```
/* -------------------------
   PARTICLE BACKGROUND
------------------------- */
const canvas = document.getElementById("particle-canvas");
const ctx = canvas.getContext("2d");

let particles = [];

function resizeCanvas() {
    canvas.width = window.innerWidth;
    canvas.height = window.innerHeight;
}
resizeCanvas();
window.onresize = resizeCanvas;

function createParticles(num = 80) {
    particles = [];
    for (let i = 0; i < num; i++) {
        particles.push({
            x: Math.random() * canvas.width,
            y: Math.random() * canvas.height,
            r: Math.random() * 2 + 1,
            vx: (Math.random() - 0.5) * 0.3,
            vy: (Math.random() - 0.5) * 0.3,
            alpha: Math.random() * 0.6 + 0.2
        });
    }
}

function drawParticles() {
    ctx.clearRect(0, 0, canvas.width, canvas.height);

    ctx.fillStyle = "white";

    particles.forEach(p => {
        ctx.globalAlpha = p.alpha;
        ctx.beginPath();
        ctx.arc(p.x, p.y, p.r, 0, Math.PI * 2);
        ctx.fill();

        p.x += p.vx;
        p.y += p.vy;

        if (p.x < 0 || p.x > canvas.width) p.vx *= -1;
```

```
        if (p.y < 0 || p.y > canvas.height) p.vy *= -1;
    });

    requestAnimationFrame(drawParticles);
}

createParticles();
drawParticles();

/* ------------------------
   STARTUP SOUND + FADE LOGIC
------------------------- */
const startupSound = document.getElementById("startup-sound");
const appContent = document.getElementById("app-content");
const startupScreen = document.getElementById("startup-screen");

window.onload = () => {
    setTimeout(() => {
        startupSound.play().catch(() => {});
    }, 400);

    setTimeout(() => {
        startupScreen.style.display = "none";
    }, 2100);
};

/* ------------------------
   UPLOAD BOX CLICK HANDLER
------------------------- */
const uploadBox = document.querySelector(".upload-box");

// create hidden file input
const fileInput = document.createElement("input");
fileInput.type = "file";
fileInput.accept = "video/*";
fileInput.style.display = "none";
document.body.appendChild(fileInput);

uploadBox.addEventListener("click", () => {
    fileInput.click();
});

fileInput.addEventListener("change", () => {
    if (fileInput.files.length > 0) {
```

```
      uploadBox.innerHTML = `<p><b>${fileInput.files[0].name}</b> selected</p>`;
    }
});

/* ---------- CLIENT: frame extraction + upload ---------- */

/**
 * extractFrames(file, count)
 * - returns Promise<string[]> where each string is a dataURL (jpeg base64)
 */
async function extractFrames(file, frameCount = 6) {
  return new Promise((resolve, reject) => {
    const video = document.createElement('video');
    video.preload = 'metadata';
    video.muted = true;
    video.playsInline = true;

    video.onloadedmetadata = () => {
      const duration = video.duration;
      const interval = Math.max(0.05, duration / (frameCount + 1));
      const canvas = document.createElement('canvas');
      const ctx = canvas.getContext('2d');

      // set canvas size once video has dimensions
      canvas.width = video.videoWidth || 640;
      canvas.height = video.videoHeight || 360;

      let frames = [];
      let i = 1;

      const capture = (time) => {
        video.currentTime = Math.min(time, duration - 0.01);
      };

      video.onseeked = () => {
        try {
          ctx.drawImage(video, 0, 0, canvas.width, canvas.height);
          // Reduce size & quality for smaller payload: jpeg quality 0.7
          frames.push(canvas.toDataURL('image/jpeg', 0.7));
        } catch (e) {
          // ignore draw errors on some devices
          console.error('drawImage error', e);
        }
```

```
      if (i <= frameCount) {
        i++;
        capture(i * interval);
      } else {
        resolve(frames);
      }
    };

    // start capturing from small offset to avoid black frames
    capture(interval);
  };

  video.onerror = () => reject(new Error('Video load error (file may be corrupt)'));
  // load file as blob URL
  video.src = URL.createObjectURL(file);
 });
}

/**
 * sendFramesToServer(frames, focus)
 * - POST frames + metadata to your /analyze endpoint
 * - returns parsed JSON { result, reason, score }
 */
async function sendFramesToServer(frames, focus = 'auto') {
 // progress hook example (you can wire to UI)
 const body = {
  frames: frames.map(d => d.split(',')[1]), // send base64 body only (no data: prefix)
  focus
 };

 const resp = await fetch('/analyze', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify(body),
 });

 if (!resp.ok) {
  const text = await resp.text();
  throw new Error(`Server error: ${resp.status} ${text}`);
 }

 const json = await resp.json();
 return json; // expected { result: 'REAL'|'DEEPFAKE', reason: '...', score: 0-100 }
}
```

```
/* ---------- Example wiring in your existing upload flow ---------- */

fileInput.addEventListener('change', async () => {
  if (!fileInput.files || fileInput.files.length === 0) return;
  const file = fileInput.files[0];

  // show UI: "extracting frames..."
  showStatus('Extracting frames…');

  try {
    const frames = await extractFrames(file, 6); // 6 frames by default
    showStatus('Analyzing…');

    // show progress (optional) and send to server
    const result = await sendFramesToServer(frames,
document.getElementById('analysisType')?.value || 'auto');

    // Update UI with result
    updateResultsUI(result);
    showStatus('Done');

  } catch (err) {
    console.error(err);
    showError(err.message || 'Analysis failed');
  }
});
```