Gmail **Fillipe Guerra <fillipe.backup@gmail.com>**

IA_REVIEW_PARTE2

1 mensagem

Fillipe Guerra <fillipe.backup@gmail.com> 27 de outubro de 2025 às 18:24
Para: Fillipe Augusto Gomes Guerra <fillipe182@hotmail.com>, Fillipe Guerra <fillipe.backup@gmail.com>

Etapa 2 — Parsers completos + OCR + Score (γ/δ/ζ) + Ingest por URL/Arquivo

0) Novas dependências

No `package.json` (além das da Etapa 1):

```
{
  "dependencies": {
    "jsdom": "^24.1.0",
    "pdf-parse": "^1.1.1",
    "mammoth": "^1.6.0",
    "iconv-lite": "^0.6.3",
    "node-fetch": "^3.3.2",
    "sharp": "^0.33.4",
    "tesseract.js": "^5.0.5",
    "unzipper": "^0.11.0",
    "papaparse": "^5.4.1",
    "dayjs": "^1.11.13"
  }
}
```

`.env` (acréscimos):

```
AI_PARSER_MAX_BYTES=52428800
AI_OCR_LANG=eng+por
AI_FRESH_DAYS_HALF=14      # meia-vida do frescor (em dias) para o score δ
```

---

1) Schemas adicionais (grafo/frescor/autoridade)

Se ainda não tiver do round anterior, adicione:

/shared/schema.ai.graph.ts

```
import { pgTable, uuid, varchar, text, real, integer, timestamp, index, primaryKey } from "drizzle-orm/pg-core";

export const aiEntities = pgTable("ai_entities", {
  id: uuid("id").primaryKey().defaultRandom(),
  tenantId: varchar("tenant_id",{length:64}).notNull(),
  type: varchar("type",{length:32}).notNull(), // PERSON|ORG|LOC|PRODUCT|DATE|MISC
  value: varchar("value",{length:256}).notNull(),
  createdAt: timestamp("created_at").defaultNow().notNull()
}, t=>({
  idx: index("ai_ent_tenant_val_idx").on(t.tenantId, t.value)
}));

export const aiEntityLinks = pgTable("ai_entity_links", {
  tenantId: varchar("tenant_id",{length:64}).notNull(),
  srcId: uuid("src_id").notNull(),
  dstId: uuid("dst_id").notNull(),
  weight: real("weight").default(0).notNull(),
```

```
  updatedAt: timestamp("updated_at").defaultNow().notNull()
}, t=>({
  pk: primaryKey({ columns: [t.tenantId, t.srcId, t.dstId] })
}));
```

Observação: a autoridade virá como metadado de documento (`metaJson`) e será traduzida em um score.

