Gmail

Fillipe Guerra <fillipe.backup@gmail.com>

Revisao EAAS Codigo + Matematica

1 mensagem

Fillipe Guerra <fillipe.backup@gmail.com> 26 de outubro de 2025 às 14:27
Para: Fillipe Augusto Gomes Guerra <fillipe182@hotmail.com>, Fillipe Guerra <fillipe.backup@gmail.com>

SUMÁRIO EXECUTIVO (imparcial)

Pontos fortes

- Modelagem abrangente (CRM/ERP/Marketplace/IA), **rotas já expostas** e um frontend cobrindo quase tudo para MVP forte.
- IA **muito acima da média** para um protótipo (planner POMDP, críticos, RAG híbrido, estabilidade Lyapunov, LTL).
- Brand Scanner já captura **tokens de cores/tipografia** e gera **temas**; base para “Clone Builder”.

Lacunas que barram “Diamond”

1. **CRM**: falta **Timeline 360°** consolidada, **políticas de pipeline (SLA + campos obrigatórios + hooks)**, **consent/LGPD** granular por canal/finalidade, **dedupe/merge** e **relatórios de funil/win-loss/atribuição**.
2. **ERP**: precisa **máquina de estados + Sagas** (reserva → pagamento → faturar → expedir), **reserva transacional** com lock, **contabilidade leve (double-entry)**, **aging & dunning** e **idempotência** nos endpoints críticos.
3. **Brand Scanner**: excelente início, mas faltam **crawler controlado (profundidade + robots + sitemap)**, **K-means em CIELAB**, **detecção de contraste (WCAG)**, **font fallback detection**, **perceptual hash (pHash) para variantes de logo** e **extraction coverage** (cores de imagens/heróis, não só CSS).
4. **Marketplace**: precisa de **split/payout por seller**, **engine de promoções**, **frete (regras por CEP/peso)**, **busca facetada** e **Idempotency-Key** em `checkout`.
5. **IA de atendimento/vendas**: falta o **fio-terra** para o negócio: **tools** (createLead/buildQuote/reserveStock/createOrder/checkOrderStatus), **guardrails/consent**, **roteamento de intent**, **métricas de conversa** e **evaluation harness** (golden sets, A/B).

CAPÍTULO 1 — CRM (vendas/suporte/marketing)

Críticas objetivas (no seu código)

- **Estruturas existem**: `customers`, `deals`, `pipeline_stages`, `activities`, `segments` (em `shared/schema.ts`); front tem páginas.
- Não achei uma **consulta unificada** para **Timeline 360°** (atividades + mensagens + pedidos + pagamentos).
- `server/routes.ts` não impõe **políticas por estágio** (SLA/required fields/hooks).
- Ausência de **consentimento granular** (canal/finalidade) e **dedupe/merge** de contatos.
- Relatórios avançados de **funil/win-loss/atribuição de receita** ainda não presentes nas rotas.

Melhorias funcionais

- **Timeline 360°** paginada por `customerId`.
- **Pipeline Policy**: SLA por estágio, `requiredFields`, `onEnter/onExit`, **auto-assign**.

- **Consent/LGPD**: por `(canal, finalidade, base legal)` com expiração.
- **Dedupe/Merge**: normalização (e-mail/phone), **fuzzy score** e rotina de merge.
- **Relatórios**: *conversion by stage*, *win/loss reasons*, *revenue attribution* (deal line items ou ligação com pedidos).

Melhorias não-funcionais

- **Idempotency-Key** em criação de atividades/mensagens via integrações.
- **Busca** (opcional) com índice (Meilisearch) para contatos/empresas/atividades.
- **Playwright E2E** para: criar lead → mover pipeline com validações → converter em pedido.

