radius: 6px;
border: 1px solid #111827;
}
.waveform-container {
position: relative;
width: 100%;
padding-top: 160px; /* SPAZIO riservato al logo */
}
```

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;

let recStartTime = 0;
let recTimerId = null;

// 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");

// Drive UI
const btnAuthDrive = document.getElementById("btnAuthDrive");
const btnUploadWav = document.getElementById("btnUploadWav");
```

```
const btnUploadProcessedWav = document.getElementById("btnUploadProcessedWav");
const driveStatusEl = document.getElementById("driveStatus");

// 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: 100, step: 1, value: 50 },
{ id: "pitch", label: "Pitch", min: 0.5, max: 2.0, step: 0.01, value: 1.00 },
{ id: "lowpass", label: "Lowpass Filter", min: 200, max: 20000, step: 1, value: 20000 },
{ id: "highpass", label: "Highpass Filter",min: 10, max: 5000, step: 1, value: 10 },
{ id: "delayTime", label: "Delay", min: 0, max: 0.5, step: 0.01, value: 0 }, // 0–500 ms
{ id: "reverbMix", label: "Reverb", min: 0, max: 1, step: 0.1, value: 0.3 }
];

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

```
// Parameter values
const paramValues = {
gain: 0.5, // 50% (perché ora gain è 0–100 come UI)
pitch: 1.00,
lowpass: 20000,
highpass: 10,
delayTime: 0,
reverbMix: 0.3
};

//
// ============================================================================
// GOOGLE DRIVE (USER'S DRIVE) - OAUTH + UPLOAD
//
// ============================================================================

// >>> INSERISCI QUI IL TUO CLIENT ID (OAuth Web Client) <<<
// Esempio: "1234567890-abc123def456.apps.googleusercontent.com"
const GOOGLE_CLIENT_ID = "704802154881-t0b03q9dc11ijifmopp1f662rnh4hiuf.apps.googleusercontent.com";

// Cartella da creare automaticamente nel Drive dell'utente
const DRIVE_FOLDER_NAME = "Aurora Registrazioni";

// Scope minimo consigliato: consente alla tua app di creare/gestire i file che crea.
const DRIVE_SCOPES = "https://www.googleapis.com/auth/drive.file";

let tokenClient = null;
let driveAccessToken = null;
let driveFolderIdCache = null;

function setDriveStatus(msg) {
if (driveStatusEl) driveStatusEl.textContent = msg;
}

function canUploadNow() {
return !!driveAccessToken && !!audioBlob;
}

function refreshDriveButtons() {
const ok = canUploadNow();
if (btnUploadWav) btnUploadWav.disabled = !ok;
if (btnUploadProcessedWav) btnUploadProcessedWav.disabled = !ok;
}

// Initialize Google Identity Services token client
function initDriveAuth() {
if (!btnAuthDrive || !window.google || !google.accounts || !google.accounts.oauth2) {
// GIS script non ancora caricato (async defer) oppure mancano i bottoni
```

```
return;
}
if (tokenClient) return;
if (!GOOGLE_CLIENT_ID || GOOGLE_CLIENT_ID.includes("PASTE_YOUR_CLIENT_ID_HERE")) {
setDriveStatus("Drive: inserisci GOOGLE_CLIENT_ID in script.js");
btnAuthDrive.disabled = true;
return;
}

tokenClient = google.accounts.oauth2.initTokenClient({
client_id: GOOGLE_CLIENT_ID,
scope: DRIVE_SCOPES,
callback: (resp) => {
driveAccessToken = resp.access_token;
driveFolderIdCache = null; // il token è nuovo: forza refresh ricerca cartella
setDriveStatus("Drive: autorizzato");
refreshDriveButtons();
},
});

btnAuthDrive.addEventListener("click", () => {
setDriveStatus("Drive: in autorizzazione...");
// Chiedi un access token (Google gestisce eventuale consenso)
tokenClient.requestAccessToken({ prompt: "consent" });
});
}

// Prova a inizializzare non appena possibile (script GIS è async)
window.addEventListener("load", () => {
initDriveAuth();
// Se GIS si carica dopo, riprova per un breve periodo
let tries = 0;
const t = setInterval(() => {
initDriveAuth();
tries++;
if (tokenClient || tries > 40) clearInterval(t);
}, 250);
});

//
// ============================================================================
// 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 Recording Is Ready";


