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

IA_Autonoma_Parte11

1 mensagem

Fillipe Guerra <fillipe.backup@gmail.com> 28 de outubro de 2025 às 10:53
Para: Fillipe Augusto Gomes Guerra <fillipe182@hotmail.com>, Fillipe Guerra <fillipe.backup@gmail.com>

Se algum arquivo já existir dos meus envios anteriores, **substitua** pelo daqui (são supersets “state-of-the-art”).

🚀 FASE 4 — Aprendizado Autônomo / Curadoria Web (UI Final, robusta)

0) Visão do pipeline (end-to-end)

Tema/Link → Descoberta (sitemaps + SERP) → Coleta (HTTP/JS) → Normalização (Readability) → Regras/Ética → Dedup/Canon. → Scoring (qualidade/frescor/autoridade) → Pré-visualização (UI) → Aprovação/Curadoria → Chunking/Embedding → ANN incremental → Telemetria.

- **Restrições**: whitelists de domínios, respeito a `robots.txt`, limites por host (rate limit), janela de profundidade, desindexação sob demanda.
- **Qualidade**: heurísticas de “boilerplate removal”, linguagem, entropia, compressibilidade, **frescor** (meia-vida).
- **Curadoria**: aprova/edita/descarta em lote, com **dif de texto** e versão do doc.
- **Aprendizado ativo**: a cada sprint, seleciona *k* exemplos com maior incerteza (C≈τ) para revisão manual → melhora thresholds e pesos.

---

1) Banco — novas tabelas (Drizzle)

```
/shared/schema.ai.learn.ts

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

export const aiSeedJobs = pgTable("ai_seed_jobs", {
  id: uuid("id").primaryKey().defaultRandom(),
  tenantId: varchar("tenant_id", { length:64 }).notNull(),
  kind: varchar("kind", { length:16 }).notNull(), // "theme" | "link"
  payload: jsonb("payload").$type<{ theme?:string; url?:string; domainAllow?:string[] }>().notNull(),
  status: varchar("status", { length:16 }).notNull().default("queued"), // queued|running|done|error
  createdAt: timestamp("created_at").defaultNow().notNull(),
  updatedAt: timestamp("updated_at").defaultNow().notNull()
}, t => ({
  idx: index("ai_seed_jobs_idx").on(t.tenantId, t.status, t.createdAt)
}));

export const aiDiscoveries = pgTable("ai_discoveries", {
  id: uuid("id").primaryKey().defaultRandom(),
  tenantId: varchar("tenant_id", { length:64 }).notNull(),
  seedJobId: uuid("seed_job_id").notNull(),
  url: varchar("url", { length:1024 }).notNull(),
  canonicalUrl: varchar("canonical_url", { length:1024 }),
  depth: integer("depth").notNull().default(0),
  meta: jsonb("meta").$type<any>().default({}), // {title, lang, source: serp|sitemap|crawl}
  fetched: boolean("fetched").notNull().default(false),
  allowed: boolean("allowed").notNull().default(true),
  score: real("score").default(0),
```

```
  createdAt: timestamp("created_at").defaultNow().notNull()
}, t => ({
  idx: index("ai_disc_idx").on(t.tenantId, t.seedJobId)
}));

export const aiFetches = pgTable("ai_fetches", {
  id: uuid("id").primaryKey().defaultRandom(),
  tenantId: varchar("tenant_id", { length:64 }).notNull(),
  discoveryId: uuid("discovery_id").notNull(),
  url: varchar("url", { length:1024 }).notNull(),
  status: varchar("status", { length:16 }).notNull(), // ok|blocked|error
  statusCode: integer("status_code"),
  bytes: integer("bytes").default(0),
  meta: jsonb("meta").$type<any>().default({}), // {robots:true, reason, contentType,...}
  createdAt: timestamp("created_at").defaultNow().notNull()
});

export const aiCurations = pgTable("ai_curations", {
  id: uuid("id").primaryKey().defaultRandom(),
  tenantId: varchar("tenant_id", { length:64 }).notNull(),
  discoveryId: uuid("discovery_id").notNull(),
  documentId: uuid("document_id"),
  approved: boolean("approved").notNull().default(false),
  rejected: boolean("rejected").notNull().default(false),
  editorNote: text("editor_note"),
  scoreBefore: real("score_before").default(0),
  scoreAfter: real("score_after").default(0),
  createdAt: timestamp("created_at").defaultNow().notNull()
});
```

Mantém compatibilidade com `ai_documents/ai_chunks` e com as tabelas de telemetria que já criamos nas fases anteriores.