---

2) Parser completo (HTML/PDF/DOCX/PPTX/CSV/IMG) + OCR

/server/ai/parser.full.ts

```
import fetch from "node-fetch";
import { JSDOM } from "jsdom";
import pdf from "pdf-parse";
import mammoth from "mammoth";
import iconv from "iconv-lite";
import fs from "fs";
import path from "path";
import unzipper from "unzipper";
import Papa from "papaparse";
import sharp from "sharp";
import Tesseract from "tesseract.js";

const MAX_BYTES = Number(process.env.AI_PARSER_MAX_BYTES || 50*1024*1024);
const OCR_LANG = process.env.AI_OCR_LANG || "eng+por";

export async function fetchBuffer(url: string) {
  const r = await fetch(url);
  if (!r.ok) throw new Error(`HTTP ${r.status}`);
  const ab = await r.arrayBuffer();
  if (ab.byteLength > MAX_BYTES) throw new Error("Arquivo excede limite");
  return Buffer.from(ab);
}

export function htmlToText(html: string) {
  const dom = new JSDOM(html);
  const document = dom.window.document;
  ["script","style","noscript","iframe","nav","footer","header","form"].forEach(s =>
    document.querySelectorAll(s).forEach(n => n.remove())
  );
  return (document.body?.textContent || "").replace(/\s+/g," ").trim();
}

async function parsePDF(buf: Buffer) {
  const out = await pdf(buf);
  return { text: out.text, meta: { type: "pdf", pages: out.numpages || undefined, sourceRank: 0.8 } };
}

// DOCX (mammoth). Para .doc (binário legado), sugerimos instalar "antiword" no sistema e chamar via
child_process.
async function parseDOCX(buf: Buffer, name?: string) {
  const out = await mammoth.extractRawText({ buffer: buf });
  return { text: out.value, meta: { type: "docx", name, sourceRank: 0.7 } };
}

async function parsePPTX(buf: Buffer, name?: string) {
  // PPTX: extrai textos dos XML de slides
  const tmp: string[] = [];
  const zip = await unzipper.Open.buffer(buf);
  for (const file of zip.files) {
    if (/ppt\/slides\/slide\d+\.xml$/i.test(file.path)) {
      const cnt = await file.buffer();
      const xml = iconv.decode(cnt, "utf8");
      const texts = [...xml.matchAll(/<a:t>(.*?)<\/a:t>/g)].map(m => m[1]).join(" ");
      if (texts.trim()) tmp.push(texts);
    }
  }
  return { text: tmp.join("\n"), meta: { type: "pptx", name, slides: tmp.length, sourceRank: 0.65 } };
```

```
}

async function parseCSV(buf: Buffer, name?: string) {
  const str = iconv.decode(buf, "utf8");
  const parsed = Papa.parse<string[]>(str);
  const rows = parsed.data as string[][];
  const lines = rows.map(r => r.join(" | ")).join("\n");
  return { text: lines, meta: { type: "csv", name, rows: rows.length, sourceRank: 0.6 } };
}

async function parseHTML(buf: Buffer, url: string) {
  const html = iconv.decode(buf, "utf8");
  const text = htmlToText(html);
  const titleMatch = html.match(/<title[^>]*>(.*?)<\/title>/i);
  const title = titleMatch ? titleMatch[1].trim() : undefined;
  return { text, title, meta: { type: "html", url, sourceRank: 0.9 } };
}

export async function parseURL(url: string): Promise<{ text: string; title?: string; meta?: any }> {
  const buf = await fetchBuffer(url);
  const lower = url.toLowerCase();
  if (lower.endsWith(".pdf")) return parsePDF(buf);
  if (lower.endsWith(".docx")) return parseDOCX(buf, path.basename(url));
  if (lower.endsWith(".pptx")) return parsePPTX(buf, path.basename(url));
  if (lower.endsWith(".csv")) return parseCSV(buf, path.basename(url));
  return parseHTML(buf, url);
}

export async function parseLocalFile(fp: string): Promise<{ text: string; title?: string; meta?: any
}> {
  const stat = fs.statSync(fp);
  if (stat.size > MAX_BYTES) throw new Error("Arquivo excede limite");
  const buf = fs.readFileSync(fp);
  const lower = fp.toLowerCase();
  if (lower.endsWith(".pdf")) return parsePDF(buf);
  if (lower.endsWith(".docx")) return parseDOCX(buf, path.basename(fp));
  if (lower.endsWith(".pptx")) return parsePPTX(buf, path.basename(fp));
  if (lower.endsWith(".csv")) return parseCSV(buf, path.basename(fp));
  const str = iconv.decode(buf, "utf8");
  if (/<html/i.test(str)) return parseHTML(buf, fp);
  return { text: str, title: path.basename(fp), meta: { type: "txt", sourceRank: 0.6 } };
}

/** Decodifica PNG/JPEG → RGBA + (opcional) OCR para texto embutido */
export async function decodeImageToRGBA(buf: Buffer, doOCR = false) {
  const img = sharp(buf).ensureAlpha();
  const { data, info } = await img.raw().toBuffer({ resolveWithObject: true });
  let ocrText = "";
  if (doOCR) {
    const ocr = await Tesseract.recognize(buf, OCR_LANG, { logger: () => {} });
    ocrText = (ocr.data?.text || "").replace(/\s+/g," ").trim();
  }
  return { rgba: new Uint8ClampedArray(data.buffer, data.byteOffset, data.byteLength), width:
info.width, height: info.height, ocrText };
}
```