  // Drive buttons (se l'utente ha già autorizzato)
  refreshDriveButtons();
};


mediaRecorder.start();
recStartTime = performance.now();
```

if (recTimerId) clearInterval(recTimerId);
recTimerId = setInterval(() ⇒ {
const elapsed = (performance.now() - recStartTime) / 1000;
statusEl.textContent = `Recording... ${elapsed.toFixed(1)}s`;
}, 100);

```
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());
}
if (recTimerId) {
clearInterval(recTimerId);
recTimerId = null;
}

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 = "The Recording 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);
}

//
==========================================================================
// DRIVE HELPERS
//
==========================================================================

async function driveFetch(url, options = {}) {
if (!driveAccessToken) throw new Error("Drive non autorizzato");
const headers = new Headers(options.headers || {});
headers.set("Authorization", Bearer ${driveAccessToken});
return fetch(url, { ...options, headers });
}

async function getOrCreateAuroraFolderId() {
if (driveFolderIdCache) return driveFolderIdCache;

// Cerca cartella per nome (My Drive). Se esiste, prende la prima.
const q = name='${DRIVE_FOLDER_NAME.replace(/'/g, "\\'")}' and
mimeType='application/vnd.google-apps.folder' and trashed=false;
```

```
const listUrl = https://www.googleapis.com/drive/v3/files?
q=${encodeURIComponent(q)}&fields=files(id,name);

const r = await driveFetch(listUrl);
if (!r.ok) throw new Error(await r.text());
const data = await r.json();
if (data.files && data.files.length) {
driveFolderIdCache = data.files[0].id;
return driveFolderIdCache;
}

// Crea cartella
const createUrl = "https://www.googleapis.com/drive/v3/files";
const body = {
name: DRIVE_FOLDER_NAME,
mimeType: "application/vnd.google-apps.folder",
};
const c = await driveFetch(createUrl, {
method: "POST",
headers: { "Content-Type": "application/json" },
body: JSON.stringify(body),
});
if (!c.ok) throw new Error(await c.text());
const created = await c.json();
driveFolderIdCache = created.id;
return driveFolderIdCache;
}

function safeTimestampName(ext) {
const d = new Date();
const pad = (n) => String(n).padStart(2, "0");
const stamp =
${d.getFullYear()}-${pad(d.getMonth()+1)}-${pad(d.getDate())}_${pad(d.getHours())}-${pad(d
.getMinutes())}-${pad(d.getSeconds())};
return Aurora_${stamp}.${ext};
}

async function uploadBlobToDriveResumable(blob, filename, mimeType, folderId) {
const metadata = {
name: filename,
parents: [folderId],
};

// 1) start session
const start = await driveFetch("https://www.googleapis.com/upload/drive/v3/files?uploadType=
resumable", {
method: "POST",
headers: {
```

```
"Content-Type": "application/json; charset=UTF-8",
"X-Upload-Content-Type": mimeType,
"X-Upload-Content-Length": String(blob.size),
},
body: JSON.stringify(metadata),
});

if (!start.ok) throw new Error(await start.text());
const uploadUrl = start.headers.get("Location");
if (!uploadUrl) throw new Error("Upload URL mancante (Location header)");

// 2) upload data
const put = await fetch(uploadUrl, {
method: "PUT",
headers: {
"Content-Type": mimeType,
"Content-Length": String(blob.size),
},
body: blob,
});

if (!put.ok) throw new Error(await put.text());
const data = await put.json();
return data.id;
}

async function getRawWavBlob() {
if (!audioBlob) throw new Error("Nessuna registrazione");
initAudioGraph();
const arrayBuffer = await audioBlob.arrayBuffer();
const audioBuffer = await audioCtx.decodeAudioData(arrayBuffer);
const wavBuffer = audioBufferToWav(audioBuffer);
return new Blob([wavBuffer], { type: "audio/wav" });
}

async function getProcessedWavBlob() {
if (!audioBlob) throw new Error("Nessuna registrazione");

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);
return new Blob([wavBuffer], { type: "audio/wav" });
}
```