---

2) Backend — crawler + normalizador + curator + scheduler

2.1 Config (env)

`.env` (ou variáveis no Replit)

```
AION_WHITELIST=https://www.yesyoudeserve.tours,https://www.visitportugal.com
AION_CRAWL_MAX_PAGES=60
AION_CRAWL_MAX_DEPTH=2
AION_RATE_PER_HOST=0.5   # req/seg por host
AION_JS_RENDER=false     # true usa Playwright headless (opcional)
AION_FETCH_USER_AGENT=AION-Bot/1.0 (+https://aion.local)
```

2.2 Utilitários: robots/sitemap/normalização

`/server/learn/http.ts`

```
import fetch from "node-fetch";
import { parse as parseRobots } from "robots-txt-parse";
import { JSDOM } from "jsdom";
import { Readability } from "@mozilla/readability";
import * as url from "url";

export async function canFetch(u:string, ua:string){
  try{
    const rUrl = new URL("/robots.txt", new URL(u).origin).toString();
    const txt = await (await fetch(rUrl)).text();
    const robots = parseRobots(txt);
    return robots.isAllowed(u, ua);
  }catch{ return true; }
}

export function canonicalize(u:string){
  const p = new URL(u);
  p.hash = "";
```

```
  p.searchParams.sort();
  return p.toString();
}

export async function fetchAndExtract(u:string, ua:string){
  const r = await fetch(u, { headers:{ "User-Agent": ua, "Accept-Language":"pt,en;q=0.8" }});
  const contentType = r.headers.get("content-type")||"";
  const bytes = Number(r.headers.get("content-length")||0);
  const html = await r.text();
  if (!/text\/html/i.test(contentType) && !/<html/i.test(html)) {
    return { ok:false, reason:"unsupported content", bytes, contentType };
  }
  const dom = new JSDOM(html, { url: u });
  const reader = new Readability(dom.window.document);
  const article = reader.parse();
  const text = (article?.textContent || dom.window.document.body.textContent || "").replace(/\s+/g," ").trim();
  const title = article?.title || dom.window.document.title || "";
  const lang = dom.window.document.documentElement.lang || "und";
  const links = Array.from(dom.window.document.querySelectorAll("a[href]"))
    .map((a:any)=>a.getAttribute("href"))
    .filter(Boolean)
    .map((href:string)=> new URL(href, u).toString());
  return { ok:true, title, text, lang, links, bytes, contentType };
}
```

2.3 Scoring (qualidade/frescor/autoridade/tema)

/server/learn/scoring.ts

```
// s_t (texto): entropia e compressibilidade; s_f: frescor; s_a: autoridade (rank manual); s_m: match do tema
export function textEntropyScore(s:string){
  // Shannon entropy normalizada 0..1
  const freq = new Map<string,number>();
  for (const ch of s.slice(0,20000)) freq.set(ch, (freq.get(ch)||0)+1);
  const n = [...freq.values()].reduce((a,b)=>a+b,0)||1;
  let H=0; for (const c of freq.values()){ const p=c/n; H += -p*Math.log2(p); }
  const Hmax = Math.log2(Math.min(128, freq.size||1));
  return Hmax? Math.min(1, H/Hmax) : 0.3;
}
export function freshnessScore(ageDays:number, halfLife=60){
  return Math.exp(-Math.log(2)* (ageDays/halfLife));
}
export function authorityScore(rank:number){ // 1..5
  return Math.min(1, Math.max(0, (rank-1)/4 ));
}
export function topicMatchScore(text:string, theme:string){
  const rx = new RegExp(theme.split(/\s+/).filter(Boolean).join("|"), "i");
  return rx.test(text) ? 1 : 0;
}
export function overallScore({ st, sf, sa, sm }:{st:number; sf:number; sa:number; sm:number}){
  // pesos podem vir do painel depois; defaults
  const wt=0.55, wf=0.15, wa=0.15, wm=0.15;
  return wt*st + wf*sf + wa*sa + wm*sm;
}
```

2.4 Discoverer (Sitemaps + SERP) e Crawler

/server/learn/discover.ts

```
import fetch from "node-fetch";
import { parseStringPromise as parseXML } from "xml2js";

export async function sitemapUrls(origin:string, cap=100){
  try{
    const xml = await (await fetch(new URL("/sitemap.xml", origin).toString())).text();
    const j = await parseXML(xml);
    const urls: string[] = [];
```

```
    if (j.urlset?.url) for (const u of j.urlset.url) if (u.loc?.[0]) urls.push(u.loc[0]);
    return urls.slice(0, cap);
  }catch{ return []; }
}

export async function simpleSERP(theme:string, cap=20){
  // usamos DuckDuckGo JSON como fallback simples
  try{
    const r = await (await fetch(`https://api.duckduckgo.com/?q=${encodeURIComponent(
theme)}&format=json`)).json();
    const links = (r?.RelatedTopics||[]).map((x:any)=>x?.FirstURL).filter(Boolean);
    return links.slice(0, cap);
  }catch{ return []; }
}
```

