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

IA_Autonoma_Parte7

1 mensagem

Fillipe Guerra <fillipe.backup@gmail.com> 28 de outubro de 2025 às 09:49
Para: Fillipe Augusto Gomes Guerra <fillipe182@hotmail.com>, Fillipe Guerra <fillipe.backup@gmail.com>

AION **nível perfeição** com:

- **CRUD completo da Knowledge Base** (editar conhecimento existente, versionar, soft-delete/restore).
- **Ingest multimodal via painel**: upload de **texto, imagem e vídeo** (com OCR e ASR locais).
- **"Aprender por tema/link"** direto no painel: você dá um **tema ou link**, ela **descobre** → **pré-visualiza** → **você aprova** → **ingere** na KB local.
- **Config e Telemetria** no Admin: fallback, orçamento, confiança, **logs de fallback**, **auditoria de KB**.
- **Base matemática e thresholds** para decisão, frescor e autoridade.

É cumulativo com os apêndices que já te mandei: trate isto como **Apêndices AB (KB CRUD & Upload), AC (Vídeo/ASR), AD (Learn-by-Theme), AE (Telemetria & Migrações)**.

Se algo já existir no seu repo, **substitua** pela versão daqui (é retrocompatível com single-tenant).

APÊNDICE AB — Knowledge Base CRUD & Upload (texto/imagem)

AB.1 Migrações (versão/soft-delete/auditoria)

```
/shared/schema.ai.kb.ts

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

export const aiDocuments = pgTable("ai_documents", {
  id: uuid("id").primaryKey().defaultRandom(),
  tenantId: varchar("tenant_id", { length: 64 }).notNull(),
  source: varchar("source", { length: 32 }).notNull(),      // url | file | ocr | video | manual
  uri: varchar("uri", { length: 512 }).notNull(),
  title: varchar("title", { length: 256 }),
  metaJson: jsonb("meta_json").$type<any>().default({}),
  version: integer("version").notNull().default(1),
  isDeleted: boolean("is_deleted").notNull().default(false),
  createdAt: timestamp("created_at").defaultNow().notNull(),
  updatedAt: timestamp("updated_at").defaultNow().notNull()
}, t => ({
  idx: index("ai_docs_tenant_uri_idx").on(t.tenantId, t.uri)
}));

export const aiChunks = pgTable("ai_chunks", {
  id: uuid("id").primaryKey().defaultRandom(),
  tenantId: varchar("tenant_id", { length: 64 }).notNull(),
  documentId: uuid("document_id").notNull(),
  order: integer("order").notNull().default(0),
  text: text("text"),
  vector: jsonb("vector").$type<number[] | null>().default(null),
  imageVec: jsonb("image_vec").$type<number[] | null>().default(null),
  metaJson: jsonb("meta_json").$type<any>().default({}),
  createdAt: timestamp("created_at").defaultNow().notNull()
}, t => ({
  idx: index("ai_chunks_doc_idx").on(t.documentId)
}));

export const aiKbAudit = pgTable("ai_kb_audit", {
  id: uuid("id").primaryKey().defaultRandom(),
  tenantId: varchar("tenant_id", { length: 64 }).notNull(),
  actor: varchar("actor", { length: 128 }).notNull(),       // "admin:dani" etc
  action: varchar("action", { length: 32 }).notNull(),      // create|update|delete|restore|
ingest|edit-chunk
  documentId: uuid("document_id"),
  chunkId: uuid("chunk_id"),
  payload: jsonb("payload").$type<any>().default({}),
  createdAt: timestamp("created_at").defaultNow().notNull()
}, t => ({
  idx: index("ai_kb_audit_tenant_idx").on(t.tenantId, t.createdAt)
}));
```

Se já tiver tabelas com nomes iguais, ajuste apenas os campos novos (version/isDeleted/audit).