---

3) Grafo semântico e frescor/autoridade (γ/δ/ζ)

/server/ai/kg.ts (NER leve + links) — use o do round anterior; aqui adicionamos frescor/autoridade no scorer.

/server/ai/hybrid-score.ts (substituir pelo ampliado):

```
import { embedTexts } from "./embeddings.text";
import { knnText, knnImage, getChunksByIds } from "./vector-store";
import { db } from "../db";
import { aiPolicies, aiDocuments, aiChunks } from "@shared/schema.ai.core";
import { nerLight, prLikeScore } from "./kg";
import dayjs from "dayjs";
```

```
import { and, eq } from "drizzle-orm";

type Weights = { alpha:number; beta:number; gamma:number; delta:number; zeta:number };

async function getWeights(tenantId: string): Promise<Weights> {
  const rows = await db.select().from(aiPolicies)
    .where(and(eq(aiPolicies.tenantId, tenantId), eq(aiPolicies.name, "retrieval_weights_v2"))).limi
t(1);
  if (!rows.length) return { alpha: 0.7, beta: 0.15, gamma: 0.1, delta: 0.03, zeta: 0.02 };
  try { const w = JSON.parse(rows[0].dataJson);
    return { alpha: w.alpha ?? 0.7, beta: w.beta ?? 0.15, gamma: w.gamma ?? 0.1, delta: w.delta ??
0.03, zeta: w.zeta ?? 0.02 };
  } catch { return { alpha: 0.7, beta: 0.15, gamma: 0.1, delta: 0.03, zeta: 0.02 }; }
}

function freshness(createdAt?: string | Date) {
  if (!createdAt) return 0;
  const half = Number(process.env.AI_FRESH_DAYS_HALF || 14);
  const ageDays = Math.max(0, dayjs().diff(dayjs(createdAt), "day"));
  // Score ~ e^{-ln(2)*age/half}  (decai meia-vida)
  return Math.exp(-Math.log(2) * (ageDays / Math.max(1, half)));
}
function authority(metaJson?: string) {
  try {
    const m = metaJson ? JSON.parse(metaJson) : {};
    const r = Number(m.sourceRank ?? 0.5); // 0..1
    return Math.max(0, Math.min(1, r));
  } catch { return 0.5; }
}

export async function hybridRetrieveFull(tenantId: string, queryText: string, imageQueryVec?:
number[], k=12) {
  const W = await getWeights(tenantId);

  const [qv] = await embedTexts([queryText]);
  const vText = await knnText(tenantId, qv, 80);
  const vImage = imageQueryVec ? await knnImage(tenantId, imageQueryVec, 80) : [];

  const map = new Map<string, { text:number; image:number }>();
  for (const r of vText) map.set(r.chunkId, { text: r.score, image: 0 });
  for (const r of vImage) map.set(r.chunkId, { ...(map.get(r.chunkId) || { text:0, image:0 }), image:
r.score });

  const ids = [...map.keys()];
  const chunks = await getChunksByIds(ids);

  // γ: grafo semântico – entidades da query dão bônus para chunks do mesmo documento (PR-like)
  const ents = nerLight(queryText).map(e => e.value);
  // prLikeScore retorna um mapa de entidade->peso agregado
  const pr = await prLikeScore(tenantId, ents);

  const scored = chunks.map(ch => {
    const s = map.get(ch.id)!;
    const docId = ch.document_id;
    let g = 0;
    if (ents.length) {
      // bônus simples: soma de pesos para entidades mencionadas (documento como proxy)
      // (poderia guardar entidade→documento na ingest; aqui mantemos simples)
      const bonus = ents.reduce((acc, e) => acc + (pr.get(e) || 0), 0);
      g = Math.min(1, Math.log1p(bonus)/5);
    }
    const f = freshness(ch.created_at || undefined as any);
    const a = authority(ch.meta_json || undefined);

    const score = W.alpha*(s.text||0) + W.beta*(s.image||0) + W.gamma*g + W.delta*f + W.zeta*a;
    return { id: ch.id, score, parts: { text:s.text||0, image:s.image||0, graph:g, fresh:f, auth:a },
chunk: ch };
  }).sort((a,b)=>b.score-a.score).slice(0, k);
```

```
  return scored;
}
```