2.5 Curadoria & Ingest

```
/server/learn/curator.ts

import { db } from "../db";
import { aiDocuments, aiChunks } from "@shared/schema.ai.kb";
import { embedTextMany } from "../ai/vector";
import { smartChunk } from "../ai/math";

export async function approveToKB(tenantId:string, items:{discoveryId:string; canonicalUrl:string;
title:string; text:string}[]){
  const createdIds: string[] = [];
  for (const it of items){
    const [doc] = await db.insert(aiDocuments).values({
      tenantId, source: "url", uri: it.canonicalUrl, title: it.title
    }).returning();
    const parts = smartChunk(it.text, 1200, 200);
    // embedding por lote
    const vecs = await embedTextMany(parts);
    let order=0;
    for (let i=0;i<parts.length;i++){
      await db.insert(aiChunks).values({
        tenantId, documentId: doc.id, order: order++,
        text: parts[i], vector: vecs[i], approved: true
      });
    }
    createdIds.push(doc.id);
  }
  return createdIds;
}
```

2.6 Scheduler (fila, rate limit, retries, backoff)

```
/server/learn/scheduler.ts

import Bottleneck from "bottleneck";
import { db } from "../db";
import { aiSeedJobs, aiDiscoveries, aiFetches } from "@shared/schema.ai.learn";
import { canFetch, canonicalize, fetchAndExtract } from "./http";
import { sitemapUrls, simpleSERP } from "./discover";
import { textEntropyScore, freshnessScore, authorityScore, topicMatchScore, overallScore } from
"./scoring";

const UA = process.env.AION_FETCH_USER_AGENT || "AION-Bot/1.0";
const WHITELIST = (process.env.AION_WHITELIST||"").split(",").filter(Boolean);
const MAX_PAGES = Number(process.env.AION_CRAWL_MAX_PAGES||"60");
const MAX_DEPTH = Number(process.env.AION_CRAWL_MAX_DEPTH||"2");

// Rate limit por host
const limiter = new Bottleneck({ minTime: Math.round(1000/(Number(process.env.AION_RATE_PER_HOST
||"0.5")||0.5)) });

function allowed(url:string){
  return !WHITELIST.length || WHITELIST.some(w => url.startsWith(w));
```

```
}

export async function submitSeedTheme(tenantId:string, theme:string){
  const [job] = await db.insert(aiSeedJobs).values({tenantId, kind:"theme", payload:
{theme}}).returning();
  return job.id;
}
export async function submitSeedLink(tenantId:string, link:string){
  const [job] = await db.insert(aiSeedJobs).values({tenantId, kind:"link", payload:
{url:link}}).returning();
  return job.id;
}

// Worker principal: descobre → coleta → pontua → salva para curadoria
export async function runWorkerOnce(){
  const [job] = await db.execute(`SELECT * FROM ai_seed_jobs WHERE status='queued' ORDER BY created_at
ASC LIMIT 1`);
  if (!(job as any[]).length) return 0;
  const J = (job as any[])[0];

  await db.execute(`UPDATE ai_seed_jobs SET status='running', updated_at=NOW() WHERE id=$1`, [J.id] as
any);

  try{
    // 1) seeds
    let seeds: string[] = [];
    if (J.kind === "theme") {
      // para cada domínio permitido, sitemap + SERP
      for (const w of WHITELIST){
        seeds.push(...await sitemapUrls(new URL(w).origin, 50));
      }
      seeds.push(...await simpleSERP(J.payload.theme||"", 30));
    } else {
      seeds.push(J.payload.url);
    }
    seeds = Array.from(new Set(seeds)).filter(allowed).slice(0, MAX_PAGES);

    // 2) crawl superficial com depth limitado
    const queue: { url:string; depth:number }[] = seeds.map(u=>({url:u, depth:0}));
    const seen = new Set<string>();
    let processed=0;

    while (queue.length && processed<MAX_PAGES){
      const { url } = queue.shift()!;
      const can = allowed(url) && await canFetch(url, UA);
      const cu = canonicalize(url);
      if (!can || seen.has(cu)) continue;
      seen.add(cu);

      const fx = await limiter.schedule(()=>fetchAndExtract(cu, UA));
      await db.insert(aiFetches).values({
        tenantId:J.tenant_id, discoveryId: J.id, url:cu,
        status: fx.ok? "ok" : "error", statusCode: fx.ok? 200 : 400,
        bytes: fx.bytes||0, meta: { contentType: fx.contentType }
      });

      if (fx.ok){
        // 3) pontuação
        const st = textEntropyScore(fx.text);
        const sf = freshnessScore(0); // sem data exata → 0 dias (ou parse meta if available)
        const sa = authorityScore(3); // pode vir de tabela/manual (editor ajusta depois)
        const sm = topicMatchScore(fx.text, J.payload.theme||"");
        const score = overallScore({ st, sf, sa, sm });

        await db.insert(aiDiscoveries).values({
          tenantId:J.tenant_id, seedJobId:J.id, url, canonicalUrl: cu, depth:0,
          meta:{ title: fx.title, lang: fx.lang }, fetched: true, allowed: true, score
        });
```

```
        // 4) próxima camada (se houver)
        // (por segurança, só adiciona links do mesmo domínio e depth 1)
        const origin = new URL(cu).origin;
        const next = (fx.links||[]).filter((l:string)=>l.startsWith(origin)).slice(0,10);
        for (const n of next) if (!seen.has(canonicalize(n)))
          queue.push({ url:n, depth:1 });
      }
      processed++;
    }

    await db.execute(`UPDATE ai_seed_jobs SET status='done', updated_at=NOW() WHERE id=$1`, [J.id] as
any);
    return processed;
  } catch (e){
    await db.execute(`UPDATE ai_seed_jobs SET status='error', updated_at=NOW() WHERE id=$1`, [J.id] as
any);
    return -1;
  }
}
```