```
// Upload buttons
if (btnUploadWav) {
btnUploadWav.addEventListener("click", async () => {
try {
setDriveStatus("Drive: preparo upload...");
const folderId = await getOrCreateAuroraFolderId();
const wavBlob = await getRawWavBlob();
const filename = safeTimestampName("wav");
setDriveStatus("Drive: caricamento WAV Mic...");
const fileId = await uploadBlobToDriveResumable(wavBlob, filename, "audio/wav", folderId);
setDriveStatus(`Drive: caricato (${filename})`);
console.log("Drive fileId (mic):", fileId);
} catch (e) {
console.error(e);
setDriveStatus("Drive: errore upload (vedi console)");
}
});
}

if (btnUploadProcessedWav) {
btnUploadProcessedWav.addEventListener("click", async () => {
try {
setDriveStatus("Drive: preparo upload...");
const folderId = await getOrCreateAuroraFolderId();
const wavBlob = await getProcessedWavBlob();
const filename = safeTimestampName("wav").replace("Aurora_", "Aurora_fx_");
setDriveStatus("Drive: caricamento WAV Effects...");
const fileId = await uploadBlobToDriveResumable(wavBlob, filename, "audio/wav", folderId);
setDriveStatus(`Drive: caricato (${filename})`);
console.log("Drive fileId (fx):", fileId);
} catch (e) {
console.error(e);
setDriveStatus("Drive: errore upload (vedi console)");
}
});
}

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

if (id === "gain" && masterGain) masterGain.gain.value = v / 100;
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)`;

let startMouseAngle = 0;

function mouseAngleDeg(ev, element) {
const r = element.getBoundingClientRect();
const cx = r.left + r.width / 2;
const cy = r.top + r.height / 2;
const dx = ev.clientX - cx;
const dy = ev.clientY - cy;
return Math.atan2(dy, dx) * (180 / Math.PI); // -180..180
}

knob.addEventListener("mousedown", (e) => {
e.preventDefault();
dragging = true;

lastMouseAngle = mouseAngleDeg(e, knob);

document.body.style.userSelect = "none";
});

window.addEventListener("mouseup", () => {
dragging = false;
```

```
document.body.style.userSelect = "";
});

window.addEventListener("mousemove", (e) => {
if (!dragging) return;

const currentMouseAngle = mouseAngleDeg(e, knob);
let delta = currentMouseAngle - lastMouseAngle;

if (delta > 180) delta -= 360;
if (delta < -180) delta += 360;

const speed = e.shiftKey ? 0.35 : 1.0;
angle = clamp(angle + delta * speed, -135, 135);

lastMouseAngle = currentMouseAngle;

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 knobs = document.querySelectorAll(".knob");

knobs.forEach(k => {
const id = k.dataset.target;

//  il volume NON deve essere toccato dai preset
if (id === "gain") return;

if (!(id in p)) return;

const val = p[id];
```

```
paramValues[id] = val;
updateParam(id, val);
updateValLabel(id, val);

const min = +k.dataset.min;
const max = +k.dataset.max;
k.style.transform = rotate(${valueToAngle(val, min, max)}deg);
});

}

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

Di tutto questo codice forniscimi la spiegazione più dettagliata possibile della funzione di ogni componente del programma di modo da poterlo spiegare interamente anche a chi non lo ha mai visto prima

Ottima richiesta: il progetto che hai condiviso è un'applicazione web complessa ma ben strutturata — un **registratore audio con controlli di effetti, gestione di preset, esportazione dei file (anche in WAV), e integrazione con Google Drive** per l'upload. Ti spiegherò tutto passo-passo, mantenendo il linguaggio accessibile ma tecnico, in modo che tu possa comprendere *ogni parte* e poi spiegarla autonomamente a chiunque.

1️⃣ Struttura generale e scopo

Questa web-app consente di:

1. Registrare audio dal microfono via `MediaRecorder`.
2. Visualizzare la forma d'onda in tempo reale con un `<canvas>`.
3. Applicare effetti audio (lowpass, highpass, delay, riverbero, pitch, volume).
4. Salvare o scaricare il risultato in formato `.wav`.
5. Caricare le registrazioni (originali o elaborate) sul proprio Google Drive tramite OAuth 2.

Il progetto combina:

- **HTML** → scheletro grafico e pulsanti dell'interfaccia.
- **CSS** → stile visivo e layout.
- **JavaScript** → tutta la logica per audio, grafica, controlli e Google Drive.

2 HTML: struttura dell'interfaccia

L'HTML definisce **i blocchi visivi e i controlli**.