AB.2 Rotas de **CRUD** e **Upload** (texto/imagem) — Admin

```
/server/ai/routes.kb.ts

import type { Express, Request } from "express";
import multer from "multer";
import { db } from "../db";
import { aiDocuments, aiChunks, aiKbAudit } from "@shared/schema.ai.kb";
import { eq, and } from "drizzle-orm";
import { parseLocalFile } from "./parser.full";
import { parseImage } from "./parser.multimodal";
import fs from "fs";
import path from "path";

const upload = multer({ dest: "./uploads" });
function T(req: Request){ return (req as any).tenantId || process.env.PRIMARY_TENANT_ID!; }
async function audit(tenantId:string, action:string, payload:any, ids?:{documentId?:string;
chunkId?:string}){
  await db.insert(aiKbAudit).values({ tenantId, actor: "admin", action, payload, ...ids });
}

// LIST
export function registerKbRoutes(app: Express) {
  app.get("/api/ai/kb/list", async (req,res)=>{
    const tenantId = T(req);
    const docs = await db.select().from(aiDocuments).where(eq(aiDocuments.tenantId, tenantId));
    res.json(docs);
  });

  // GET document with chunks
  app.get("/api/ai/kb/doc/:id", async (req,res)=>{
    const tenantId = T(req); const id = req.params.id;
    const [doc] = await db.select().from(aiDocuments).where(and(eq(aiDocuments.tenantId, tenantId),
eq(aiDocuments.id, id)));
    if (!doc) return res.status(404).json({ error: "not found" });
    const chunks = await db.select().from(aiChunks).where(eq(aiChunks.documentId,
id)).orderBy(aiChunks.order as any);
    res.json({ doc, chunks });
  });

  // CREATE / UPDATE (text) — manual editor
  app.post("/api/ai/kb/upsert", async (req,res)=>{
    const tenantId = T(req);
    const { id, title, uri, text, meta={} } = req.body || {};
    if (!text || !uri) return res.status(400).json({ error:"uri & text required" });

    if (!id) {
      // create
      const [doc] = await db.insert(aiDocuments).values({
        tenantId, source:"manual", uri, title, metaJson: meta
      }).returning();
      await db.insert(aiChunks).values({ tenantId, documentId: doc.id, order: 0, text, metaJson: {}
});
      await audit(tenantId, "create", { uri, title }, { documentId: doc.id });
      return res.json({ ok:true, id: doc.id });
    } else {
      // update: bump version, replace first chunk text (simple editor)
      const [docOld] = await db.select().from(aiDocuments).where(and(eq(aiDocuments.tenantId,
tenantId), eq(aiDocuments.id, id)));
      if (!docOld) return res.status(404).json({ error:"not found" });
      await db.update(aiDocuments).set({ title, uri, metaJson: meta, version: (docOld.version||1)+1,
updatedAt: new Date() })
        .where(eq(aiDocuments.id, id));
      const [first] = await db.select().from(aiChunks).where(eq(aiChunks.documentId,
id)).orderBy(aiChunks.order as any).limit(1);
      if (first) {
        await db.update(aiChunks).set({ text }).where(eq(aiChunks.id, first.id));
      } else {
        await db.insert(aiChunks).values({ tenantId, documentId: id, order:0, text });
      }
      await audit(tenantId, "update", { id, uri, title }, { documentId: id });
      return res.json({ ok:true, id });
    }
  });

  // SOFT DELETE / RESTORE
  app.post("/api/ai/kb/delete", async (req,res)=>{
    const tenantId = T(req); const { id } = req.body || {};
    await db.update(aiDocuments).set({ isDeleted: true, updatedAt: new Date()
}).where(and(eq(aiDocuments.tenantId, tenantId), eq(aiDocuments.id, id)));
    await audit(tenantId, "delete", { id }, { documentId: id });
    res.json({ ok:true });
  });
  app.post("/api/ai/kb/restore", async (req,res)=>{
    const tenantId = T(req); const { id } = req.body || {};
    await db.update(aiDocuments).set({ isDeleted: false, updatedAt: new Date()
}).where(and(eq(aiDocuments.tenantId, tenantId), eq(aiDocuments.id, id)));
    await audit(tenantId, "restore", { id }, { documentId: id });
    res.json({ ok:true });
  });

  // UPLOAD (texto/imagem) com OCR/CLIP dependendo do tipo
  app.post("/api/ai/kb/upload", upload.single("file"), async (req,res)=>{
    const tenantId = T(req);
    if (!req.file) return res.status(400).json({ error:"file required" });
    const fp = path.resolve(req.file.path);
    const name = req.file.originalname.toLowerCase();

    try {
      if (/\.(png|jpe?g|webp)$/i.test(name)) {
        const buf = fs.readFileSync(fp);
        const parsed = await parseImage(buf, true);
        // cria doc base (imagem) + chunk OCR (se houver) — vetores tratados em ingest/background
        const [doc] = await db.insert(aiDocuments).values({
          tenantId, source:"file", uri: req.file.originalname, title: req.file.originalname, metaJson:
parsed.meta
        }).returning();
        const text = parsed.ocrText || "";
        await db.insert(aiChunks).values({
          tenantId, documentId: doc.id, order:0, text, metaJson: { clipVec: parsed.clipVec }
        });
        await audit(tenantId, "ingest", { type:"image", uri:req.file.originalname }, { documentId:
doc.id });
        return res.json({ ok:true, id: doc.id });
      } else {
        // pdf/docx/pptx/csv/txt/html
        const { text, title, meta } = await parseLocalFile(fp);
        const [doc] = await db.insert(aiDocuments).values({
          tenantId, source:"file", uri: req.file.originalname, title: title || req.file.originalname,
metaJson: meta
        }).returning();
        await db.insert(aiChunks).values({ tenantId, documentId: doc.id, order:0, text });
        await audit(tenantId, "ingest", { type:"file", uri:req.file.originalname }, { documentId:
doc.id });
        return res.json({ ok:true, id: doc.id });
      }
    } finally {
      fs.unlinkSync(fp);
    }
  });
}
```

