"use client";

import React from "react";

import Image from "next/image";

import { useRouter } from "next/navigation";

import Papa from "papaparse";

import {

  Card, CardContent, CardHeader, CardTitle,

} from "@/components/ui/card";

import { Button } from "@/components/ui/button";

import { Input } from "@/components/ui/input";

import { Label } from "@/components/ui/label";

import { Tabs, TabsContent, TabsList, TabsTrigger } from "@/components/ui/tabs";

import { Switch } from "@/components/ui/switch";

import { Slider } from "@/components/ui/slider";

import { Progress } from "@/components/ui/progress";

import { Badge } from "@/components/ui/badge";

import {

  ChevronDown, Database, ShieldCheck, Settings, FileText, Activity, Lock,

  Filter, Layers, AlertTriangle, Download, Eye, PlayCircle, Table as TableIcon,

  Gauge, History, Wand2, RefreshCcw, SlidersHorizontal,

} from "lucide-react";

/\* ---------- Helpers UI ---------- \*/

const Tag = ({ children }: { children: React.ReactNode }) => (

  <span className="px-2 py-0.5 text-xs rounded-full bg-brand-50 text-brand-600 border border-brand-100">

    {children}

  </span>

);

const Pill = ({ children }: { children: React.ReactNode }) => (

  <span className="px-2.5 py-1 text-xs rounded-2xl border">{children}</span>

);

const Row = ({ label, value }: { label: React.ReactNode; value: React.ReactNode }) => (

  <div className="flex items-center justify-between py-2">

    <span className="text-sm text-gray-600">{label}</span>

    <span className="text-sm font-medium">{value}</span>

  </div>

);

const Divider = () => <div className="h-px bg-gray-200 my-3" />;

/\* ===================================================================== \*/

export default function AnonymizationStudio() {

  const router = useRouter();

  /\* === ÉTATS PRINCIPAUX === \*/

  const [fileName, setFileName] = React.useState("");

  const [columns, setColumns] = React.useState<string[]>([]);

  const [rowsPreview, setRowsPreview] = React.useState<string[][]>([]);

  const [delimiter, setDelimiter] = React.useState<string>(",");

  const [lastCounts, setLastCounts] = React.useState<{ kept: number; total: number } | null>(null);

  const [totalRows, setTotalRows] = React.useState<number | null>(null); // nombre réel (streaming)

  /\* === REFS D’IMPORT === \*/

  const fileInputRef = React.useRef<HTMLInputElement>(null);

  const lastFileRef = React.useRef<File | null>(null);

  const openPicker = () => fileInputRef.current?.click();

  /\* === JOURNAL / AUDIT === \*/

  type LogItem = { at: string; action: string };

  const [logs, setLogs] = React.useState<LogItem[]>([]);

  function addLog(action: string) {

    const at = new Date().toISOString().replace("T", " ").slice(0, 16);

    setLogs((x) => [{ at, action }, ...x]);

  }

  /\* === PARAMÈTRES & TOGGLES === \*/

  const [epsilon, setEpsilon] = React.useState(2);

  const [kAnon, setKAnon] = React.useState(10);

  const [lDiv, setLDiv] = React.useState(3);

  const [tClose, setTClose] = React.useState(0.3);

  const [suppressExtras, setSuppressExtras] = React.useState(true);

  const [zip3, setZip3] = React.useState(true);

  const [yearOnly, setYearOnly] = React.useState(true);

  const [hashIds, setHashIds] = React.useState(true);

  const [maskPII, setMaskPII] = React.useState(true);

  function notify(feature: string) {

    addLog(`Action: ${feature}`);

    window.alert(`${feature} déclenchée (journalisée). Ajoute ta logique métier ici.`);

  }

  /\* === MAPPING GÉNÉRIQUE === \*/

  type Mapping = {

    id?: string;        // identifiant

    postal?: string;    // code postal

    date1?: string;     // date 1

    date2?: string;     // date 2

    revenue?: string;   // revenu/montant

    sensitive?: string; // attribut sensible

    qids: string[];     // colonnes QI

  };

  const [map, setMap] = React.useState<Mapping>({ qids: [] });

  const COL = (k: keyof Mapping) => (map[k] || "") as string;

  function guessMapping(cols: string[]): Mapping {

    const lc = cols.map((c) => c.toLowerCase());

    const find = (...patterns: RegExp[]) =>

      cols.find((c, i) => patterns.some((p) => p.test(lc[i]))) || undefined;

    const m: Mapping = {

      id: find(/(^|\_)(id|ident|client|no[\_-]?client|numero[\_-]?client)($|\_)/),

      postal: find(/postal|code[\_-]?postal|zip|fsa/),

      date1: find(/date.\*(naiss|birth|ouv|open|transaction|op[eé]ration)/, /^date$/),

      date2: find(/date.\*(op[eé]ration|transaction|maj|update|modif)/),

      revenue: find(/revenu|salaire|income|montant|amount|solde/),

      sensitive: find(/type.\*(compte|assurance|produit|segment)|diagnostic|statut|risk/),

      qids: [],

    };

    m.qids = [m.date1, m.postal, m.revenue].filter(Boolean) as string[];

    return m;

  }

  React.useEffect(() => {

    if (!columns.length) return;

    setMap((prev) => {

      if (prev.qids.length || prev.id || prev.postal || prev.date1 || prev.revenue || prev.sensitive) return prev;

      return guessMapping(columns);

    });

  }, [columns]);

  /\* === IMPORT CSV robuste === \*/

  function countAllRows(file: File) {

    let count = 0;

    Papa.parse(file, {

      header: true, worker: true, skipEmptyLines: true,

      chunk: (res) => { count += (res.data as any[]).length; },

      complete: () => setTotalRows(count),

      error: () => setTotalRows(null),

    });

  }

  const onFile = (e: React.ChangeEvent<HTMLInputElement>) => {

    const f = e.target.files?.[0];

    if (!f) return;

    lastFileRef.current = f;

    setFileName(f.name);

    Papa.parse(f, {

      header: true, worker: true, skipEmptyLines: true, preview: 200,

      complete: (res) => {

        const fields = (res.meta as any).fields as string[] | undefined;

        const data = res.data as Record<string, any>[];

        if (fields && fields.length && data && data.length) {

          setColumns(fields);

          setRowsPreview(data.map((row) => fields.map((c) => String(row?.[c] ?? ""))));

          setDelimiter((res.meta as any).delimiter || ",");

          addLog(`Import CSV (header): ${f.name}`);

          countAllRows(f);

          return;

        }

        Papa.parse(f, {

          header: false, worker: true, skipEmptyLines: true, preview: 200,

          complete: (res2) => {

            const arr = res2.data as unknown as string[][];

            if (!arr || !arr.length) {

              setColumns([]); setRowsPreview([]); window.alert("CSV vide ou illisible."); addLog("Import: CSV vide/illisible"); return;

            }

            const [hdr, ...rest] = arr;

            const looksHeader = hdr.some((x) => typeof x === "string" && /[A-Za-z\_]/.test(String(x)));

            const cols = looksHeader ? (hdr as string[]) : hdr.map((\_, i) => `Col ${i + 1}`);

            const rows = looksHeader ? rest : arr;

            setColumns(cols); setRowsPreview(rows);

            setDelimiter((res2.meta as any).delimiter || ",");

            addLog(`Import CSV (no header): ${f.name}`); countAllRows(f);