> Nota: `nerLight` e `prLikeScore` estão no `./kg` (do round anterior). Se ainda não colou, me avise que eu re-incluo.

---

4) Ingest por URL/Arquivo (com auto-detecção + OCR)

/server/ai/routes.ingest.ext.ts

```
import type { Express, Request } from "express";
import multer from "multer";
import { parseURL, parseLocalFile, decodeImageToRGBA } from "./parser.full";
import { ingestText, ingestImage } from "./ingest";
import path from "path";
import fs from "fs";

const upload = multer({ dest: "./uploads" });

function ctx(req: Request) {
  const tenantId = (req as any).tenantId || process.env.PRIMARY_TENANT_ID!;
  return { tenantId };
}

export function registerIngestExtendedRoutes(app: Express) {
  app.post("/api/ai/ingest.url", async (req, res) => {
    try {
      const { tenantId } = ctx(req);
      const { url, source="url" } = req.body || {};
      if (!url) return res.status(400).json({ error: "url required" });
      const { text, title, meta } = await parseURL(url);
      const r = await ingestText(tenantId, { source, uri: url, title, text, meta });
      res.json(r);
    } catch (e:any) { res.status(400).json({ error: e?.message }); }
  });

  app.post("/api/ai/ingest.file", upload.single("file"), async (req, res) => {
    try {
      const { tenantId } = ctx(req);
      if (!req.file) return res.status(400).json({ error: "file required" });
      const fp = path.resolve(req.file.path);
      const lower = req.file.originalname.toLowerCase();

      // imagens comuns: png/jpg/jpeg/webp → faz OCR + embedding
      if (/\.(png|jpg|jpeg|webp)$/i.test(lower)) {
        const buf = fs.readFileSync(fp);
        const { rgba, width, height, ocrText } = await decodeImageToRGBA(buf, true);
        const meta = { type: "image", name: req.file.originalname, sourceRank: 0.6, ocr: !!ocrText };
        const rImg = await ingestImage(tenantId, { source: "file", uri: req.file.originalname, title:
req.file.originalname, rgbaData: rgba, width, height, meta });
        // se OCR gerou texto, também entra como doc textual adicional (mesmo arquivo)
        if (ocrText && ocrText.length > 20) {
          await ingestText(tenantId, { source: "ocr", uri: req.file.originalname+"#ocr", title:
req.file.originalname+" (OCR)", text: ocrText, meta: { sourceRank: 0.5 } });
        }
        fs.unlinkSync(fp);
        return res.json(rImg);
      }

      // demais: PDF/DOCX/PPTX/CSV/TXT/HTML
      const { text, title, meta } = await parseLocalFile(fp);
      const r = await ingestText(tenantId, { source: "file", uri: req.file.originalname, title, text,
meta });
      fs.unlinkSync(fp);
      res.json(r);
    } catch (e:any) { res.status(400).json({ error: e?.message }); }
  });
}
```

Registrar no bootstrap (`server/routes.ts`):

```
import { registerAiRoutesV2 } from "./ai/routes";
import { registerIngestExtendedRoutes } from "./ai/routes.ingest.ext";

export function registerRoutes(app: Express) {
  app.use("/api", (req, _res, next) => { (req as any).tenantId = process.env.PRIMARY_TENANT_ID!;
next(); });
  registerAiRoutesV2(app);
  registerIngestExtendedRoutes(app);
}
```