Registre no bootstrap:

```
import { registerKbRoutes } from "./ai/routes.kb";
registerKbRoutes(app);
```

AB.3 Página Admin 🆎 (CRUD + Upload)

`/ui/pages/admin/kb.tsx`

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

export default function KbPage(){
  const [docs,setDocs] = useState<any[]>([]);
  const [sel,setSel] = useState<any|null>(null);
  const [editor,setEditor] = useState({ id:"", uri:"", title:"", text:"", meta:{} as any });
  const [file,setFile] = useState<File|null>(null);

  async function load(){
    const r = await fetch("/api/ai/kb/list"); const j = await r.json();
    setDocs(j||[]);
  }
  useEffect(()=>{ load(); },[]);

  async function open(id:string){
    const r = await fetch(`/api/ai/kb/doc/${id}`); const j = await r.json();
    setSel(j);
    const text = j.chunks?.[0]?.text || "";
    setEditor({ id:j.doc.id, uri:j.doc.uri||"", title:j.doc.title||"", text, meta:j.doc.metaJson||{}
});
  }
  async function save(){
    await fetch("/api/ai/kb/upsert", { method:"POST", headers:{ "Content-Type":"application/json" },
body: JSON.stringify(editor) });
    await load();
  }
  async function del(id:string){
    await fetch("/api/ai/kb/delete", { method:"POST", headers:{ "Content-Type":"application/json" },
body: JSON.stringify({ id }) });
    await load();
  }
  async function restore(id:string){
    await fetch("/api/ai/kb/restore", { method:"POST", headers:{ "Content-Type":"application/json" },
body: JSON.stringify({ id }) });
    await load();
  }
  async function uploadFile(){
    if (!file) return;
    const fd = new FormData(); fd.append("file", file);
    await fetch("/api/ai/kb/upload", { method:"POST", body: fd });
    setFile(null); await load();
  }

  return (
    <div className="p-6 grid grid-cols-3 gap-6">
      <div className="col-span-1">
        <h1 className="text-xl font-bold mb-3">Knowledge Base</h1>
        <div className="space-y-2">
          <div className="border rounded p-3">
            <div className="font-semibold mb-2">Upload</div>
            <input type="file" onChange={e=>setFile(e.target.files?.[0]||null)} />
            <button className="ml-2 px-3 py-1 bg-emerald-600 text-white rounded" onClick=
{uploadFile}>Enviar</button>
          </div>
          <div className="border rounded p-3">
            <div className="font-semibold mb-2">Documentos</div>
            <ul className="space-y-1 max-h-[60vh] overflow-auto">
              {docs.map((d:any)=>(
                <li key={d.id} className="flex items-center justify-between">
                  <button className="text-blue-700 underline" onClick={()=>open(d.id)}>{d.title ||
d.uri}</button>
                  {d.isDeleted
                    ? <button className="text-xs text-emerald-700" onClick={()=>restore(d.id)}>
restaurar</button>
                    : <button className="text-xs text-rose-700" onClick={()=>del(d.id)}>
remover</button>}
                </li>
              ))}
            </ul>
          </div>
        </div>
      </div>