O **worker** pode rodar via **cron** (abaixo) ou endpoint manual. Ele respeita robots, whitelists, faz sitemap+SERP, normaliza conteúdo com Readability, pontua, salva descobertas **para curadoria**.

2.7 Rotas Admin de Aprendizado/Curadoria

```
/server/ai/routes.learning.ts

import type { Express, Request } from "express";
import { db } from "../db";
import { aiSeedJobs, aiDiscoveries, aiCurations } from "@shared/schema.ai.learn";
import { submitSeedTheme, submitSeedLink, runWorkerOnce } from "../learn/scheduler";
import { approveToKB } from "../learn/curator";

const T = (req:Request)=> (req as any).tenantId || process.env.PRIMARY_TENANT_ID!;

export function registerLearningRoutes(app:Express){
  // Enfileirar tema
  app.post("/api/ai/learn/seed.theme", async (req,res)=>{
    const id = await submitSeedTheme(T(req), req.body.theme);
    res.json({ ok:true, id });
  });
  // Enfileirar link
  app.post("/api/ai/learn/seed.link", async (req,res)=>{
    const id = await submitSeedLink(T(req), req.body.url);
    res.json({ ok:true, id });
  });
  // Executar 1 batelada do worker (ou agendar via cron)
  app.post("/api/ai/learn/worker.run", async (_req,res)=>{
    const n = await runWorkerOnce();
    res.json({ ok:true, processed:n });
  });

  // Listar descobertas para curadoria
  app.get("/api/ai/learn/discoveries", async (req,res)=>{
    const rows = await db.execute(`
      SELECT id, url, canonical_url, meta, score FROM ai_discoveries
      WHERE tenant_id=$1 AND fetched=true ORDER BY score DESC LIMIT 250
    `, [T(req)] as any);
    res.json(rows);
  });

  // Aprovação em lote → KB
  app.post("/api/ai/learn/curate.approve", async (req,res)=>{
    const tenantId = T(req);
    const items = (req.body?.items||[]) as { id:string }[];
    if (!items.length) return res.json({ ok:false, msg:"empty" });
    const rows = await db.execute(`SELECT id, canonical_url, meta FROM ai_discoveries WHERE id =
ANY($1)`, [items.map(i=>i.id)] as any);
    const payload = (rows as any[]).map(r=>({
```

```
      discoveryId: r.id,
      canonicalUrl: r.canonical_url,
      title: r.meta?.title || r.canonical_url,
      text: r.meta?.text || "" // veremos a seguir: vamos guardar texto no meta do discovery
    }));
    const created = await approveToKB(tenantId, payload);
    for (const r of rows as any[]){
      await db.insert(aiCurations).values({ tenantId, discoveryId: r.id, approved:true, scoreBefore:
0, scoreAfter: 1 });
    }
    res.json({ ok:true, created });
  });
}
```

> **Nota**: Para manter o texto acessível à curadoria, armazene `text` no `meta` do `ai_discoveries` durante `fetchAndExtract` (ajuste simples: quando salvar discovery, inclua `meta:{..., text: fx.text }`).