Etapa 3 — Painel Admin (Next.js) com Métricas + Heatmap + Knowledge Explorer

0) Endpoints de dados para o painel

/server/ai/routes.metrics.ts

```
import type { Express, Request } from "express";
import { db } from "../db";
import { aiEvalDaily } from "@shared/schema.ai.eval";
import { aiEntities, aiEntityLinks } from "@shared/schema.ai.graph";

function ctx(req: Request){ return (req as any).tenantId || process.env.PRIMARY_TENANT_ID!; }

export function registerMetricsRoutes(app: Express) {
  app.get("/api/ai/metrics/daily", async (req, res) => {
    const tenantId = ctx(req);
    const rows = await db.select().from(aiEvalDaily).where((aiEvalDaily.tenantId as
any).eq(tenantId));
    res.json({ rows });
  });

  app.get("/api/ai/entities", async (req, res) => {
    const tenantId = ctx(req);
    const ents:any = await db.execute(`select id, type, value from ai_entities where tenant_id=$1`,
[tenantId]);
    res.json({ entities: ents });
  });

  app.get("/api/ai/entities/links", async (req, res) => {
    const tenantId = ctx(req);
    const links:any = await db.execute(`
      select l.src_id, l.dst_id, l.weight
      from ai_entity_links l where tenant_id=$1
    `, [tenantId]);
    res.json({ links });
  });
}
```

/server/ai/routes.knowledge.ts (explorer com busca guiada — stub simples para v1 do painel)

```
import type { Express, Request } from "express";
import { parseURL } from "./parser.full";
import { ingestText } from "./ingest";

function ctx(req: Request){ return (req as any).tenantId || process.env.PRIMARY_TENANT_ID!; }

export function registerKnowledgeRoutes(app: Express) {
  // Buscar uma lista de URLs (pré-selecionadas pelo admin) e retornar prévias para curadoria
  app.post("/api/ai/knowledge/preview", async (req, res) => {
    const { urls=[] } = req.body || {};
    const out:any[] = [];
    for (const u of urls) {
      try{
```

```
        const { text, title, meta } = await parseURL(u);
        out.push({ url: u, ok: true, title, excerpt: text.slice(0, 600), meta });
      } catch(e:any) {
        out.push({ url: u, ok: false, error: e?.message });
      }
    }
    res.json({ previews: out });
  });

  // Ingerir selecionados
  app.post("/api/ai/knowledge/ingest", async (req, res) => {
    const tenantId = ctx(req);
    const { items=[] } = req.body || {}; // [{url,title,meta,approved:true}]
    const results:any[] = [];
    for (const it of items) {
      if (!it.approved) continue;
      try{
        const { text, title, meta } = await parseURL(it.url);
        const r = await ingestText(tenantId, { source: "url", uri: it.url, title: it.title || title,
text, meta });
        results.push({ url: it.url, ok: true, r });
      } catch(e:any) {
        results.push({ url: it.url, ok: false, error: e?.message });
      }
    }
    res.json({ results });
  });
}
```

Registrar:

```
import { registerMetricsRoutes } from "./ai/routes.metrics";
import { registerKnowledgeRoutes } from "./ai/routes.knowledge";

export function registerRoutes(app: Express) {
  app.use("/api", (req, _res, next) => { (req as any).tenantId = process.env.PRIMARY_TENANT_ID!;
next(); });
  // ...
  registerMetricsRoutes(app);
  registerKnowledgeRoutes(app);
}
```

---

1) Páginas Next.js

1.1 `/admin/metrics` — gráficos (nDCG/MRR/CTR/CR)