Componenti principali

- **Waveform** (`<canvas id="waveform">`) → serve per disegnare la forma d'onda.
  Sopra al canvas, un `<img>` di logo è posizionato "overlay".
- **Sezione di registrazione**:
  - Pulsante tondo rosso "REC" (`btnStartRec`)
  - Pulsante quadrato "STOP" (`btnStopRec`)
  - Etichetta di stato (`#status`)
- **Sezione di playback**:
  - `<audio id="player">` → permette di riascoltare la registrazione originale.
  - Pulsante "Play with Effects" (`btnPlayProcessed`) per sentirla con filtri applicati.
- **Knobs** (`#knobsContainer`) → container vuoto che lo script popola dinamicamente con manopole virtuali.
- **Presets** (`#presetsContainer`) → container per i pulsanti dei preset effetti (Clean, Hall, ecc.).
- **Download**:
  - "Download WAV Mic" (file originale)
  - "Download WAV with Effects" (elaborato)
- **Drive**:
  - Pulsante per autorizzare Google Drive.
  - Due pulsanti per caricare le rispettive versioni audio.
  - Span "Drive: non autorizzato" che mostra lo stato dell'account.

 Tutti questi elementi rimangono disattivati o abilitati dinamicamente dallo script JS quando l'audio o i permessi sono disponibili.

3 CSS: aspetto visivo e layout

Il CSS costruisce un'interfaccia piacevole e coerente.

Temi generali

- Stile "soft", con colori panna-avorio e ombre morbide.
- Layout centrato (`display: flex; justify-content: center;`).
- Ogni sezione separata da righe, margini e padding coerenti.

Pulsanti principali

- I pulsanti standard blu, disattivati con grigio.
- I pulsanti REC e STOP hanno forma e colore specifici:
  - `.btn-round-rec`: cerchio rosso con alone più chiaro all'interno.
  - `.btn-square-stop`: rettangolo rosso con quadratino chiaro al centro.

Knobs (manopole)

Ogni manopola è:

- Un elemento `.knob` circolare con gradiente chiaro.
- Ruotabile con il mouse (gestito da JS).
- Ha una label e il valore numerico sotto.

Waveform e logo

- Il `<canvas>` ha sfondo nero e bordo arrotondato.
- Il logo viene sovrapposto sopra il canvas tramite `position: absolute;` e `pointer-events: none` per non bloccare i click.

4️⃣ JavaScript: cuore dell'applicazione

Il JS contiene l'intera logica in vari **moduli funzionali**.

✅ a) Stato e variabili globali

Le prime variabili servono per:

- Gestire la registrazione (`mediaRecorder`, `recordedChunks`...).
- Creare la catena dei nodi audio (`audioCtx`, `lowpassFilter`, `delayNode`, ecc.).
- Salvare i parametri correnti (`paramValues`).
- Gestire la visualizzazione della waveform e gli eventuali animazioni (`animationId`).

✅ b) UI e riferimenti

Tutti gli elementi HTML vengono referenziati con `getElementById` (es. `btnStartRec`, `btnDownloadWav`, ecc.), così da collegare l'interfaccia alla logica.

✅ c) Configurazione: knobs + presets

Knobs

Array `knobsConfig` descrive ogni manopola:

- id → quale parametro controlla (es. `"gain"`, `"lowpass"`),
- valore iniziale, min, max, step,
- label da mostrare sull'interfaccia.

Presets

Oggetto `presetsConfig` contiene preset predefiniti:

- `"clean"`, `"phone"`, `"hall"`, `"lofi"`.
- Ognuno imposta valori ai parametri (filtri, eco, riverbero, pitch...).

✅ d) Google Drive (OAuth & Upload)

È una delle parti più tecniche.

1. **OAuth 2 Client ID** – va messo il proprio ID ottenuto da Google Cloud Console.
   Serve per chiedere all'utente di autorizzare la web app ad accedere al suo Drive.
2. **Token Client**

   ```
   tokenClient = google.accounts.oauth2.initTokenClient({...})
   ```

   Inizializza il flusso di autenticazione.
3. Dopo l'autorizzazione, l'app riceve un **access token** (`driveAccessToken`) che viene poi usato per effettuare richieste a Google Drive API.
4. L'upload avviene in due step:
   - Creazione (o recupero) di una cartella dedicata `Aurora Registrazioni`.
   - Caricamento del file in modalità **resumable upload** (più efficiente e sicura).
5. Le funzioni dedicate (`getRawWavBlob`, `getProcessedWavBlob`) preparano i file WAV originali o elaborati prima di inviarli.