2.8 Agendamento (cron)

`/server/learn/cron.ts`

```
import cron from "node-cron";
import { runWorkerOnce } from "./scheduler";

// a cada 5 minutos processa uma batelada (ajuste conforme uso)
export function startLearnCron(){
  cron.schedule("*/5 * * * *", async ()=>{
    try{ await runWorkerOnce(); }catch(e){ /* log */ }
  });
}
```

No bootstrap do servidor:

```
import { registerLearningRoutes } from "./ai/routes.learning";
import { startLearnCron } from "./learn/cron";

registerLearningRoutes(app);
startLearnCron(); // opcional, se quiser automático
```

---

3) Frontend — UI final de Aprendizado/Curadoria

`/ui/pages/admin/learning.tsx` (substitui versão simples; **completa**)

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

type Disc = { id:string; url:string; canonical_url:string; meta:any; score:number };

export default function LearningPage(){
  const [theme,setTheme] = useState("");
  const [link,setLink] = useState("");
  const [discoveries,setDiscoveries] = useState<Disc[]>([]);
  const [selected,setSelected] = useState<Record<string,boolean>>({});
  const [running,setRunning] = useState(false);

  async function enqueueTheme(){
    await fetch("/api/ai/learn/seed.theme",{ method:"POST", headers:{ "Content-Type":"application/
json" }, body: JSON.stringify({ theme }) });
    setTheme("");
  }
  async function enqueueLink(){
    await fetch("/api/ai/learn/seed.link",{ method:"POST", headers:{ "Content-Type":"application/json"
}, body: JSON.stringify({ url: link }) });
    setLink("");
  }
  async function runWorker(){
    setRunning(true);
    await fetch("/api/ai/learn/worker.run", { method:"POST" });
    await loadDiscoveries();
```

```
      setRunning(false);
    }
    async function loadDiscoveries(){
      const r = await fetch("/api/ai/learn/discoveries"); const j = await r.json();
      setDiscoveries(j as Disc[]);
    }
    useEffect(()=>{ loadDiscoveries(); },[]);

    function toggle(id:string){ setSelected(s=>({ ...s, [id]: !s[id] })); }

    async function approveSelected(){
      const items = Object.keys(selected).filter(k=>selected[k]).map(id=>({ id }));
      if (!items.length) return;
      await fetch("/api/ai/learn/curate.approve", { method:"POST", headers:{ "Content-
Type":"application/json" }, body: JSON.stringify({ items }) });
      setSelected({}); await loadDiscoveries();
      alert("Aprovado e enviado para a Knowledge Base.");
    }

    return (
      <div className="space-y-6">
        <h1 className="text-2xl font-bold">Aprendizado Autônomo & Curadoria Web</h1>

        <div className="grid grid-cols-1 md:grid-cols-3 gap-4">
          <div className="border rounded p-3 space-y-2">
            <div className="font-semibold">Por Tema</div>
            <input className="border p-2 w-full" placeholder="Ex.: turismo sustentável em Sintra"
              value={theme} onChange={e=>setTheme(e.target.value)} />
            <button className="px-3 py-2 bg-emerald-600 text-white rounded" onClick={enqueueTheme}>
Enfileirar</button>
          </div>

          <div className="border rounded p-3 space-y-2">
            <div className="font-semibold">Por Link</div>
            <input className="border p-2 w-full" placeholder="https://exemplo.com/artigo"
              value={link} onChange={e=>setLink(e.target.value)} />
            <button className="px-3 py-2 bg-emerald-600 text-white rounded" onClick={enqueueLink}>
Enfileirar</button>
          </div>

          <div className="border rounded p-3 space-y-2">
            <div className="font-semibold">Execução</div>
            <button className="px-3 py-2 bg-indigo-600 text-white rounded" onClick={runWorker} disabled=
{running}>
              {running? "Processando..." : "Processar Batelada"}
            </button>
            <div className="text-xs text-gray-400">Dica: habilite o agendador para processar
automaticamente a cada 5 min.</div>
          </div>
        </div>

        <div className="border rounded p-3">
          <div className="flex items-center justify-between mb-2">
            <div className="font-semibold">Pré-visualização / Curadoria</div>
            <button className="px-3 py-2 bg-blue-600 text-white rounded" onClick={approveSelected}>
Aprovar Selecionados → KB</button>
          </div>
          <div className="max-h-[55vh] overflow-auto">
            <table className="w-full text-sm">
              <thead className="text-gray-400">
                <tr><th></th><th>Score</th><th>Título</th><th>URL</th><th>Resumo</th></tr>
              </thead>
              <tbody>
                {discoveries.map(d=>{
                  const title = d.meta?.title || "(sem título)";
                  const excerpt = (d.meta?.text || "").slice(0,240);
                  return (
                    <tr key={d.id} className="border-t border-gray-800">
                      <td className="align-top p-2"><input type="checkbox" checked={!!selected[d.id]}
```

```
onChange={()=>toggle(d.id)} /></td>
                    <td className="align-top p-2">{d.score.toFixed(2)}</td>
                    <td className="align-top p-2">{title}</td>
                    <td className="align-top p-2"><a className="text-blue-400 underline" href=
{d.canonical_url} target="_blank" rel="noreferrer">{new URL(d.canonical_url).hostname}</a></td>
                    <td className="align-top p-2"><pre className="whitespace-pre-wrap text-xs text-
gray-400">{excerpt}</pre></td>
                  </tr>
                );
              })}
            </tbody>
          </table>
        </div>
      </div>