/ui/pages/admin/metrics.tsx

```
import { useEffect, useState } from "react";
import dynamic from "next/dynamic";
const { Line } = dynamic(() => import("react-chartjs-2").then(m => ({ default: m.Line })), { ssr:false
}) as any;

export default function MetricsPage(){
  const [data,setData] = useState<any[]>([]);
  useEffect(()=>{ (async()=>{
    const r = await fetch("/api/ai/metrics/daily"); const j = await r.json();
    setData(j.rows||[]);
  })(); },[]);
  const labels = data.map((r:any)=>r.day);
  const ndcg  = data.map((r:any)=>r.ndcgAt5);
  const mrr   = data.map((r:any)=>r.mrr);
  const ctr   = data.map((r:any)=>r.ctr);
  const cr    = data.map((r:any)=>r.cr);

  const mk = (label:string, arr:number[]) => ({
    label, data: arr, tension: 0.25, fill:false
  });
```

```
  return (
    <div className="p-6 space-y-10">
      <h1 className="text-2xl font-bold">IA · Métricas Diárias</h1>
      <section>
        <h2 className="font-semibold mb-2">nDCG@5 / MRR</h2>
        <Line data={{ labels, datasets: [mk("nDCG@5", ndcg), mk("MRR", mrr)] }} />
      </section>
      <section>
        <h2 className="font-semibold mb-2">Funil · CTR / CR</h2>
        <Line data={{ labels, datasets: [mk("CTR", ctr), mk("CR", cr)] }} />
      </section>
    </div>
  );
}
```

Observação: se não tiver Chart.js configurado, instale `chart.js` e `react-chartjs-2`.

```
npm i chart.js react-chartjs-2
```

No `_app.tsx` ou página, registre os chart controllers conforme a doc do Chart.js.

1.2 `/admin/entities` — heatmap (grafo de entidades)

/ui/pages/admin/entities.tsx

```
import { useEffect, useMemo, useState } from "react";

type Ent = { id:string; type:string; value:string };
type Link = { src_id:string; dst_id:string; weight:number };

export default function EntitiesPage(){
  const [ents,setEnts] = useState<Ent[]>([]);
  const [links,setLinks] = useState<Link[]>([]);
  useEffect(()=>{ (async()=>{
    const a = await fetch("/api/ai/entities").then(r=>r.json());
    const b = await fetch("/api/ai/entities/links").then(r=>r.json());
    setEnts(a.entities||[]); setLinks(b.links||[]);
  })(); },[]);

  const map = useMemo(()=>{
    const byId = new Map<string, Ent>();
    for (const e of ents) byId.set(e.id, e as any);
    const edges = links.map(l => ({ s: byId.get(l.src_id)!, d: byId.get(l.dst_id)!, w: l.weight }));
    return { byId, edges };
  }, [ents, links]);

  // Heatmap simplificado: tabela de top pares por peso
  const top = useMemo(()=> [...map.edges].sort((a,b)=>b.w-a.w).slice(0,50), [map.edges]);

  return (
    <div className="p-6 space-y-6">
      <h1 className="text-2xl font-bold">IA · Entidades & Relações</h1>
      <p className="text-sm text-gray-600">Top relações por co-ocorrência (peso).</p>
      <table className="min-w-full text-sm">
        <thead>
          <tr className="text-left border-b"><th>Origem</th><th>Destino</th><th>Peso</th></tr>
        </thead>
        <tbody>
          {top.map((e,i)=>(
            <tr key={i} className="border-b">
              <td>{e.s?.value} <span className="text-gray-400">({e.s?.type})</span></td>
              <td>{e.d?.value} <span className="text-gray-400">({e.d?.type})</span></td>
              <td>{e.w}</td>
            </tr>
          ))}
        </tbody>
      </table>
    </div>
```

```
  );
}
```

1.3 `/admin/knowledge` — explorer (pré-visualização + ingest aprovado)