      <div className="col-span-2">
        <h2 className="text-lg font-semibold mb-2">Editor</h2>
        <div className="space-y-2">
          <input className="border p-2 w-full" placeholder="URI" value={editor.uri} onChange=
{e=>setEditor({...editor,uri:e.target.value})}/>
          <input className="border p-2 w-full" placeholder="Título" value={editor.title} onChange=
{e=>setEditor({...editor,title:e.target.value})}/>
          <textarea className="border p-2 w-full h-64" placeholder="Texto do documento" value=
{editor.text} onChange={e=>setEditor({...editor,text:e.target.value})}/>
          <button className="px-4 py-2 bg-blue-600 text-white rounded" onClick={save}>Salvar</button>
        </div>
        {sel && (
          <div className="mt-6">
            <div className="text-sm text-gray-500">Versão: {sel.doc.version} · Deletado:
{String(sel.doc.isDeleted)}</div>
          </div>
        )}
      </div>
    </div>
  );
}
```

No menu admin, adicione **“IA → Knowledge Base”** apontando para `/admin/kb`.

APÊNDICE AC — **Vídeo** (frames + OCR + **ASR local**)

- **Frames**: amostragem 1 fps → OCR (Tesseract) + CLIP (para busca visual).
- **Audio/ASR**: extração de áudio via `ffmpeg` + **Whisper local** (via `faster-whisper`, CPU OK em curtas; se pesado, rode como job offline).

AC.1 Dependências

```
# sistema
apt-get update && apt-get install -y ffmpeg

# node deps (já tem sharp/onnx)
```

```
pnpm add fluent-ffmpeg

# python (asr)
pip3 install faster-whisper
```

AC.2 Ingest de vídeo (rota)

```
/server/ai/routes.kb.video.ts

import type { Express, Request } from "express";
import multer from "multer";
import ffmpeg from "fluent-ffmpeg";
import fs from "fs";
import path from "path";
import { db } from "../db";
import { aiDocuments, aiChunks, aiKbAudit } from "@shared/schema.ai.kb";
import { parseImage } from "./parser.multimodal";
import { spawn } from "child_process";

const upload = multer({ dest: "./uploads" });
function T(req: Request){ return (req as any).tenantId || process.env.PRIMARY_TENANT_ID!; }
async function audit(tenantId:string, action:string, payload:any, ids?:{documentId?:string;
chunkId?:string}){
  await db.insert(aiKbAudit).values({ tenantId, actor:"admin", action, payload, ...ids });
}

export function registerKbVideoRoute(app: Express){
  app.post("/api/ai/kb/upload.video", upload.single("file"), async (req,res)=>{
    if (!req.file) return res.status(400).json({ error:"file required" });
    const tenantId = T(req);
    const tmp = path.resolve(req.file.path);
    const base = path.basename(req.file.originalname, path.extname(req.file.originalname));
    const work = `./uploads/${base}_${Date.now()}`; fs.mkdirSync(work, { recursive:true });

    // 1) Extrai áudio (.wav) e frames (1 fps)
    const wav = path.join(work, "audio.wav");
    const frames = path.join(work, "frames"); fs.mkdirSync(frames, { recursive:true });

    await new Promise<void>((resolve, reject)=>{
      ffmpeg(tmp).outputOptions(["-ar 16000","-ac 1"]).save(wav).on("end", ()=>resolve()).on("error",
reject);
    });
    await new Promise<void>((resolve, reject)=>{
      ffmpeg(tmp).output(path.join(frames, "f_%05d.png")).outputOptions(["-vf","fps=1"]).on("end",
()=>resolve()).on("error", reject).run();
    });