          },

          error: (err2) => { console.error(err2); setColumns([]); setRowsPreview([]); addLog("Erreur import (fallback)"); window.alert("Impossible de lire le CSV."); },

        });

      },

      error: (err) => { console.error(err); setColumns([]); setRowsPreview([]); addLog("Erreur d'import CSV"); window.alert("Impossible de lire le CSV."); },

    });

  };

  const handleFileChange = (e: React.ChangeEvent<HTMLInputElement>) => {

    const f = e.target.files?.[0]; if (!f) return; lastFileRef.current = f; onFile(e); e.target.value = "";

  };

  const rePreview = () => {

    const f = lastFileRef.current; if (!f) { openPicker(); return; }

    const dt = new DataTransfer(); dt.items.add(f);

    const fakeEvt = { target: { files: dt.files } } as unknown as React.ChangeEvent<HTMLInputElement>;

    onFile(fakeEvt); addLog("Prévisualisation relancée");

  };

  /\* === EXPORTS === \*/

  function exportPipelineJSON() {

    const pipeline = {

      meta: { createdAt: new Date().toISOString(), delimiter },

      settings: { epsilon, kAnon, lDiv, tClose, zip3, yearOnly, hashIds, maskPII, suppressExtras },

      mapping: map, columns,

      transforms, // <<< ajout pour traçabilité

    };

    const blob = new Blob([JSON.stringify(pipeline, null, 2)], { type: "application/json" });

    const url = URL.createObjectURL(blob); const a = document.createElement("a");

    a.href = url; a.download = "pipeline.json"; a.click(); URL.revokeObjectURL(url);

    addLog("Export pipeline JSON");

  }

  function exportReportHTML() {

    const esc = (s: unknown) =>

      (s ?? "").toString().replace(/[&<>]/g, (ch) => ({ "&": "&amp;", "<": "&lt;", ">": "&gt;" }[ch] as string));

    const html = `<!doctype html><meta charset="utf-8"><title>Rapport</title>

<style>body{font-family:system-ui,Segoe UI,Roboto,Helvetica,Arial,sans-serif;margin:24px}h1{font-size:20px;margin:0 0 8px}h2{font-size:16px;margin:24px 0 8px}table{border-collapse:collapse;width:100%}th,td{border:1px solid #ddd;padding:6px 8px;font-size:12px}th{background:#f6f6f6;text-align:left}.meta{color:#666;font-size:12px;margin-bottom:16px}</style>

<h1>Rapport d’anonymisation</h1>

<div class="meta">Généré: ${new Date().toLocaleString()}</div>

<h2>Jeu de données</h2>

<p>Fichier: <b>${esc(fileName) || "(non importé)"}</b> — Délimiteur: <b>${delimiter}</b> — Lignes: <b>${totalRows ?? "?"}</b></p>

<h2>Paramètres</h2>

<ul><li>k=${kAnon}</li><li>l=${lDiv}</li><li>t=${tClose}</li><li>ε=${epsilon}</li></ul>

<h2>Mapping</h2>

<table><thead><tr><th>Rôle</th><th>Colonne</th></tr></thead><tbody>

${["id","postal","date1","date2","revenue","sensitive"].map((k)=>`<tr><td>${k}</td><td>${esc((map as any)[k]||"—")}</td></tr>`).join("")}

<tr><td>QI</td><td>${(map.qids||[]).join(", ") || "—"}</td></tr></tbody></table>`;

    const blob = new Blob([html], { type: "text/html;charset=utf-8" });

    const url = URL.createObjectURL(blob); const a = document.createElement("a");

    a.href = url; a.download = "rapport.html"; a.click(); URL.revokeObjectURL(url);

    addLog("Télécharger rapport (HTML)");

  }

  async function logout() { await fetch("/api/logout", { method: "POST" }); router.push("/login"); }

  function resetAll() {

    setFileName(""); setColumns([]); setRowsPreview([]); setDelimiter(",");

    setLastCounts(null); setTotalRows(null); lastFileRef.current = null;

    if (fileInputRef.current) fileInputRef.current.value = "";

    addLog("Réinitialisation du studio");

  }

  /\* === Classifieur colonnes (catégories limitées) === \*/

  function classifyColumn(name: string) {

    const n = name.toLowerCase();

    let type = "Texte";

    if (/(^|\_)id($|\_)|ident|client|ssn|nas/.test(n) || /email|courriel/.test(n) || /phone|tel|tél/.test(n)) type = "Identifiant";

    else if (/date/.test(n)) type = "Date";

    else if (/postal|code[\_-]?postal|zip|fsa|ville|adresse/.test(n)) type = "Géoloc";

    else if (/revenu|salaire|income|montant|amount|solde|balance/.test(n)) type = "Numérique";

    let cat: "Renseignements personnels" | "Finance" | "Santé" | "Assurance" = "Renseignements personnels";

    if (/revenu|salaire|income|montant|amount|solde|balance|iban|bic|numero[\_-]?compte|carte[\_-]?credit/.test(n)) cat = "Finance";

    else if (/sant[eé]|m[eé]dical|diagnostic|pathologie|maladie|allergie|mutuelle/.test(n)) cat = "Santé";

    else if (/assur|police[\_-]?assurance|sinistre/.test(n)) cat = "Assurance";

    const pii =

      /(^|\_)id($|\_)|ident|client|ssn|nas/.test(n) ||

      /email|courriel/.test(n) || /phone|tel|tél/.test(n) ||

      /date/.test(n) || /postal|code[\_-]?postal|zip|fsa/.test(n) ||

      /nom|pr[eé]nom|adresse|ville/.test(n);

    let risk = 20;

    if (/(^|\_)id($|\_)|ident|client|ssn|nas/.test(n)) risk = 92;

    else if (/email|courriel|phone|tel|tél/.test(n)) risk = 88;

    else if (/date/.test(n)) risk = 78;

    else if (/postal|code[\_-]?postal|zip|fsa/.test(n)) risk = 61;

    else if (/revenu|salaire|income|montant|amount|solde|balance|carte[\_-]?credit/.test(n)) risk = 69;

    return { type, cat, pii, risk };

  }

  /\* === Affichages % + buckets === \*/

  function pct(n: number | null) { return n == null ? "—" : `${Math.round(n)}%`; }

  function riskBucket(score: number) {

    if (score >= 70) return { label: "Élevé", text: "text-danger-600", bg: "bg-danger-50", pill: "bg-danger-50 text-danger-700" };

    if (score >= 40) return { label: "Moyen", text: "text-warn-600", bg: "bg-warn-50", pill: "bg-warn-50 text-warn-700" };

    return { label: "Faible", text: "text-success-600", bg: "bg-success-50", pill: "bg-success-50 text-success-700" };

  }

  /\* === Anonymisation helpers === \*/

  async function sha256Hex(input: string) {

    const data = new TextEncoder().encode(input);

    const digest = await crypto.subtle.digest("SHA-256", data);

    return Array.from(new Uint8Array(digest)).map(b => b.toString(16).padStart(2,"0")).join("");