/ui/pages/admin/knowledge.tsx

```
import { useState } from "react";

export default function KnowledgePage(){
  const [urls,setUrls] = useState<string>("");
  const [previews,setPreviews] = useState<any[]>([]);
  const [selected,setSelected] = useState<Record<string, boolean>>({});

  async function preview(){
    const list = urls.split(/\s+/).filter(Boolean);
    const r = await fetch("/api/ai/knowledge/preview", { method:"POST", headers:{"Content-Type":"
application/json"}, body: JSON.stringify({ urls:list }) });
    const j = await r.json(); setPreviews(j.previews||[]);
    const sel:Record<string,boolean> = {}; for (const p of j.previews||[]) sel[p.url]=true;
setSelected(sel);
  }
  async function ingest(){
    const items = previews.map(p => ({ url: p.url, title: p.title, meta: p.meta, approved:
!!selected[p.url] }));
    await fetch("/api/ai/knowledge/ingest", { method:"POST", headers:{"Content-Type":"
application/json"}, body: JSON.stringify({ items }) });
    alert("Ingestão enviada.");
  }

  return (
    <div className="p-6 space-y-6">
      <h1 className="text-2xl font-bold">IA · Knowledge Explorer</h1>
      <div className="space-y-2">
        <textarea className="w-full border p-2 h-28" placeholder="Cole URLs (uma ou várias, separadas
por espaço ou quebra de linha)" value={urls} onChange={e=>setUrls(e.target.value)} />
        <div className="flex gap-3">
          <button className="px-4 py-2 bg-blue-600 text-white rounded" onClick={preview}>Pré-
visualizar</button>
          <button className="px-4 py-2 bg-emerald-600 text-white rounded" onClick={ingest}>Ingerir
selecionados</button>
        </div>
      </div>

      <div className="space-y-4">
        {previews.map((p:any)=>(
          <div key={p.url} className="border p-3 rounded">
            <div className="flex items-center justify-between">
              <div>
                <div className="font-semibold">{p.title || p.url}</div>
                <div className="text-xs text-gray-500">{p.url}</div>
              </div>
              <label className="flex items-center gap-2">
                <input type="checkbox" checked={!!selected[p.url]} onChange={e=>setSelected(s=>({...s,
[p.url]:e.target.checked}))} />
                <span className="text-sm">Aprovar</span>
              </label>
            </div>
            <p className="text-sm mt-2 whitespace-pre-wrap">{p.ok ? (p.excerpt || "").slice(0,600) :
`Erro: ${p.error}`}</p>
          </div>
        ))}
      </div>
    </div>
  );
}
```

Se seu Next já tem layout/menu, só adicione links para: `/admin/metrics`, `/admin/entities`, `/admin/knowledge`.

2) Rotas “ask” usando o **score completo**

Substitua o endpoint de consulta para usar `hybridRetrieveFull`:

/server/ai/routes.ts (atualize o `/ask.base` → `/ask.full`)

```
import { hybridRetrieveFull } from "./hybrid-score";

app.post("/api/ai/ask.full", async (req, res) => {
  try {
    const c = ctx(req);
    const { query, k=12 } = req.body || {};
    if (!query) return res.status(400).json({ error: "query required" });
    const r = await hybridRetrieveFull(c.tenantId, query, undefined, k);
    res.json(r);
  } catch (e:any) { res.status(400).json({ error: e?.message }); }
});
```

Sanidade final e operação

1. **Instale deps** novas e rode **migrations** de todos os schemas (core + graph + eval).
2. Garanta os **modelos ONNX** no `/models` (texto e visão).
3. Reinicie o servidor.
4. Testes rápidos:
    - **Ingest URL** (PDF/HTML/DOCX/PPTX/CSV): `POST /api/ai/ingest.url`
    - **Ingest arquivo** (inclui OCR para imagens): `POST /api/ai/ingest.file`
    - **Consultar (score completo)**: `POST /api/ai/ask.full`
    - **Painel Admin**: visite `/admin/metrics`, `/admin/entities`, `/admin/knowledge`.