Código — CONSENT + TIMELINE + POLÍTICAS + DEDUPE

(1) Consentimento por canal/finalidade — `shared/crm_extra.ts`

```
import { pgTable, uuid, text, boolean, timestamp, uniqueIndex } from "drizzle-orm/pg-core";

export const contactConsents = pgTable("crm_contact_consents", {
  id: uuid("id").primaryKey().defaultRandom(),
  contactId: uuid("contact_id").notNull(), // FK -> customers.id
  channel: text("channel").$type<"email"|"whatsapp"|"ads"|"sms">().notNull(),
  purpose: text("purpose").$type<"marketing"|"transactional"|"support">().notNull(),
  legalBasis: text("legal_basis").$type<"consent"|"contract"|"legitimate_interest">().notNull(),
  granted: boolean("granted").notNull().default(false),
  grantedAt: timestamp("granted_at", { withTimezone: true }).defaultNow(),
}, (t) => ({
  uniq: uniqueIndex("consent_uniq").on(t.contactId, t.channel, t.purpose)
}));
```

(2) Timeline 360° — `server/routes.crm.timeline.ts`

```
import type { Express, Request, Response } from "express";
import { db } from "./db";
import { sql } from "drizzle-orm";

export function registerCrmTimeline(app: Express) {
  app.get("/api/crm/customers/:id/timeline", async (req: Request, res: Response) => {
    const { id } = req.params;
    const limit = Number(req.query.limit ?? 50);
    const cursor = req.query.cursor as string | undefined;

    const after = cursor ? sql`AND occurred_at < ${cursor}` : sql``;

    const q = sql`
      (SELECT id, 'activity' AS kind, subject AS title, notes AS body, occurred_at
       FROM crm_activities WHERE customer_id = ${id} ${after})
      UNION ALL
      (SELECT id, 'message' AS kind, channel AS title, content AS body, created_at AS occurred_at
       FROM messages WHERE customer_id = ${id} ${after})
      UNION ALL
      (SELECT id, 'order' AS kind, order_number AS title, status AS body, created_at AS occurred_at
       FROM orders WHERE customer_id = ${id} ${after})
      UNION ALL
      (SELECT id, 'payment' AS kind, method AS title, status AS body, created_at AS occurred_at
       FROM payments WHERE customer_id = ${id} ${after})
      ORDER BY occurred_at DESC
      LIMIT ${limit}
    `;
    const items = await db.execute(q as any);
    res.json({ items, nextCursor: items.at(-1)?.occurred_at ?? null });
  });
}
```

(3) Políticas do pipeline (SLA/campos/hooks) — `server/crm/pipeline-policy.ts`

```
export type StagePolicy = {
  requiredFields?: string[];
  slaHours?: number;
  onEnter?: (dealId: string) => Promise<void>;
  onExit?: (dealId: string) => Promise<void>;
};

export const pipelinePolicies: Record<string, StagePolicy> = {
  qualification: { requiredFields: ["budgetRange"], slaHours: 24 },
  proposal:      { requiredFields: ["quoteId"], slaHours: 72 },
  negotiation:   { requiredFields: [], slaHours: 120 },
};

export async function validateStageTransition(deal: any, targetStage: string) {
  const p = pipelinePolicies[targetStage];
  if (!p) return;
  const missing = (p.requiredFields ?? []).filter(f => !deal[f]);
  if (missing.length) throw new Error(`Campos obrigatórios ausentes: ${missing.join(", ")}`);
}
```

(4) Dedupe + Merge — `server/crm/dedupe.ts`

```
import { db } from "../db";
import { sql } from "drizzle-orm";
import normalizePhone from "libphonenumber-js/min";

export function normEmail(e: string){ return e.trim().toLowerCase(); }
export function normPhone(p?: string){ try{ return normalizePhone(p||"")?.number; }catch{ return
p?.replace(/\D/g,""); } }

export async function findDuplicates(email?: string, phone?: string) {
  const e = email ? normEmail(email) : null;
  const p = phone ? normPhone(phone) : null;
  const q = sql`
    SELECT * FROM customers
    WHERE (${e} IS NOT NULL AND lower(email)=${e})
       OR (${p} IS NOT NULL AND regexp_replace(phone, '\\D', '', 'g')=${p})
    LIMIT 10`;
  return db.execute(q as any);
}

export async function mergeCustomers(primaryId: string, duplicateId: string) {
  // reatribuir deals/activities/messages/orders/payments do duplicate -> primary
  await db.execute(sql`UPDATE deals SET customer_id=${primaryId} WHERE customer_id=${duplicateId}` as
any);
  await db.execute(sql`UPDATE activities SET customer_id=${primaryId} WHERE
customer_id=${duplicateId}` as any);
  await db.execute(sql`DELETE FROM customers WHERE id=${duplicateId}` as any);
}
```