  }

  function onlyYear(v: any) { const s = String(v ?? ""); const m = s.match(/\b(19|20)\d{2}\b/); return m ? m[0] : s.slice(0,4); }

  function toFSA(postal: any) { const s = String(postal ?? "").replace(/\s+/g,"").toUpperCase(); return s ? s.slice(0,3) + "\*\*\*" : s; }

  function binIncome(v: any) { const n = Number(String(v).replace(/[^\d.-]/g, "")); if (!isFinite(n)) return ""; const step = 10000; const low = Math.floor(n/step)\*step; const high = low + step; return `${low/1000}–${high/1000}k`; }

  function maskSimple(value: any) {

    const s = String(value ?? ""); if (!s) return s;

    if (s.includes("@")) { const [u,d] = s.split("@"); return `${u?.slice(0,2)||""}\*\*\*@${d||""}`; }

    const digits = s.replace(/\D/g,""); if (digits.length>=8) return s.slice(0,2)+"\*\*\*\*"+s.slice(-2);

    return s.replace(/.(?=..)/g,"\*");

  }

  function laplaceNoise(value: number, epsilonVal: number, sensitivity = 1) {

    if (!epsilonVal || epsilonVal <= 0) return value;

    const u = Math.random() - 0.5; const b = sensitivity / epsilonVal;

    const noise = -b \* Math.sign(u) \* Math.log(1 - 2 \* Math.abs(u));

    return value + noise;

  }

  function qiValueForKey(r: Record<string, any>, col: string) {

    if (!col) return "";

    if (map.date1 === col || map.date2 === col) return r.\_year ?? r[col];

    if (map.postal === col) return r.\_fsa ?? r[col];

    if (map.revenue === col) return r.\_revBin ?? r[col];

    return r[col];

  }

  function quasiKeyDynamic(r: Record<string, any>) {

    return (map.qids || []).map((c) => String(qiValueForKey(r, c))).join("|");

  }

  function distribution(rows: Record<string, any>[], key: string) {

    const counts = new Map<string, number>(); let total = 0;

    for (const r of rows) { const v = String(r[key] ?? ""); counts.set(v,(counts.get(v)||0)+1); total++; }

    const dist = new Map<string, number>(); counts.forEach((c,k)=>dist.set(k, c/Math.max(1,total))); return dist;

  }

  function tvd(p: Map<string, number>, q: Map<string, number>) {

    const keys = new Set<string>([...p.keys(), ...q.keys()]); let sum = 0;

    keys.forEach((k)=>{ const a=p.get(k)||0, b=q.get(k)||0; sum += Math.abs(a-b); }); return 0.5\*sum;

  }

  /\* === Loi 25 — proxies === \*/

  type Loi25Metrics = { individualization: number | null; correlation: number | null; inference: number | null; };

  function isDirectPIIName(n: string) {

    const s = n.toLowerCase();

    return (/^id$|(^|\_)id($|\_)|ident|client|ssn|nas/.test(s) || /email|courriel/.test(s) || /phone|tel|tél/.test(s));

  }

  function buildTmpRows(raw: Record<string, any>[]) {

    return raw.map((r0) => {

      const r = { ...r0 } as Record<string, any>;

      const dateCol = map.date1 || map.date2; const dateSrc = dateCol ? r[dateCol] : undefined;

      const y = dateSrc != null ? onlyYear(dateSrc) : ""; r.\_year = y;

      if (map.postal) r.\_fsa = toFSA(r[map.postal]);

      if (map.revenue) r.\_revBin = binIncome(r[map.revenue]);

      return r;

    });

  }

  function loi25Individualization(rows: Record<string, any>[], cols: string[], qids: string[]) {

    if (!rows.length) return null;

    const piiCols = cols.filter(isDirectPIIName);

    let cntPII = 0;

    for (const r of rows) if (piiCols.some((c)=>{const v=r[c]; return v!=null && String(v).trim()!=="";})) cntPII++;

    const piiPresence = cntPII / rows.length;

    let uniqueRatio = 0;

    if (qids.length) {

      const groups = new Map<string, number>();

      for (const r of rows) { const key = qids.map((c)=>String(qiValueForKey(r,c))).join("|"); groups.set(key,(groups.get(key)||0)+1); }

      let uniques = 0; groups.forEach((n)=>{ if (n===1) uniques++; });

      uniqueRatio = uniques / Math.max(1, groups.size);

    }

    return Math.max(0, Math.min(100, 100\*(0.6\*piiPresence + 0.4\*uniqueRatio)));

  }

  function loi25Correlation(rows: Record<string, any>[], qids: string[]) {

    if (!rows.length || !qids.length) return null;

    const groups = new Map<string, number>();

    for (const r of rows) { const key = qids.map((c)=>String(qiValueForKey(r,c))).join("|"); groups.set(key,(groups.get(key)||0)+1); }

    let singletons = 0; groups.forEach((n)=>{ if (n===1) singletons++; });

    const ratio = singletons / Math.max(1, groups.size);

    return Math.max(0, Math.min(100, 100\*ratio));

  }

  function loi25Inference(rows: Record<string, any>[], sensitive: string | undefined, qids: string[]) {

    if (!rows.length || !sensitive || !qids.length) return null;

    const gDist = distribution(rows, sensitive);

    let total = 0, groups = 0;

    const mapGroups = new Map<string, Record<string, any>[]>();

    for (const r of rows) { const key = qids.map((c)=>String(qiValueForKey(r,c))).join("|"); if (!mapGroups.has(key)) mapGroups.set(key, []); mapGroups.get(key)!.push(r); }

    for (const [, g] of mapGroups) { const d = distribution(g, sensitive); let maxP = 0; d.forEach((p)=>{ if (p>maxP) maxP=p; }); total += maxP; groups++; }

    const avgMax = total / Math.max(1, groups);

    let entropy = 0; gDist.forEach((p)=>{ if (p>0) entropy += -p\*Math.log2(p); });

    const entropyNorm = Math.min(1, entropy / 4);

    return Math.max(0, Math.min(100, 100 \* (avgMax \* (0.6 + 0.4 \* (1 - entropyNorm)))));

  }

  const [loi25Before, setLoi25Before] = React.useState<Loi25Metrics | null>(null);

  const [loi25After, setLoi25After] = React.useState<Loi25Metrics | null>(null);

  React.useEffect(() => {

    if (!columns.length || !rowsPreview.length) { setLoi25Before(null); setLoi25After(null); return; }

    const raw = rowsPreview.map((row) => { const o: Record<string, any> = {}; columns.forEach((c, i) => (o[c] = row[i])); return o; });

    const beforeRows = raw.map((r)=>({ ...r }));

    const before: Loi25Metrics = {

      individualization: loi25Individualization(beforeRows, columns, map.qids),

      correlation: loi25Correlation(beforeRows, map.qids),

      inference: loi25Inference(beforeRows, map.sensitive, map.qids),

    };

    setLoi25Before(before);

    const afterRows = buildTmpRows(raw);

    const after: Loi25Metrics = {

      individualization: loi25Individualization(afterRows, columns, map.qids),

      correlation: loi25Correlation(afterRows, map.qids),

      inference: loi25Inference(afterRows, map.sensitive, map.qids),

    };

    setLoi25After(after);

  }, [columns, rowsPreview, map, yearOnly, zip3, epsilon, kAnon, lDiv, tClose]);

  /\* === TRANSFORMATIONS PAR COLONNE (RECO) === \*/

  type RecAction = "none" | "generalize" | "mask" | "pseudonymize" | "noise" | "suppress";

  type TransformRule = { action: RecAction; params?: Record<string, any> };

  const [transforms, setTransforms] = React.useState<Record<string, TransformRule>>({});