    // 2) Cria documento de vídeo
    const [doc] = await db.insert(aiDocuments).values({
      tenantId, source:"video", uri:req.file.originalname, title:req.file.originalname, metaJson:{
type:"video" }
    }).returning();

    // 3) ASR (Python: faster-whisper) — retorna JSON com textos e timestamps
    const asrOut = path.join(work, "asr.json");
    await new Promise<void>((resolve, reject)=>{
      const p = spawn("python3", ["./trainer/asr/transcribe.py", wav, asrOut], { stdio:"inherit" });
      p.on("exit", code => code===0 ? resolve() : reject(new Error("asr failed")));
    });
    const asr = JSON.parse(fs.readFileSync(asrOut,"utf8"));
    const transcript = asr.text || "";
    await db.insert(aiChunks).values({
      tenantId, documentId: doc.id, order: 0, text: transcript, metaJson: { kind:"asr", segments:
asr.segments?.slice(0,30) || [] }
    });

    // 4) Frames → OCR + CLIP
    const files = fs.readdirSync(frames).filter(f=>/\.png$/i.test(f)).slice(0, 300);
    let order = 1;
    for (const f of files) {
      const buf = fs.readFileSync(path.join(frames, f));
      const parsed = await parseImage(buf, true);
      await db.insert(aiChunks).values({
        tenantId, documentId: doc.id, order: order++,
        text: parsed.ocrText || "",
        metaJson: { keyframe: f, clipVec: parsed.clipVec, w: parsed.width, h: parsed.height }
      });
    }

    await audit(tenantId, "ingest", { type:"video", uri:req.file.originalname }, { documentId: doc.id
});

    // limpeza
    fs.unlinkSync(tmp);
    res.json({ ok:true, id: doc.id, frames: files.length, transcriptLen: transcript.length });
  });
}
```

AC.3 Script ASR (Python)

```
/trainer/asr/transcribe.py

import sys, json
from faster_whisper import WhisperModel

wav_in = sys.argv[1]
out_json = sys.argv[2]

# modelo pequeno para CPU; troque por medium/large se tiver GPU
model = WhisperModel("small", device="cpu", compute_type="int8")
segments, info = model.transcribe(wav_in, vad_filter=True)

out = {"language": info.language, "duration": info.duration, "segments": [], "text": ""}
txt = []
for s in segments:
    seg = {"start": s.start, "end": s.end, "text": s.text.strip()}
    out["segments"].append(seg); txt.append(seg["text"])
out["text"] = " ".join(txt)

with open(out_json, "w") as f:
    json.dump(out, f, ensure_ascii=False)
```

> **Pronto**: vídeo agora alimenta a KB com **texto (ASR)** e **visão (frames OCR/CLIP)**, tudo **local**.

APÊNDICE AD — Learn-by-Theme (tema ou link → aprender e salvar)

Já tínhamos “Discover/Preview/Ingest” por **seed URL** (crawler). Agora adicionamos **“Tema”**:

- Você insere um **tema** (ex.: “tuk-tuk Sintra segurança”),
- A AION gera **consultas canônicas** e **seeds** (p.ex., domínio do cliente ou fontes permitidas),
- Crawler roda **apenas** nos domínios permitidos (política/ética),
- Preview → aprova → ingere.

AD.1 Rota

```
/server/ai/curator/theme.routes.ts

import type { Express } from "express";
import { crawlSeed } from "./crawler";
import { parseURL } from "../parser.full";
import { basicPolicy } from "./filters.policy";