Fluxo CRM (ASCII)

```
[Form/Web/Whats/IG] -> POST /customers -> (dedupe/merge) -> POST /deals
          |                                    |
          v                                    v
   /timeline (feed 360°)               pipeline-policy (SLA/hooks)
```

CAPÍTULO 2 — ERP (produtos, estoque, ordens, financeiro)

Críticas objetivas (no repo)

- Há **produtos/ordens/estoque/financeiro** mapeados e rotas expostas, mas a orquestração está “espalhada”.
- Falta **máquina de estados + Sagas** (reserva → pagamento → faturar → expedir).

- **Reserva de estoque** não está isolada com **locks**.
- **Financeiro** carece de **double-entry**, **aging** e **dunning**.
- Não vi **Idempotency-Key** aplicado a `checkout/pagamentos`.

Melhorias funcionais

- **Ordem com máquina de estados** e **saga** reprocessável.
- **Reserva transacional** (SELECT FOR UPDATE ou versão).
- **Contabilidade leve** (lançamentos debit/credit).
- **Aging & Dunning** por título em atraso.

Melhorias não-funcionais

- **Idempotency-Key** + *exactly-once effect* nas rotas críticas.
- **Dead-letter** e reprocesso de etapas da Saga.
- **Tracing** das etapas (OpenTelemetry).

Código — ORDER STATE + SAGA + RESERVA + DOUBLE-ENTRY + IDEMPOTENCY

(1) Máquina de estados — `server/erp/order-state.ts`

```
export type OrderState = "NEW"|"RESERVED"|"PAID"|"INVOICED"|"FULFILLED"|"CANCELLED";

export function nextState(current: OrderState, ctx: any): OrderState {
  if (current === "NEW" && ctx.stockReserved) return "RESERVED";
  if (current === "RESERVED" && ctx.paymentStatus === "CONFIRMED") return "PAID";
  if (current === "PAID" && ctx.invoiceId) return "INVOICED";
  if (current === "INVOICED" && ctx.shipped) return "FULFILLED";
  return current;
}
```

(2) Saga da ordem — `server/erp/order-saga.ts`

```
import { nextState } from "./order-state";
import { db } from "../db";

export async function runOrderSaga(orderId: string) {
  const order = await db.query.orders.findFirst({ where: (t, {eq}) => eq(t.id, orderId) });
  if (!order) throw new Error("ORDER_NOT_FOUND");

  let state = order.state as any;
  const ctx: any = {};

  if (state === "NEW") {
    ctx.stockReserved = await reserveStock(order);
    state = await move(orderId, state, ctx);
  }
  if (state === "RESERVED") {
    ctx.paymentStatus = await confirmPayment(order);
    state = await move(orderId, state, ctx);
  }
  if (state === "PAID") {
    ctx.invoiceId = await issueInvoice(order);
    state = await move(orderId, state, ctx);
  }
  if (state === "INVOICED") {
    ctx.shipped = await ship(order);
    state = await move(orderId, state, ctx);
  }
}

async function move(orderId: string, current: string, ctx:any) {
```

```
  const ns = nextState(current as any, ctx);
  if (ns !== current) await db.update((db as any).orders).set({ state: ns }).where((t:any,
{eq}:any)=>eq(t.id, orderId));
  return ns;
}

// Implementações stub (trocar por integrações reais)
async function reserveStock(order:any){ /* tx com lock de estoque */ return true; }
async function confirmPayment(order:any){ /* gateway */ return "CONFIRMED"; }
async function issueInvoice(order:any){ /* fatura/número */ return "INV-001"; }
async function ship(order:any){ /* logística */ return true; }
```