  /\*\* petite recommandation par défaut selon le type (non bloquante) \*/

  function defaultRecFor(col: string): TransformRule {

    const meta = classifyColumn(col);

    if (meta.type === "Identifiant") return { action: "pseudonymize" };

    if (meta.type === "Numérique" || meta.cat === "Finance") return { action: "generalize", params: { granularity: "bin" } };

    if (meta.type === "Date") return { action: "generalize", params: { granularity: "year" } };

    if (meta.type === "Géoloc") return { action: "generalize", params: { granularity: "postal\_fsa" } };

    return { action: "none" };

  }

  function ensureRec(col: string) {

    setTransforms((t) => (t[col] ? t : { ...t, [col]: defaultRecFor(col) }));

  }

  /\* === EXECUTE anonymisation === \*/

  async function applyTransforms(rows: Record<string, any>[]) {

    const out = rows.map(r => ({ ...r }));

    const colsSet = new Set(Object.keys(out[0] || {}));

    for (const [col, rule] of Object.entries(transforms)) {

      if (!colsSet.has(col)) continue;

      const action = rule.action;

      for (const r of out) {

        const v = r[col];

        switch (action) {

          case "suppress":

            r[col] = undefined;

            break;

          case "pseudonymize":

            if (v != null && String(v).trim() !== "") {

              r[col] = await sha256Hex(String(v)); // pseudo stable

            }

            break;

          case "mask":

            r[col] = maskSimple(v);

            break;

          case "generalize": {

            const g = rule.params?.granularity ?? "auto";

            const s = String(v ?? "");

            if (!s) break;

            if (g === "year") {

              r[col] = onlyYear(v);

            } else if (g === "postal\_fsa") {

              r[col] = toFSA(v);

            } else if (g === "bin") {

              r[col] = binIncome(v);

            } else if (g === "auto") {

              // heuristique simple

              const meta = classifyColumn(col);

              if (meta.type === "Date") r[col] = onlyYear(v);

              else if (meta.type === "Géoloc") r[col] = toFSA(v);

              else if (meta.type === "Numérique" || meta.cat === "Finance") r[col] = binIncome(v);

            }

            break;

          }

          case "noise": {

            const n = Number(v);

            if (isFinite(n)) {

              const eps = Number(rule.params?.epsilon ?? epsilon ?? 2);

              const rounded = Number(rule.params?.round ?? 10);

              const noisy = laplaceNoise(n, eps, 500);

              r[col] = Math.round(noisy / rounded) \* rounded;

            }

            break;

          }

          case "none":

          default:

            break;

        }

      }

    }

    return out;

  }

  async function executeAnonymization() {

    const file = lastFileRef.current;

    if (!file) { window.alert("Aucun fichier sélectionné."); return; }

    if (!map.qids.length) { window.alert("Sélectionne au moins un quasi-identifiant (QI)."); return; }

    addLog("Exécuter l’anonymisation");

    await new Promise<void>((resolve, reject) => {

      Papa.parse<Record<string, any>>(file, {

        header: true, skipEmptyLines: true, worker: true,

        complete: async (res) => {

          try {

            let rows = res.data;

            // 1) Appliquer recommandations personnalisées (par colonne)

            rows = await applyTransforms(rows);

            // 2) Tes toggles globaux (hash, masquage simple, année/FSA, bruit revenus)

            for (const r of rows) {

              if (map.id && (map.id in r) && hashIds) r[map.id] = await sha256Hex(String(r[map.id]));

              if (maskPII) {

                for (const k of Object.keys(r)) {

                  const n = k.toLowerCase();

                  if (/(email|e-mail|courriel|phone|tél|telephone)/.test(n)) r[k] = maskSimple(r[k]);

                }

              }

              const dateCol = map.date1 || map.date2;

              const dateSrc = dateCol ? r[dateCol] : undefined;

              const y = dateSrc != null ? onlyYear(dateSrc) : "";

              if (yearOnly && dateCol) r[dateCol] = y;

              r.\_year = y;

              if (map.postal && map.postal in r) {

                r.\_fsa = toFSA(r[map.postal]);

                if (zip3) r[map.postal] = r.\_fsa;

              } else { r.\_fsa = ""; }

              if (map.revenue && map.revenue in r) {

                const n = Number(String(r[map.revenue]).replace(/[^\d.-]/g, ""));

                const noisy = isFinite(n) ? laplaceNoise(n, epsilon || 2, 500) : NaN;

                r[map.revenue] = isFinite(noisy) ? Math.round(noisy) : r[map.revenue];

                r.\_revBin = binIncome(r[map.revenue]);

              } else { r.\_revBin = ""; }

            }

            // 3) k/l/t – conformité Loi 25

            const k = Math.max(2, Number(kAnon) || 2);

            const l = Math.max(1, Number(lDiv) || 1);

            const t = Math.max(0, Number(tClose) || 0);

            const SENS = map.sensitive || "";

            const globalDist = SENS ? distribution(rows, SENS) : new Map<string, number>();

            const groupMap = new Map<string, Record<string, any>[]>();

            for (const r of rows) {

              const key = quasiKeyDynamic(r);

              if (!groupMap.has(key)) groupMap.set(key, []);

              groupMap.get(key)!.push(r);

            }

            const kept: Record<string, any>[] = [];

            for (const [, grp] of groupMap) {

              if (!grp.length) continue;

              if (grp.length < k) continue;

              if (SENS && SENS in grp[0]) {

                const distinct = new Set(grp.map((x) => String(x[SENS] ?? "")));

                if (distinct.size < l) continue;

                const gDist = distribution(grp, SENS);

                const dist = tvd(gDist, globalDist);

                if (dist > t && suppressExtras) continue;

              }

              kept.push(...grp);

            }

            for (const r of kept) { delete r.\_year; delete r.\_fsa; delete r.\_revBin; }

            const csv = Papa.unparse(kept, { quotes: false });

            const blob = new Blob([csv], { type: "text/csv;charset=utf-8" });

            const url = URL.createObjectURL(blob);

            const a = document.createElement("a");

            a.href = url; a.download = `${file.name.replace(/\.csv$/i, "")}-anonymise.csv`; a.click();

            URL.revokeObjectURL(url);

            setLastCounts({ kept: kept.length, total: rows.length });

            addLog(`Export CSV anonymisé (${kept.length}/${rows.length} lignes; k=${k}, l=${l}, t≤${t})`);

            resolve();

          } catch (e) { console.error(e); window.alert("Erreur pendant l’anonymisation."); reject(e as any); }

        },

        error: (err) => { console.error(err); window.alert("Impossible de relire le CSV."); reject(err as any); }

      });

    });

  }

  function autoPreset() {

    setHashIds(true); setMaskPII(true);

    setYearOnly(true); setZip3(true);

    setEpsilon(2); setKAnon(10); setLDiv(3); setTClose(0.3);

    setSuppressExtras(true);

    addLog("Preset auto-anonymisation (DP+k+l+t) appliqué");