    </div>
  );
}
```

Essa UI permite **enfileirar temas/links**, **rodar batelada**, **pré-visualizar e aprovar** para entrar na KB com **chunking + embeddings + ANN incremental** (via rotas anteriores).

---

4) Matemática (decisões e aprendizado ativo)

- **Score de qualidade** S (para ordenar pré-visualização):

$$S = w_t s_t + w_f s_f + w_a s_a + w_m s_m,$$

onde s_t é entropia/compressibilidade (conteúdo não boilerplate), s_f é frescor (meia-vida h), s_a autoridade (rank 1..5), e s_m match de tema. Pesos ajustáveis no painel.

- **Aprendizado ativo**: a cada sprint, pegue os k itens com **confiança** C mais próxima de τ ($|C-\tau|$ mínimo) → curadoria humana → **atualiza pesos** (w_t, w_f, w_a, w_m) e thresholds (via regressão logística simples, se quiser expandir depois).

📊 FASE 5 — Telemetria & Métricas (nDCG/MRR/CTR/CR + custos + fallbacks + heatmaps)

1) Banco — eventos e séries

`/shared/schema.ai.metrics.ts` (superset do que enviei antes)

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

export const aiEvents = pgTable("ai_events", {
  id: uuid("id").primaryKey().defaultRandom(),
  tenantId: varchar("tenant_id", { length:64 }).notNull(),
  kind: varchar("kind", { length:32 }).notNull(), // answer|retrieval|fallback|
click|purchase|budget|error
  value: real("value").default(0),                // ex.: custo USD em fallback, ou score
  meta: jsonb("meta").$type<any>().default({}),   // {q, rank, rel, tokens, model, reason}
  createdAt: timestamp("created_at").defaultNow().notNull()
}, t => ({
  idx: index("ai_events_idx").on(t.tenantId, t.kind, t.createdAt)
}));

export const aiDaily = pgTable("ai_daily", {
  id: uuid("id").primaryKey().defaultRandom(),
  tenantId: varchar("tenant_id", { length:64 }).notNull(),
  day: varchar("day", { length:10 }).notNull(), // YYYY-MM-DD
  ndcg: real("ndcg").default(0),
  mrr: real("mrr").default(0),
  ctr: real("ctr").default(0),
  cr: real("cr").default(0),
```

```
  fallbackRate: real("fallback_rate").default(0),
  costUSD: real("cost_usd").default(0),
  createdAt: timestamp("created_at").defaultNow().notNull()
});
```

> **aiEvents** é granular (eventos). **aiDaily** são agregados (pré-computados por job).
> Você já tinha **ai_fallback_logs** e métricas básicas; aqui consolidamos num **modelo único de eventos + agregados**.

2) Agregador diário

/server/metrics/aggregate.ts

```
import { db } from "../db";
import { aiEvents, aiDaily } from "@shared/schema.ai.metrics";

export async function recomputeDaily(tenantId:string, dayISO:string){
  // Carrega eventos do dia
  const rows = await db.execute(`
    SELECT * FROM ai_events
    WHERE tenant_id=$1 AND created_at::date = $2::date
  `, [tenantId, dayISO] as any) as any[];

  // Coleta relevâncias e ranks p/ nDCG/MRR
  const rels: number[] = [];
  const ranks: number[] = [];
  let clicks=0, impressions=0, purchases=0, fallbacks=0, cost=0;

  for (const e of rows){
    if (e.kind==="retrieval" && e.meta?.rel!=null) rels.push(Number(e.meta.rel));
    if (e.kind==="answer" && e.meta?.rank!=null) ranks.push(Number(e.meta.rank));
    if (e.kind==="click") clicks++;
    if (e.kind==="answer") impressions++;
    if (e.kind==="purchase") purchases++;
    if (e.kind==="fallback") { fallbacks++; cost += Number(e.value||0); }
  }

  const ndcg = computeNDCGMean(rels, 10);
  const mrr  = computeMRR(ranks);
  const ctr  = impressions? clicks/impressions : 0;
  const cr   = impressions? purchases/impressions : 0;
  const fr   = impressions? fallbacks/impressions : 0;

  await db.execute(`DELETE FROM ai_daily WHERE tenant_id=$1 AND day=$2`, [tenantId, dayISO] as any);
  await db.insert(aiDaily).values({ tenantId, day: dayISO, ndcg, mrr, ctr, cr, fallbackRate: fr,
costUSD: cost });
}

function computeNDCGMean(rels:number[], k:number){
  if (!rels.length) return 0;
  // aqui, para simplicidade: média dos rels como se já fossem top-k (você pode guardar dcg/idcg por
query no aiEvents.meta)
  const srt = [...rels].sort((a,b)=>b-a).slice(0,k);
  const idcg = srt.reduce((s,r,i)=> s + (Math.pow(2,r)-1)/Math.log2(i+2), 0);
  const dcg  = rels.slice(0,k).reduce((s,r,i)=> s + (Math.pow(2,r)-1)/Math.log2(i+2), 0);
  return idcg? dcg/idcg : 0;
}
function computeMRR(ranks:number[]){ return ranks.length? 1 / Math.max(1, Math.min(...ranks)) : 0; }
```