// simples gerador de seeds por tema. Em produção, amarre a domínios whitelisted.
function seedsForTheme(theme:string){
  const base = (process.env.AION_THEME_BASE || "").split(",").filter(Boolean);
  // ex: AION_THEME_BASE=https://www.yesyoudeserve.tours,https://www.visitportugal.com
  const paths = ["", "/blog", "/faq", "/terms", "/about"];
  const list:string[] = [];
  for (const b of base) for (const p of paths) list.push(new URL(p, b).toString());
  return list;
}

export function registerThemeRoutes(app: Express){
  app.post("/api/ai/curator/theme.discover", async (req,res)=>{
    const { theme } = req.body || {};
    if (!theme) return res.status(400).json({ error:"theme required" });
    const seeds = seedsForTheme(theme).slice(0, 5);
    let urls: string[] = [];
    for (const s of seeds) {
      const u = await crawlSeed(s);
      urls = urls.concat(u);
    }
    res.json({ seeds, urls });
  });

  app.post("/api/ai/curator/theme.preview", async (req,res)=>{
    const { theme, urls=[] } = req.body || {};
    const out:any[] = [];
    for (const u of urls.slice(0,200)) {
      try{
        const { text, title, meta } = await parseURL(u);
        const ok = basicPolicy(u, text.slice(0,800));
        // filtro adicional: contém alguma palavra do tema?
        const hit = new RegExp(theme.split(/\s+/).filter(Boolean).join("|"), "i").test(text);
        out.push({ url:u, ok: ok.ok && hit, title, excerpt:text.slice(0,800), meta, reason: ok.reason
|| (!hit && "sem match de tema") });
      } catch (e:any){ out.push({ url:u, ok:false, error:e?.message }); }
    }
    res.json({ previews: out });
  });
}
```

Registre no bootstrap:

```
import { registerThemeRoutes } from "./ai/curator/theme.routes";
registerThemeRoutes(app);
```

AD.2 UI: seção "Aprender por Tema"

Em `/ui/pages/admin/knowledge.tsx`, adicione outro bloco (sem remover o que já existe):

```
// dentro do componente, acrescente estados:
const [theme,setTheme] = useState("");
const [tSeeds,setTSeeds] = useState<string[]>([]);
const [tFound,setTFound] = useState<string[]>([]);

async function themeDiscover(){
  const r = await fetch("/api/ai/curator/theme.discover", { method:"POST", headers:{ "Content-
Type":"application/json" }, body: JSON.stringify({ theme }) });
  const j = await r.json(); setTSeeds(j.seeds||[]); setTFound(j.urls||[]);
}
async function themePreview(){
  const r = await fetch("/api/ai/curator/theme.preview", { method:"POST", headers:{ "Content-
Type":"application/json" }, body: JSON.stringify({ theme, urls: tFound }) });
  const j = await r.json(); setPreviews(j.previews||[]);
}
```

E o bloco visual:

```
<div className="border p-4 rounded space-y-2">
  <h2 className="font-semibold">Aprender por Tema</h2>
  <div className="flex gap-2">
    <input className="flex-1 border p-2" placeholder="Ex.: tuk-tuk Sintra segurança"
           value={theme} onChange={e=>setTheme(e.target.value)} />
    <button className="px-3 py-2 bg-indigo-600 text-white rounded" onClick={themeDiscover}>
Descobrir</button>
    <button className="px-3 py-2 bg-slate-700 text-white rounded" onClick={themePreview}>Pré-
visualizar</button>
  </div>
  {!!tSeeds.length && <p className="text-xs text-gray-500">Seeds: {tSeeds.join(", ")}</p>}
  {!!tFound.length && <p className="text-xs text-gray-500">Encontradas: {tFound.length} URLs</p>}
</div>
```

> **Política/ética/domínios**: controle via `AION_THEME_BASE` no `.env` para restringir a onde a AION "pode" aprender (clientes, parceiros, sites próprios).

APÊNDICE AE — Telemetria, Fallback Logs & Matemática de Confiança

AE.1 Tabelas

```
/shared/schema.ai.telemetry.ts

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

export const aiFallbackLogs = pgTable("ai_fallback_logs", {
  id: uuid("id").primaryKey().defaultRandom(),
  tenantId: varchar("tenant_id", { length: 64 }).notNull(),
  reason: varchar("reason", { length: 32 }).notNull(),     // low_confidence | rate_limit | budget
  queryHash: varchar("query_hash", { length: 64 }).notNull(),
  confidence: real("confidence").notNull(),
  costEstUSD: real("cost_est_usd").notNull().default(0),
  tokensEst: integer("tokens_est").notNull().default(0),
  createdAt: timestamp("created_at").defaultNow().notNull(),
  metaJson: jsonb("meta_json").$type<any>().default({})
}, t => ({
  idx: index("ai_fb_logs_tenant_idx").on(t.tenantId, t.createdAt)
}));
```

AE.2 Hooks no driver (registrar fallbacks)

No seu `generateFallback` (em `llm.ts`), **após** `registerSpend(estUsd)`:

```
import crypto from "crypto";
import { db } from "../db";
import { aiFallbackLogs } from "@shared/schema.ai.telemetry";

async function logFallback(tenantId:string, reason:string, prompt:string, confidence:number,
est:number, tok:number){
  const queryHash = crypto.createHash("sha256").update(prompt.slice(0,2000)).dig
est("hex").slice(0,64);
  await db.insert(aiFallbackLogs).values({
    tenantId, reason, queryHash, confidence, costEstUSD: est, tokensEst: tok, metaJson: {}
  });
}
```

E chame `logFallback(tenantId, "low_confidence", prompt, conf, estUsd, (opts.maxTokens ?? MAXTOK))` no retorno do fallback.
(Se bloquear por budget/rate, registre `"budget"`/`"rate_limit"`.)

AE.3 Painel (telemetria)

Crie `/ui/pages/admin/ai-telemetry.tsx` com dois cards simples:

- **Fallback rate por dia** (bar chart)
- **Histograma de confiança** (bins 0-0.2…0.8-1.0)

(Para manter foco, deixo como TODO curto — se quiser, mando o código do gráfico já plugado no seu `/api/ai/metrics/daily` + nova rota `/api/ai/fallback/stats`.)

Matemática dos thresholds (resumo prático)

1. **Confiança** C∈[0,1] agregada dos componentes do retrieval (texto, grafo, frescor, autoridade):

C=wtst+wgsg+wfsf+wasa+wisi,∑wj=1.

Sugerido: wt=0.55, wg=0.25, wf=0.10, wa=0.05, wi=0.05.

2. **Decisão de fallback** (controlada no painel):
   Se C<τ **e** orçamento & rate disponíveis → **fallback**; senão → local.

3. **Frescor** (meia-vida h dias):

sf=exp(−ln2·hageDays)

(h ajustável na UI de métricas — já usamos como env no parser).

4. **Autoridade**: normalizada [0,1] a partir de metadados de fonte (rank manual/heurístico por tipo). Você pode editar no painel KB (campo `metaJson.sourceRank`).

Integração final (checklist)

1. **Adicionar arquivos** dos apêndices AB, AC, AD, AE (mais os que já te mandei antes).

2. **Registrar rotas** no `server/routes.ts`:

```
registerKbRoutes(app);
registerKbVideoRoute(app);
registerCuratorRoutes(app);
registerThemeRoutes(app);
registerAiSettingsRoutes(app);
registerTrainRoutes(app);
registerAnswerRoute(app);
```

3. **Menu Admin**:
   - IA → **Knowledge Base** (`/admin/kb`)
   - IA → **Fallback & Orçamento** (`/admin/ai-settings`)
   - IA → **Métricas** (`/admin/metrics`)
   - IA → **Entidades** (`/admin/entities`)
   - IA → **Aprender (Knowledge)** (`/admin/knowledge`)
   - IA → **Telemetria** (`/admin/ai-telemetry`) *(opcional agora)*

4. **.env** (aditivos):

```
AION_THEME_BASE=https://www.yesyoudeserve.tours
AION_CRAWL_MAX_PAGES=50
AION_OCR_LANG=eng+por
USE_OPENAI=true
```

5. **Dependências** (caso não tenha):

```
pnpm add multer fluent-ffmpeg
apt-get update && apt-get install -y ffmpeg
pip3 install faster-whisper
```

Fechou. Com isso você:

- **Edita** qualquer conhecimento e **versiona** no painel,
- **Adiciona** conhecimento por **upload (texto/imagem/vídeo)**, com OCR/CLIP/ASR **locais**,
- **Ensina por Tema ou Link** com curadoria,
- Controla **fallback/orçamento/confiança** no Admin, com **logs e auditoria**,
- E mantém a AION **aprendendo continuamente** e usando **cada vez menos** serviços externos.