  }

  /\* === SCORES indicatifs (moyenne risques colonnes) === \*/

  type RiskScores = { before: number; after: number; target: number };

  const [scores, setScores] = React.useState<RiskScores | null>(null);

  function columnRisk(col: string) { return classifyColumn(col).risk; }

  function riskReductionFor(col: string) {

    const n = col.toLowerCase(); let d = 0;

    if (hashIds && (/(^|\_)id($|\_)|ident|client|ssn|nas|email|phone|tel/.test(n))) d += 40;

    if (yearOnly && /date/.test(n)) d += 20;

    if (zip3 && /(code[\_-]?postal|postal|zip|fsa)/.test(n)) d += 15;

    if (/revenu|salaire|income|montant|amount|solde/.test(n)) { if (epsilon <= 3) d += 10; else if (epsilon <= 5) d += 5; }

    return Math.min(50, d);

  }

  function policyTarget() {

    const kPart = Math.min(25, Math.max(0, kAnon - 5));

    const epsPart = Math.min(20, Math.max(0, 8 - epsilon) \* 2.5);

    return Math.max(15, Math.round(55 - kPart - epsPart));

  }

  React.useEffect(() => {

    if (!columns.length) { setScores(null); return; }

    const raw = columns.map(columnRisk);

    const before = raw.length ? Math.round(raw.reduce((a,b)=>a+b,0)/raw.length) : 0;

    const afterArr = columns.map((c,i)=>Math.max(0, raw[i]-riskReductionFor(c)));

    const after = afterArr.length ? Math.round(afterArr.reduce((a,b)=>a+b,0)/afterArr.length) : 0;

    setScores({ before, after, target: policyTarget() });

  }, [columns, hashIds, yearOnly, zip3, epsilon, kAnon]);

 /\* === Composant de cellule Recommandation (version simplifiée, sans erreurs TS) === \*/

function RecommendationCell({

  col,

  disabled = false,

}: {

  col: string;

  disabled?: boolean;

}) {

  const meta = classifyColumn(col);

  // 1) Règle par défaut robuste (évite erreurs si defaultRecFor diffère)

  const getDefaultRule = (): { action: RecAction; param?: any } => {

    try {

      // Si ton defaultRecFor attend un nom de colonne, garde (col).

      // S'il attend un meta, remplace par (meta).

      const r = (defaultRecFor as any)(col);

      if (r && typeof r.action === "string") return r;

    } catch {}

    return { action: "none" };

  };

  const rule = (transforms[col] as any) ?? getDefaultRule();

  // 2) Liste d'options SANS spread/const compliqués (évite erreurs TS)

  const opts: Array<{ value: RecAction; label: string }> = [

    { value: "none", label: "Aucune" },

  ];

  // Identifiants → pseudo / masque

  if (meta.type === "Identifiant") {

    opts.push(

      { value: "pseudonymize", label: "Pseudonymiser" },

      { value: "mask", label: "Masquer" }

    );

  }

  // Numérique / Finance → généraliser (tranches) + bruit (ε-DP)

  if (meta.type === "Numérique" || meta.cat === "Finance") {

    opts.push(

      { value: "generalize", label: "Généraliser (tranches)" },

      { value: "noise", label: "Bruit (ε-DP)" }

    );

  }

  // Date → généraliser sur l’année

  if (meta.type === "Date") {

    opts.push({ value: "generalize", label: "Généraliser (année)" });

  }

  // Géoloc → généraliser sur la FSA

  if (meta.type === "Géoloc") {

    opts.push({ value: "generalize", label: "Généraliser (FSA)" });

  }

  // Texte libre → masque

  if (meta.type === "Texte") {

    opts.push({ value: "mask", label: "Masquer" });

  }

  // Toujours possible

  opts.push({ value: "suppress", label: "Supprimer" });

  // Déduplication propre

  const seen = new Set<string>();

  const options = opts.filter((o) => {

    if (seen.has(o.value)) return false;

    seen.add(o.value);

    return true;

  });

  // 3) Handlers

  function updateAction(a: RecAction) {

    setTransforms((cur) => ({ ...cur, [col]: { ...rule, action: a } }));

  }

  function updateParam(v: string | number) {

    setTransforms((cur) => ({ ...cur, [col]: { ...rule, param: v } }));

  }

  // 4) Rendu (option ouverte + paramètres contextuels)

  return (

    <div className="flex items-center gap-2">

      <select

        disabled={disabled}

        value={rule.action}

        onChange={(e) => updateAction(e.target.value as RecAction)}

        className="h-8 rounded-md border px-2 text-sm"

      >

        {options.map((o) => (

          <option key={o.value} value={o.value}>

            {o.label}

          </option>

        ))}

      </select>

      {/\* Indications de granularité selon le type + param ε pour bruit \*/}

      {rule.action === "generalize" && meta.type === "Date" && (

        <span className="text-xs text-gray-500">année</span>

      )}

      {rule.action === "generalize" && meta.type === "Géoloc" && (

        <span className="text-xs text-gray-500">FSA</span>

      )}

      {rule.action === "generalize" && meta.type === "Numérique" && (

        <span className="text-xs text-gray-500">tranches</span>

      )}

      {rule.action === "noise" && (

        <div className="flex items-center gap-1">

          <span className="text-xs text-gray-500">ε</span>

          <input

            type="number"

            step="0.1"

            value={typeof rule.param === "number" ? rule.param : 5}

            onChange={(e) => updateParam(Number(e.target.value))}

            className="h-7 w-16 rounded-md border px-2 text-xs"

          />

        </div>

      )}

    </div>

  );

}

  /\* =============================== UI =============================== \*/

  return (

    <div className="min-h-screen w-full bg-white text-[#111]">

      <header className="sticky top-0 z-30 border-b bg-white/70 backdrop-blur">

        <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8 h-16 flex items-center justify-between">

          <div className="flex items-center gap-3">

            <Image src="/logo.svg" alt="Logo" width={32} height={32} />

            <div>

              <div className="font-semibold leading-tight">Studio d’anonymisation</div>

              <div className="text-xs text-gray-500 -mt-0.5">Conforme Loi 25 – Québec</div>

            </div>

          </div>

          <div className="flex items-center gap-2">

            <div className="hidden md:flex items-center gap-2">

              <Tag>Conf. diff. (ε)</Tag><Tag>k/l/t</Tag><Tag>Traçabilité</Tag>

            </div>

            <Button variant="outline" className="gap-2" onClick={() => notify("Journal")}><History className="h-4 w-4"/>Journal</Button>

            <Button className="gap-2 bg-brand-600 hover:bg-brand-600/90 text-white shadow-sm" onClick={exportReportHTML}>

              <Download className="h-4 w-4" /> Télécharger rapport

            </Button>

            <Button variant="outline" onClick={logout}>Se déconnecter</Button>

            <Button variant="destructive" className="gap-2" onClick={resetAll}><RefreshCcw className="h-4 w-4"/>Réinitialiser</Button>

          </div>

        </div>

      </header>

      <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8 py-6 grid grid-cols-1 lg:grid-cols-12 gap-6">

        {/\* Sidebar \*/}

        <aside className="lg:col-span-3 space-y-3">

          <Card className="rounded-2xl border bg-white shadow-sm">

            <CardHeader className="pb-2"><CardTitle className="text-base flex items-center gap-2"><Database className="h-4 w-4 text-brand-600"/>Jeux de données</CardTitle></CardHeader>

            <CardContent className="space-y-3">

              <Label htmlFor="dataset">Importer / sélectionner</Label>

              <div className="flex gap-2">

                <Input id="dataset" placeholder="mon\_fichier.csv" value={fileName} readOnly className="focus:ring-2 focus:ring-brand-600/30 focus:border-brand-600" />