✅ e) Costruzione dinamica dei controlli

Le funzioni `createKnobs()` e `createPresets()` creano dinamicamente i relativi elementi nel DOM.

- Ogni knob riceve i propri dati da `knobsConfig` e reagisce ai movimenti del mouse.
- Ogni preset crea un pulsante che richiama `applyPreset(name)`.

Questo approccio permette di aggiungere facilmente nuovi effetti solo modificando la configurazione.

✅ f) Catena Audio (Audio Graph)

`initAudioGraph()` inizializza il grafo **Web Audio API**, collegando i "nodi" come in un mixer:

```
Mic/Source → LowPass → HighPass → Delay (+feedback)
→ Reverb → Dry/Wet Gains → Master → Analyser → Output
```

- **LowpassFilter**: taglia le alte frequenze.
- **HighpassFilter**: taglia le basse frequenze.
- **DelayNode + Feedback**: ritardo tipo eco.
- **Convolver**: applica una riverberazione generata da un impulso simulato da `createReverbImpulse()`.
- **Gain nodes**: regolano il volume complessivo e la miscela fra segnale "secco" e "riverberato".
- **Analyser**: campiona periodicamente i dati audio per disegnare la forma d'onda.

La funzione `createReverbImpulse()` costruisce artificialmente una risposta all'impulso casuale per simulare un ambiente riverberante.

✅ g) Visualizzazione waveform

La funzione `drawWaveform()` usa l'`AnalyserNode`:

- Ottiene i campioni via `getByteTimeDomainData()`.
- Disegna la curva blu sul canvas 60 volte al secondo (`requestAnimationFrame`).

Si arresta quando la riproduzione finisce.

✅ h) Registrazione microfono

La parte più chiara del progetto:

1. Clic su "REC" → `getUserMedia({audio:true})` apre il microfono.
2. `MediaRecorder` cattura i frammenti (`ondataavailable` li accumula).
3. `onstop()` crea un `Blob` audio (tipo WebM), lo collega all'`<audio>` e abilita i pulsanti.
4. Un piccolo timer aggiorna lo stato ("Recording… 2.5 s").

Stop disattiva il microfono e aggiorna i pulsanti.

✅ i) Playback semplice e con effetti

- Il player `<audio>` può suonare il file originale. Durante la riproduzione si vede la waveform.
- "Play with Effects": decodifica il blob, lo manda nella catena audio (`audioCtx`) e applica in tempo reale i parametri correnti (pitch incluso, tramite `playbackRate`).

Non usa `<audio>` ma un `BufferSourceNode` dedicato per permettere la manipolazione diretta.

✅ j) Download dei file WAV

Due varianti:

1. **Raw WAV** – converte il blob WebM registrato in WAV (senza effetti).
   Usa la libreria esterna `audiobuffer-to-wav`.
2. **Processed WAV** – elabora offline con un `OfflineAudioContext`.
   In questo contesto tutti i nodi sono renderizzati in background ottenendo l'audio completo con effetti.

Al termine, crea un blob e lo scarica automaticamente con un `<a>` invisibile.

✅ k) Upload su Google Drive

Dopo l'autorizzazione:

- Recupera (o crea) la cartella "Aurora Registrazioni".
- Genera un nome file con timestamp (`Aurora_2026-01-28_16-40-12.wav`).
- Esegue l'upload tramite API `uploadType=resumable`.
- Aggiorna lo stato su schermo ("Drive: caricato (nomefile.wav)").

✅ l) Knobs e interazione utente

Ogni manopola reagisce alla rotazione del mouse:

- Calcola l'angolo rispetto al centro (`atan2`).
- Trasforma il movimento in valore tra minimo e massimo.
- Aggiorna al volo il parametro audio (es. `lowpassFilter.frequency.value`) e il valore numerico sul display.
- Premendo *Shift* il movimento è più fine.

Le manopole coprono tutti gli effetti configurabili:
Volume, Pitch, Filtri, Delay, Reverb.

✅ m) Presets

`applyPreset(name)` imposta i valori dei parametri come da configurazione del preset.
Aggiorna le manopole visivamente tramite la funzione `valueToAngle()` (che converte valore in gradi di rotazione).