Cron de agregação (diário):

```
import cron from "node-cron";
import { recomputeDaily } from "./aggregate";

export function startMetricsCron(){
  cron.schedule("15 0 * * *", async ()=>{
    const day = new Date().toISOString().slice(0,10);
    const tenant = process.env.PRIMARY_TENANT_ID!;
    try{ await recomputeDaily(tenant, day); }catch(e){ /* log */ }
```

```
  });
}
```

No bootstrap:

```
import { startMetricsCron } from "./metrics/aggregate.cron";
startMetricsCron();
```

3) Rotas para Telemetria

```
/server/ai/routes.telemetry.ts

import type { Express, Request } from "express";
import { db } from "../db";
import { aiDaily, aiEvents } from "@shared/schema.ai.metrics";
import { eq } from "drizzle-orm";

const T=(req:Request)=> (req as any).tenantId || process.env.PRIMARY_TENANT_ID!;

export function registerTelemetryRoutes(app:Express){
  app.get("/api/ai/metrics/daily", async (req,res)=>{
    const rows = await db.select().from(aiDaily).where(eq(aiDaily.tenantId,T(req)));
    res.json(rows);
  });
  app.get("/api/ai/metrics/events", async (req,res)=>{
    const rows = await db.select().from(aiEvents).where(eq(aiEvents.tenantId,T(req)));
    res.json(rows.slice(-1000)); // últimos 1000 eventos
  });
}
```

> Integre com os **logs de fallback** já criados (fase anterior): cada fallback também insere um evento `kind="fallback"` com `value=costUSD`.

4) UI — Telemetria avançada

`/ui/pages/admin/ai-telemetry.tsx` (substitui a versão simples; inclui heatmap de entidades e custos)

```
import { useEffect, useMemo, useState } from "react";
import Plot from "react-plotly.js";

export default function AiTelemetryPage(){
  const [daily,setDaily]=useState<any[]>([]);
  const [events,setEvents]=useState<any[]>([]);

  async function load(){
    const d = await (await fetch("/api/ai/metrics/daily")).json();
    const e = await (await fetch("/api/ai/metrics/events")).json();
    setDaily(d); setEvents(e);
  }
  useEffect(()=>{ load(); },[]);

  const x = daily.map((r:any)=>r.day);
  const s = (k:string)=> daily.map((r:any)=>Number(r[k]||0));

  // custos por fallback (últimos N)
  const fallbackCosts = events.filter((e:any)=>e.kind==="fallback").map((e:any)=>({ x: e.createdAt, y:
e.value||0 }));

  // heatmap simples de entidades (por coocorrência de termos em meta.text dos events 'retrieval')
  const terms = new Map<string,number>();
  for (const ev of events){
    if (ev.kind==="retrieval" && ev.meta?.entities) {
      for (const t of ev.meta.entities as string[]) terms.set(t, (terms.get(t)||0)+1);
    }
  }
  const top = [...terms.entries()].sort((a,b)=>b[1]-a[1]).slice(0,20).map(([k])=>k);
  const matrix = Array(top.length).fill(0).map(()=>Array(top.length).fill(0));
  for (const ev of events){
    if (ev.kind==="retrieval" && ev.meta?.entities){
```

```
      const arr = (ev.meta.entities as string[]).filter(t=>top.includes(t));
      for (let i=0;i<arr.length;i++) for (let j=i+1;j<arr.length;j++){
        const a=top.indexOf(arr[i]), b=top.indexOf(arr[j]);
        if (a>=0 && b>=0){ matrix[a][b]+=1; matrix[b][a]+=1; }
      }
    }
  }

  return (
    <div className="space-y-6">
      <h1 className="text-2xl font-bold">Telemetria & Métricas (AION)</h1>

      <div className="grid grid-cols-1 lg:grid-cols-2 gap-6">
        <Card k="nDCG" x={x} y={s("ndcg")} />
        <Card k="MRR"  x={x} y={s("mrr")}  />
        <Card k="CTR"  x={x} y={s("ctr")}  />
        <Card k="CR"   x={x} y={s("cr")}   />
        <Card k="Fallback Rate" x={x} y={s("fallbackRate")} />
        <Card k="Cost USD" x={x} y={s("costUSD")} />
      </div>

      <div className="grid grid-cols-1 lg:grid-cols-2 gap-6">
        <div className="bg-gray-800 p-3 rounded">
          <h2 className="text-sm font-semibold mb-2">Custos por Fallback (eventos recentes)</h2>
          <Plot data={[{ x: fallbackCosts.map(r=>r.x), y: fallbackCosts.map(r=>r.y), type:"bar" }]}
                layout={layout()} config={{displayModeBar:false}} style={{width:"100%"}}/>
        </div>
        <div className="bg-gray-800 p-3 rounded">
          <h2 className="text-sm font-semibold mb-2">Heatmap de Entidades (coocorrências)</h2>
          <Plot data={[{ z: matrix, x: top, y: top, type:"heatmap", colorscale:"Viridis" }]}
                layout={{...layout(), height:420, margin:{l:80,r:20,t:10,b:80}}}
                config={{displayModeBar:false}} style={{width:"100%"}}/>
        </div>
      </div>
    </div>
  );
}

function Card({k,x,y}:{k:string;x:any[];y:any[]}){
  return (
    <div className="bg-gray-800 p-3 rounded">
      <h2 className="text-sm font-semibold mb-2">{k}</h2>
      <Plot data={[{x,y,type:"scatter",mode:"lines+markers"}]}
            layout={layout()} config={{displayModeBar:false}} style={{width:"100%"}}/>
    </div>
  );
}
function layout(){ return {
  paper_bgcolor:"rgba(0,0,0,0)", plot_bgcolor:"rgba(0,0,0,0)", font:{color:"#ddd"},
  margin:{t:20,l:40,r:10,b:30}, height:260
};}
```