                <Button variant="outline" onClick={openPicker}>Parcourir</Button>

                <input ref={fileInputRef} type="file" accept=".csv,text/csv" className="hidden" onChange={handleFileChange}/>

              </div>

              <Button variant="secondary" className="w-full gap-2" onClick={rePreview} disabled={!fileName}>

                <TableIcon className="h-4 w-4"/> Prévisualiser

              </Button>

              <Divider />

              <Row

  label="Lignes détectées"

  value={

    <span>

      {totalRows !== null && totalRows !== undefined

        ? totalRows

        : rowsPreview && rowsPreview.length > 0

        ? rowsPreview.length

        : "—"}

    </span>

  }

/>

<Row

  label="Colonnes"

  value={<span>{columns && columns.length > 0 ? columns.length : "—"}</span>}

/>

<Row

  label="Dernier export"

  value={<span>{lastCounts ? `${lastCounts.kept}/${lastCounts.total}` : "—"}</span>}

/>

            </CardContent>

          </Card>

          {/\* Mapping \*/}

          <Card className="rounded-2xl border bg-white shadow-sm">

            <CardHeader className="pb-2"><CardTitle className="text-base">Schéma & mapping</CardTitle></CardHeader>

            <CardContent className="space-y-3 text-sm">

              {["id","postal","date1","date2","revenue","sensitive"].map((k) => (

                <div key={k} className="flex items-center justify-between gap-3">

                  <Label className="w-32 capitalize">{k}</Label>

                  <select className="flex-1 h-9 rounded-md border px-2"

                    value={COL(k as keyof Mapping)} onChange={(e) => setMap((m) => ({ ...m, [k]: e.target.value || undefined }))}>

                    <option value="">(Aucune)</option>

                    {columns.map((c) => <option key={c} value={c}>{c}</option>)}

                  </select>

                </div>

              ))}

              <Divider />

              <Label>Quasi-identifiants (pour k/l/t)</Label>

              <div className="grid grid-cols-1 gap-1 max-h-40 overflow-auto border rounded-lg p-2">

                {columns.map((c) => {

                  const checked = map.qids.includes(c);

                  return (

                    <label key={c} className="flex items-center gap-2">

                      <input type="checkbox" checked={checked}

                        onChange={(e) => setMap((m) => { const q = new Set(m.qids); e.target.checked ? q.add(c) : q.delete(c); return { ...m, qids: [...q] }; })}/>

                      <span className="text-xs">{c}</span>

                    </label>

                  );

                })}

              </div>

              <div className="text-xs text-gray-500">Astuce : si un QI est une date / code postal / revenu, la clé utilisera <b>l’année</b>, <b>la FSA</b>, <b>la tranche</b>.</div>

            </CardContent>

          </Card>

          {/\* Politique & conformité (k/l/t) \*/}

          <Card className="rounded-2xl border bg-white shadow-sm">

            <CardHeader className="pb-2"><CardTitle className="text-base flex items-center gap-2"><ShieldCheck className="h-4 w-4 text-brand-600"/>Politique & seuils</CardTitle></CardHeader>

            <CardContent className="space-y-4">

              <Row label="k-anonymat minimum" value={<Pill>k ≥ {kAnon}</Pill>} />

              <Row label="l-diversity" value={<Pill>l ≥ {lDiv}</Pill>} />

              <Row label="t-closeness (TVD)" value={<Pill>t ≤ {tClose}</Pill>} />

              <Divider />

              <Label className="text-sm">Bruit (ε) – confidentialité différentielle</Label>

              <Slider value={[epsilon]} min={0} max={10} step={0.5} onValueChange={(v) => setEpsilon(v[0] as number)} />

              <div className="text-xs text-gray-500">Plus ε est petit, plus la protection est forte (bruit ↑).</div>

              <Divider />

              <div className="flex items-center justify-between"><Label>Supprimer groupes non conformes</Label><Switch checked={suppressExtras} onCheckedChange={setSuppressExtras} /></div>

            </CardContent>

          </Card>

        </aside>

        {/\* Main \*/}

        <main className="lg:col-span-9 space-y-6">

          {/\* Actions \*/}

          <div className="flex flex-wrap items-center gap-2">

            <Button variant="outline" className="gap-2" onClick={() => notify("Profiler")}><Filter className="h-4 w-4" />Profiler</Button>

            <Button variant="outline" className="gap-2" onClick={() => notify("Recommandations")}><SlidersHorizontal className="h-4 w-4" />Recommandations</Button>

            <Button variant="outline" className="gap-2" onClick={() => notify("Pipelines")}><Layers className="h-4 w-4" />Pipelines</Button>

            <Button variant="outline" className="gap-2" onClick={autoPreset}><Wand2 className="h-4 w-4" />Auto-anonymiser</Button>

            <div className="ml-auto flex items-center gap-2">

              <Input placeholder="Rechercher une colonne…" className="w-64 focus:ring-2 focus:ring-brand-600/30 focus:border-brand-600" />

              <Button variant="outline" className="gap-2" onClick={() => notify("Audit")}><Eye className="h-4 w-4" />Audit</Button>

              <Button className="gap-2 bg-brand-600 hover:bg-brand-600/90 text-white shadow-sm" onClick={executeAnonymization}><PlayCircle className="h-4 w-4" />Exécuter</Button>

            </div>

          </div>

          <Tabs defaultValue="colonnes">

            <TabsList>

              <TabsTrigger value="colonnes">Colonnes & risques</TabsTrigger>

              <TabsTrigger value="transformations">Transformations</TabsTrigger>

              <TabsTrigger value="simulation">Simulation ré-id.</TabsTrigger>

              <TabsTrigger value="audit">Traçabilité</TabsTrigger>

              <TabsTrigger value="rapport">Rapport</TabsTrigger>

            </TabsList>

            {/\* Colonnes & risques \*/}

            <TabsContent value="colonnes" className="space-y-4">

              <Card className="rounded-2xl border bg-white shadow-sm">

                <CardHeader className="pb-2"><CardTitle className="text-base flex items-center gap-2"><Gauge className="h-4 w-4 text-brand-600"/>Score de risque global</CardTitle></CardHeader>

                <CardContent>

                  <div className="grid sm:grid-cols-3 gap-4">

                    <div className="p-4 rounded-2xl border"><div className="text-xs text-gray-500">Avant traitements</div><div className="text-3xl font-semibold text-danger-600">{scores ? scores.before : "—"}</div>{scores && <Progress value={scores.before} className="mt-2" />}</div>

                    <div className="p-4 rounded-2xl border"><div className="text-xs text-gray-500">Après recommandations</div><div className="text-3xl font-semibold text-warn-600">{scores ? scores.after : "—"}</div>{scores && <Progress value={scores.after} className="mt-2" />}</div>

                    <div className="p-4 rounded-2xl border"><div className="text-xs text-gray-500">Objectif politique</div><div className="text-3xl font-semibold text-success-600">{scores ? `≤ ${scores.target}` : "—"}</div>{scores && <Progress value={scores.target} className="mt-2" />}</div>

                  </div>

                </CardContent>

              </Card>

              <Card className="rounded-2xl border bg-white shadow-sm">

                <CardHeader className="pb-2"><CardTitle className="text-base flex items-center gap-2"><TableIcon className="h-4 w-4 text-brand-600"/>Dictionnaire de données</CardTitle></CardHeader>