Il "volume" (gain) non viene modificato dai preset per evitare sbalzi di livello.

✅ n) Pulizia finale

`window.beforeunload` interrompe `requestAnimationFrame()` per evitare animazioni pendenti quando si chiude la pagina.

5️⃣ Esempio logico di flusso d'uso

1. L'utente apre la pagina → vede interfaccia.
2. Clicca *Record* → si attiva il microfono, compare il tempo.
3. Stop → compare player audio, waveform, pulsanti di download/upload attivi.
4. Ruota manopole o sceglie preset → cambia gli effetti.
5. "Play with Effects" → ascolta il risultato processato.
6. "Download WAV" → salva file.
7. "Autorizza Google Drive" → login Google, poi "Carica WAV su Drive".

6️⃣ Tecnologie chiave usate

- **MediaRecorder API** → acquisizione audio dal microfono.
- **Web Audio API** → manipolazione del segnale audio e creazione di effetti.
- **Canvas 2D API** → disegno waveform in tempo reale.
- **Google Identity Services & Drive REST API** → autorizzazione e upload files.
- **OfflineAudioContext** → rendering "non-in-tempo-reale" per esportare audio con effetti applicati.
- **Dynamic DOM manipulation** → creazione automatica di manopole e preset.

🧠 Vantaggi architetturali

- Tutto "client-side": nessun backend richiesto.
- Scalabile: si possono aggiungere altri effetti solo modificando la catena audio e `knobsConfig`.
- Sicuro: l'accesso a Drive è limitato ai file creati dall'app tramite `https://www.googleapis.com/auth/drive.file`.

Vuoi che ti prepari anche **uno schema visuale del grafo audio (blocchi e flussi)** per aiutarti a spiegarlo graficamente a studenti o colleghi?

Si grazie

Propongo uno **schema a blocchi** (testuale ma facilmente trasformabile in diagramma) del grafo audio e del flusso generale, in modo che tu possa ridisegnarlo su carta o in uno strumento tipo draw.io o Figma.[1] [2] [3]

1. Schema del grafo audio "live" (play con effetti)

Questo è il grafo Web Audio usato quando premi **Play with Effects**:

1. **Sorgente audio**
   - `AudioBufferSourceNode` (creato da `audioCtx.createBufferSource()`).
   - Buffer = la registrazione decodificata.
   - `playbackRate.value = paramValues.pitch` (controllo del pitch).
2. **Catena di processori**
   - `BiquadFilterNode` (lowpass):
     `source → lowpassFilter` (frequenza = `paramValues.lowpass`).
   - `BiquadFilterNode` (highpass):
     `lowpassFilter → highpassFilter` (frequenza = `paramValues.highpass`).
   - `DelayNode`:
     `highpassFilter → delayNode` (ritardo = `paramValues.delayTime`).
   - Feedback del delay:
     - `delayNode → delayFeedback (GainNode) → delayNode`
       (crea le ripetizioni, tipo eco).
3. **Riverbero e mix dry/wet**
   - Branch "dry":
     `delayNode → dryGain`.
   - Branch "wet" (riverbero):
     `delayNode → reverbConvolver → wetGain`.
   - Impulse di riverbero generato da `createReverbImpulse(...)`.
   - Controllo reverb: `dryGain.gain = 1 - reverbMix`, `wetGain.gain = reverbMix`.
4. **Uscita**
   - `dryGain → masterGain`, `wetGain → masterGain`.
   - `masterGain → analyser → audioCtx.destination` (casse cuffie).
     (`masterGain.gain` è pilotato dal knob Volume).

Se disegni il grafo, puoi usare questo schema:

- Sorgente → LPF → HPF → Delay → (Dry → Master)
  ↘ Reverb →
  e in parallelo: `Master → Analyser → Output`.

2. Schema del grafo audio “offline” (render per download con effetti)

Quando scarichi **Recording_with_effects.wav**, il grafo è molto simile ma dentro un **OfflineAudioContext**:

1. `OfflineAudioContext(1, length, sampleRate)`
2. `BufferSource` con il file decodificato.
3. Stessa catena: `lp → hp → delay (+fb) → dry/wet → master → destination`.
4. `startRendering()` produce un `AudioBuffer` completo processato.
5. Conversione in WAV (`audioBufferToWav`) e download.