Entidades: você pode preencher `events.meta.entities` com seu extrator preferido (wink-nlp, compromise, regra), ou usar os **gráficos de entidades** que já sugeri nos apêndices anteriores.

---

5) Conexão com Fallback/Budget

Em qualquer chamada externa (ex.: fallback para ChatGPT), **registre evento**:

```
// quando houver fallback:
await db.insert(aiEvents).values({
  tenantId, kind:"fallback", value: estUsd, meta:{ reason:"low_confidence", model:"gpt-4o-mini",
tokens: estTokens }
});

// quando entregar uma resposta local:
await db.insert(aiEvents).values({
  tenantId, kind:"answer", value: 1, meta:{ rank: bestRank }
```

```
});

// quando registrar clique/purchase:
await db.insert(aiEvents).values({ tenantId, kind:"click", value:1, meta:{} });
await db.insert(aiEvents).values({ tenantId, kind:"purchase", value:1, meta:{ amount: orderValue } });
```

---

6) Matemática (nDCG, MRR, CTR, CR, fallback/custo, score global)

- $nDCG_k = \frac{DCG_k}{IDCG_k}$ $1 \sum_{i=1}^{k} \frac{2^{rel_i}-1}{\log_2(i+1)}$
- $MRR = \frac{1}{|Q|} \sum_{q} \frac{1}{rank_q}$
- CTR=impres.cliques, CR=impres.vendas
- **Custo**: soma dos `aiEvents(kind="fallback").value` por dia.
- **Score Global (painel)**:

$$S = \alpha nDCG + \beta MRR + \gamma(1-FR) + \delta CTR + \epsilon CR - \zeta CostNorm$$

com pesos $(\alpha,\beta,\gamma,\delta,\epsilon,\zeta)$ configuráveis em **Configurações**.

---

7) Dependências

```
pnpm add node-fetch @mozilla/readability jsdom xml2js robots-txt-parse bottleneck node-cron
```

(Plotly, zustand etc. já foram adicionados nas fases anteriores.)

---

8) Checklist de integração

1. **Criar** os arquivos/tabelas acima;
2. **Registrar** rotas:

```
registerLearningRoutes(app);
registerTelemetryRoutes(app);
startLearnCron(); // se quiser automático
```

3. **Adicionar** a página `/admin/learning` no Sidebar;
4. **Habilitar** whitelists no `.env`;
5. **Garantir** que `approveToKB` chama `rebuildANN` periodicamente (ou manualmente no painel KB).

✅ Resultado

- **Fase 4** agora cobre **descoberta séria (sitemap+SERP)**, **robots/whitelist**, **rate-limit**, **normalização**, **pontuação multi-critério**, **pré-visualização**, **curadoria** e **ingest automático** com chunking/embedding/ANN.
- **Fase 5** entrega **telemetria de verdade**: eventos granulares, agregação diária, **gráficos** nDCG/MRR/CTR/CR, **custos e fallbacks**, **heatmap de entidades**, prontos pro **painel único**.