                <CardContent>

                  <div className="overflow-x-auto">

                    <table className="min-w-full text-sm">

                      <thead>

                        <tr className="text-left text-gray-500">

                          <th className="py-2 font-medium">Colonne</th>

                          <th className="py-2 font-medium">Type</th>

                          <th className="py-2 font-medium">Catégorie</th>

                          <th className="py-2 font-medium">PII</th>

                          <th className="py-2 font-medium">Risque</th>

                          <th className="py-2 font-medium">Recommandation</th>

                        </tr>

                      </thead>

                      <tbody>

                        {columns.length === 0 ? (

                          <tr><td colSpan={6} className="py-8"><div className="rounded-xl border border-brand-100 bg-brand-50 text-brand-600 text-sm p-3 text-center">Importez un CSV pour voir le dictionnaire de données.</div></td></tr>

                        ) : (

                          columns.map((col) => {

                            ensureRec(col); // initialise si pas présent

                            const meta = classifyColumn(col); const bucket = riskBucket(meta.risk);

                            return (

                              <tr key={col} className="border-t">

                                <td className="py-2 font-medium">{col}</td>

                                <td className="py-2">{meta.type}</td>

                                <td className="py-2"><span className="px-2.5 py-1 text-xs rounded-2xl border">{meta.cat}</span></td>

                                <td className="py-2">{meta.pii ? (<Badge className="bg-danger-50 text-danger-600">Oui</Badge>) : (<Badge className="bg-gray-100">Non</Badge>)}</td>

                                <td className="py-2"><span className={`inline-flex items-center px-2.5 py-1 text-xs rounded-lg ${bucket.pill}`}>{bucket.label}</span></td>

                                <td className="py-2">

                                  <RecommendationCell col={col} disabled={false} />

                                </td>

                              </tr>

                            );

                          })

                        )}

                      </tbody>

                    </table>

                  </div>

                </CardContent>

              </Card>

            </TabsContent>

            {/\* Transformations \*/}

            <TabsContent value="transformations" className="space-y-4">

              <div className="grid md:grid-cols-2 gap-4">

                <Card className="rounded-2xl border bg-white shadow-sm">

                  <CardHeader className="pb-2"><CardTitle className="text-base flex items-center gap-2"><Settings className="h-4 w-4 text-brand-600"/>Paramétrages</CardTitle></CardHeader>

                  <CardContent className="space-y-3">

                    <div className="flex items-center justify-between"><Label>Date → année uniquement</Label><Switch checked={yearOnly} onCheckedChange={setYearOnly} /></div>

                    <div className="flex items-center justify-between"><Label>Code postal → FSA</Label><Switch checked={zip3} onCheckedChange={setZip3} /></div>

                    <div className="flex items-center justify-between"><Label>Identifiants → hachage (SHA-256)</Label><Switch checked={hashIds} onCheckedChange={setHashIds} /></div>

                    <div className="flex items-center justify-between"><Label>Masquage simple email/téléphone</Label><Switch checked={maskPII} onCheckedChange={setMaskPII} /></div>

                    <Divider />

                    <Label className="text-sm">k-anonymat ciblé</Label><Slider value={[kAnon]} min={2} max={50} step={1} onValueChange={(v) => setKAnon(v[0] as number)} />

                    <Label className="text-sm">l-diversity min.</Label><Slider value={[lDiv]} min={1} max={10} step={1} onValueChange={(v) => setLDiv(v[0] as number)} />

                    <Label className="text-sm">t-closeness (TVD max)</Label><Slider value={[tClose]} min={0} max={1} step={0.05} onValueChange={(v) => setTClose(v[0] as number)} />

                    <Label className="text-sm">Bruit (ε) – conf. différentielle</Label><Slider value={[epsilon]} min={0} max={10} step={0.5} onValueChange={(v) => setEpsilon(v[0] as number)} />

                  </CardContent>

                </Card>

                <Card className="rounded-2xl border bg-white shadow-sm">

                  <CardHeader className="pb-2"><CardTitle className="text-base flex items-center gap-2"><Lock className="h-4 w-4 text-brand-600"/>Prévisualisation (exemple)</CardTitle></CardHeader>

                  <CardContent>

                    <div className="grid grid-cols-2 gap-3 text-sm">

                      <div className="p-3 rounded-xl border">

                        <div className="text-xs text-gray-500 mb-1">Avant</div>

                        <div className="font-mono text-xs bg-gray-50 p-2 rounded-lg overflow-x-auto">

{`id: 879-22-0194

email: client@example.com

téléphone: 418-555-0199

code\_postal: G7H4B2

date: 1998-11-02

revenu: 73,450$`}

                        </div>

                      </div>

                      <div className="p-3 rounded-xl border">

                        <div className="text-xs text-gray-500 mb-1">Après</div>

                        <div className="font-mono text-xs bg-success-50 p-2 rounded-lg overflow-x-auto">

{`id: 7c6a3… (hash)

email: cl\*\*\*@example.com

téléphone: 41\*\*\*\*99

code\_postal: G7H\*\*\*

date: 1998

revenu: 70–80k (+bruit)`}

                        </div>

                      </div>

                    </div>

                    <Divider />

                    <Button variant="outline" className="gap-2 w-full" onClick={() => notify("Comparer sur 1000 lignes")}><Eye className="h-4 w-4" />Comparer sur 1000 lignes</Button>

                  </CardContent>

                </Card>

              </div>

            </TabsContent>

            {/\* Simulation / Loi 25 \*/}

            <TabsContent value="simulation" className="space-y-4">

              <Card className="rounded-2xl border bg-white shadow-sm">

                <CardHeader className="pb-2">

                  <CardTitle className="text-base flex items-center gap-2"><ShieldCheck className="h-4 w-4 text-brand-600"/>Conformité Loi 25 — Individualisation / Corrélation / Inférence</CardTitle>

                </CardHeader>

                <CardContent className="space-y-4">

                  <div className="text-xs text-gray-500">Seuils (indicatifs) — Individualisation ≤ <b>20%</b>, Corrélation ≤ <b>30%</b>, Inférence ≤ <b>35%</b>.</div>

                  {[

                    { key: "individualization" as const, label: "Individualisation", thr: 20 },

                    { key: "correlation" as const,       label: "Corrélation",      thr: 30 },

                    { key: "inference" as const,         label: "Inférence",        thr: 35 },

                  ].map(({ key, label, thr }) => {

                    const b = loi25Before?.[key]; const a = loi25After?.[key]; const ok = a != null ? a <= thr : null;

                    return (

                      <div key={key} className="p-3 rounded-2xl border">

                        <div className="flex items-center justify-between">

                          <div className="text-sm font-medium">{label}</div>

                          <div>{ok == null ? (<Badge className="bg-gray-100 text-gray-600">N/A</Badge>) : ok ? (<Badge className="bg-success-50 text-success-600">Conforme</Badge>) : (<Badge className="bg-danger-50 text-danger-600">À améliorer</Badge>)}</div>

                        </div>

                        <div className="grid sm:grid-cols-2 gap-3 mt-2">

                          <div className="p-3 rounded-xl border"><div className="text-xs text-gray-500">Avant</div><div className="text-2xl font-semibold text-danger-600">{pct(b ?? null)}</div>{b != null && <Progress value={Math.max(0, Math.min(100, b))} className="mt-2" />}</div>