Puoi disegnarlo quasi identico al grafo “live”, ma mettendo sopra una etichetta “OfflineAudioContext (rendering per export)”.

3. Schema audio del player “dry” + waveform

Quando usi solo il player `<audio>` (senza effetti), il flusso è:

1. `<audio id="player">` → `MediaElementAudioSourceNode` (`audioCtx.createMediaElementSource(player)`).
2. `MediaElementSource → analyser → audioCtx.destination`.

In questo caso:

- Non passa nei filtri / delay / reverb.
- Serve solo per:
  - Ascolto “pulito”.
  - Disegnare la waveform sul canvas tramite `AnalyserNode`.[4] [2] [1]

4. Schema del flusso “registrazione → blob → player”

Per spiegare la parte “registrazione” puoi usare questo diagramma:

1. **Utente preme REC**
   - `navigator.mediaDevices.getUserMedia({audio:true})`
   - Ottieni `MediaStream`.
2. **MediaRecorder**
   - `mediaRecorder = new MediaRecorder(stream)`.
   - `ondataavailable` accumula pezzi (`recordedChunks[]`).

3. **Stop**
    - `onstop`:
        - `new Blob(recordedChunks, {type:"audio/webm"})` → `audioBlob`.
        - `URL.createObjectURL(audioBlob)` → `player.src = audioUrl`.
        - Abilita: Play, Download, Upload.

Puoi rappresentarlo come:

Mic → MediaStream → MediaRecorder → Blob (WebM) → `<audio>`.

5. Schema Google Drive (semplificato)

Per la parte Drive puoi fare un diagramma più logico che tecnico:

1. **Autorizzazione**
    - Utente clicca "Autorizza Google Drive".
    - `google.accounts.oauth2.initTokenClient(...)` → ottiene `access_token`.
2. **Preparazione file**
    - Raw: `audioBlob` → `decodeAudioData` → `audioBuffer` → WAV.
    - FX: OfflineAudioContext → `AudioBuffer` processato → WAV.
3. **Cartella**
    - Chiamata API `files.list` con query per nome "Aurora Registrazioni".
    - Se non esiste → `files.create` con mimeType `application/vnd.google-apps.folder`.
4. **Upload resumable**[5]
    - `POST ...uploadType=resumable` con metadata (nome, parent).
    - Ricevi `Location` (session URI).
    - `PUT` sul `Location` con il Blob WAV.
    - Risposta = `fileId` su Drive.

Schema sintetico:

WAV Blob → (get/crea cartella) → avvia sessione upload → carica dati → file su Drive.

6. Schema di interazione UI (alto livello)

Per raccontare tutto il comportamento a chi non ha visto il codice:

1. L'utente usa i **pulsanti**:
    - REC / STOP.
    - Play normale / Play con effetti.
    - Download / Upload.

2. I **knobs** modificano le variabili `paramValues`:
    - Volume → `masterGain.gain`.
    - Pitch → `source.playbackRate`.
    - Lowpass / Highpass → frequenze dei filtri.
    - Delay → `delayNode.delayTime`.
    - Reverb → mix fra `dryGain` e `wetGain`.
3. I **preset** richiamano `applyPreset(name)`:
    - Aggiornano i valori nei `paramValues`.
    - Aggiornano posizione grafica delle manopole.
    - Volume non viene toccato.

Se vuoi, nel prossimo passo posso:

- Scriverti **esattamente** come disegnare il grafo in stile blocchi (tipo: rettangoli, frecce, colori, etichette), oppure
- Prepararti una **lista di slide** (titoli + bullet) per presentare il progetto a una classe o a un team.

⁂

1. https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API/Using_Web_Audio_API
2. https://developer.mozilla.org/en-US/docs/Web/API/AudioNode
3. https://www.w3.org/TR/webaudio-1.1/
4. https://web.dev/articles/webaudio-intro
5. https://developers.google.com/workspace/drive/api/guides/manage-uploads
6. https://developers.google.com/gdata/docs/resumable_upload
7. https://stackoverflow.com/questions/33243444/web-audio-api-discover-a-nodes-connections
8. https://github.com/CaesuraCo/web-audio-graph
9. https://www.recall.ai/blog/how-to-build-a-chrome-recording-extension
10. https://stackoverflow.com/questions/69669177/how-do-i-use-the-mediarecorder-api-on-processed-audio