                          <div className="p-3 rounded-xl border"><div className="text-xs text-gray-500">Après</div><div className="text-2xl font-semibold text-success-600">{pct(a ?? null)}</div>{a != null && <Progress value={Math.max(0, Math.min(100, a))} className="mt-2" />}</div>

                        </div>

                        {b != null && a != null && <div className="text-xs text-gray-600 mt-2">Réduction: <b>{Math.max(0, Math.round(b - a))}%</b> — Seuil: {thr}%</div>}

                      </div>

                    );

                  })}

                </CardContent>

              </Card>

            </TabsContent>

            {/\* Audit \*/}

            <TabsContent value="audit" className="space-y-4">

              <Card className="rounded-2xl border bg-white shadow-sm">

                <CardHeader className="pb-2"><CardTitle className="text-base flex items-center gap-2"><FileText className="h-4 w-4 text-brand-600" />Journal d’audit</CardTitle></CardHeader>

                <CardContent>

                  <div className="text-sm space-y-2">

                    {logs.length === 0 && (<div className="rounded-xl border border-brand-100 bg-brand-50 text-brand-600 text-sm p-3">Aucun événement pour le moment.</div>)}

                    {logs.map((e, i) => (

                      <div key={i} className="flex items-start gap-3 p-3 rounded-xl border hover:bg-gray-50">

                        <Activity className="h-4 w-4 text-gray-500 mt-0.5" />

                        <div><div className="text-xs text-gray-500">{e.at}</div><div className="font-medium">{e.action}</div></div>

                      </div>

                    ))}

                  </div>

                </CardContent>

              </Card>

            </TabsContent>

            {/\* Rapport \*/}

            <TabsContent value="rapport" className="space-y-4">

              <Card className="rounded-2xl border bg-white shadow-sm">

                <CardHeader className="pb-2"><CardTitle className="text-base flex items-center gap-2"><FileText className="h-4 w-4 text-brand-600" />Sommaire & exports</CardTitle></CardHeader>

                <CardContent className="space-y-3 text-sm">

                  <Row label="Jeu de données" value={fileName || "—"} />

                  <Row label="Colonnes détectées" value={String(columns.length || "—")} />

                  <Row label="Lignes (réelles)" value={String(totalRows ?? "—")} />

                  <Row label="Dernier export (lignes gardées)" value={lastCounts ? `${lastCounts.kept}/${lastCounts.total}` : "—"} />

                  <Divider />

                  <div className="flex flex-wrap gap-2">

                    <Button variant="outline" className="gap-2" onClick={exportReportHTML}><Download className="h-4 w-4" />Télécharger rapport (HTML)</Button>

                    <Button variant="outline" onClick={exportPipelineJSON}>Exporter JSON (pipeline)</Button>

                  </div>

                </CardContent>

              </Card>

            </TabsContent>

          </Tabs>

        </main>

      </div>

      <footer className="border-t py-6">

        <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8 text-xs text-gray-500">

          © {new Date().getFullYear()} — Prototype conforme Loi 25. k/l/t, bruit (ε), généralisation, masquage et suppression appliqués à l’export.

        </div>

      </footer>

    </div>

  );

}

FUNCTION METIERS VERSION 1

// === FONCTIONS MÉTIER ===

function previewFile() {

  const f = lastFileRef.current;

  if (!f) {

    window.alert("Aucun fichier sélectionné.");

    return;

  }

  const reader = new FileReader();

  reader.onload = (e) => {

    try {

      const text = String(e.target?.result ?? "");

      const rawLines = text.replace(/\r/g, "").split("\n").filter(Boolean).slice(0, 50);

      const grid = rawLines.map(l => l.split(delimiter || ",")); // adapte `delimiter` si tu as un état

      console.log("Aperçu du fichier :", grid);

      setRowsPreview(grid);

      addLog("Action: Prévisualiser");

    } catch (err) {

      console.error("Erreur prévisualisation :", err);

    }

  };

  reader.readAsText(f);

}

function runProfiler(ds: Record<string, any>[]) {

  if (!ds.length) {

    window.alert("Aucun jeu de données chargé.");

    return;

  }

  const colonnes = Object.keys(ds[0]);

  const stats = colonnes.map((col) => ({

    colonne: col,

    distinct: new Set(ds.map((r) => r[col])).size,

    vides: ds.filter((r) => !r[col]).length,

  }));

  console.table(stats);

  setProfilerReport(stats);

  addLog("Action: Profiler");

}

function runRecommendations(ds: Record<string, any>[]) {

  if (!ds.length) {

    window.alert("Aucun jeu de données chargé.");

    return;

  }

  const recs: Record<string, TransformRule> = {};

  for (const col of Object.keys(ds[0])) {

    const n = col.toLowerCase();

    if (/(email|courriel)/.test(n)) recs[col] = { action: "mask" };

    else if (/(phone|tél|tel)/.test(n)) recs[col] = { action: "mask" };

    else if (/date/.test(n)) recs[col] = { action: "generalize", params: { type: "date-year" } };

    else if (/postal|zip|fsa/.test(n)) recs[col] = { action: "generalize", params: { type: "postal-fsa" } };

    else if (/revenu|salaire|income|montant|amount/.test(n)) recs[col] = { action: "noise", params: { beta: 0.5 } };

  }

  console.log("✨ Recommandations générées :", recs);

  setTransforms(recs);

  addLog("Action: Recommandations");

}

async function runPipelines(ds: Record<string, any>[]) {

  console.log("🚀 Pipeline lancé");

  runProfiler(ds);

  runRecommendations(ds);

  await executeAnonymization();

  console.log("✅ Pipeline terminé");

  addLog("Action: Pipelines");

}

function runAudit(ds: Record<string, any>[], k: number) {

  if (!ds.length) {

    window.alert("Aucun jeu de données chargé.");

    return;

  }

  const groups = new Map<string, number>();

  for (const r of ds) {

    const key = JSON.stringify(r);

    groups.set(key, (groups.get(key) || 0) + 1);

  }

  const nonConformes = [...groups.values()].filter((size) => size < k).length;

  const rapport = {

    total: ds.length,

    groupes: groups.size,

    nonConformes,

    conforme: nonConformes === 0,

  };

  console.log("📊 Rapport d’audit :", rapport);

  setAuditReport(rapport);

  addLog(`Action: Audit (k=${k})`);

}

function runJournal() {

  console.table(logs);

  addLog("Action: Journal");

}

async function runAutoAnonymisation() {

  if (!rows.length) {

    window.alert("Aucun jeu de données chargé.");

    return;

  }

  console.log("🤖 Auto-anonymisation lancée");

  runRecommendations(rows);

  await executeAnonymization();

  addLog("Action: Auto-anonymisation");

  console.log("✅ Auto-anonymisation terminée");

}

function downloadReport() {

  if (!auditReport) {

    window.alert("Aucun rapport disponible. Lance d’abord un Audit.");

    return;

  }

  const blob = new Blob([JSON.stringify(auditReport, null, 2)], { type: "application/json" });

  const url = URL.createObjectURL(blob);

  const a = document.createElement("a");

  a.href = url;

  a.download = "rapport\_anonymisation.json";

  a.click();

  URL.revokeObjectURL(url);

  addLog("Action: Télécharger rapport");

  console.log("📥 Rapport téléchargé");

}

// === FIN FONCTIONS MÉTIER ===