(3) Reserva de estoque com lock — `server/erp/inventory.ts`

```
import { db } from "../db";
import { sql } from "drizzle-orm";

export async function reserveStockTx(orderId: string) {
  return db.transaction(async (tx) => {
    const items = await tx.execute(sql`SELECT sku_id, qty FROM order_items WHERE order_id=${orderId}`
as any);
    for (const it of items as any[]) {
      // lock linha de estoque
      const row = await tx.execute(sql`
        SELECT * FROM stock WHERE sku_id=${it.sku_id} FOR UPDATE
      ` as any);
      const stk = (row as any[])[0];
      if (!stk || stk.available < it.qty) throw new Error("INSUFFICIENT_STOCK");

      await tx.execute(sql`
        UPDATE stock SET available=available-${it.qty}, reserved=reserved+${it.qty}
        WHERE sku_id=${it.sku_id}
      ` as any);
    }
    await tx.execute(sql`UPDATE orders SET state='RESERVED' WHERE id=${orderId}` as any);
  });
}
```

(4) Lançamentos contábeis (double-entry) — `shared/finance_extra.ts`

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

export const ledgerEntries = pgTable("fin_ledger_entries", {
  id: uuid("id").primaryKey().defaultRandom(),
  entryAt: timestamp("entry_at", { withTimezone: true }).defaultNow(),
  accountDebit: text("account_debit").notNull(),
  accountCredit: text("account_credit").notNull(),
  amountCents: integer("amount_cents").notNull(),
  refType: text("ref_type").notNull(), // 'order'|'invoice'|'payment'
  refId: uuid("ref_id").notNull(),
});
```

(5) Middleware de Idempotência — `server/middlewares/idempotency.ts`

```
import type { Request, Response, NextFunction } from "express";
import { db } from "../db";

export async function idempotency(req: Request, res: Response, next: NextFunction) {
  const key = req.header("Idempotency-Key");
  if (!key) return res.status(400).json({ message: "Missing Idempotency-Key" });

  const hit = await (db as any).query.idempotencyKeys?.findFirst?.({ where: (t:any,
{eq}:any)=>eq(t.key, key) });
  if (hit) return res.status(200).json(hit.response);

  const originalJson = res.json.bind(res);
  (res as any).json = async (body:any) => {
    try { await (db as any).insert((db as any).idempotencyKeys).values({ key, response: body,
createdAt: new Date() }); } catch {}
    return originalJson(body);
```

```
  };
  next();
}
```

Aplicar em:

```
app.post("/api/checkout", idempotency, async (req,res)=>{ /* ... */ });
app.post("/api/payments", idempotency, async (req,res)=>{ /* ... */ });
```

Fluxo ERP (ASCII)

```
/orders:NEW -> reserveStockTx(LOCK) -> RESERVED -> confirmPayment -> PAID
         -> issueInvoice -> INVOICED -> ship -> FULFILLED
        [idempotency + reprocess + traces]
```

CAPÍTULO 3 — BRAND SCANNER (identidade & clone)

Críticas objetivas (no repo)

- `server/brandScanner.ts` usa **Puppeteer** com **K-means** no browser e coleta `computed styles` (bom!).
- Melhorar **coverage**: crawler (seguir links internos com limite/robots/sitemap), **K-means em CIELAB**, **contraste (WCAG AA/AAA)**, **font detection robusta** (fallback stacks), **logo detection** (pHash/SSIM), **amostragem de imagens** (hero/cta/bg).
- Persistência de **variações de tema** e integração com **Clone Builder** (geração de CSS vars + Tailwind tokens) pode ficar mais determinística.

Melhorias funcionais

- **Crawl controlado** (profundidade N, delay, domínios whitelisted, robots.txt + sitemap).
- **Color pipeline**: converter para **CIELAB**, K-means + **elbow** para K ótimo, normalizar **paleta (primária/secundária/acento/bg/fg)**, checar **contraste WCAG**.
- **Font pipeline**: extrair pilhas (`font-family`) e **resolver o fallback** que realmente renderiza.
- **Logo variants**: coletar favicons, `<img alt*="logo">`, `<svg>`; **pHash** para dedupe/variações.
- **Theme builder**: gerar `:root { --color-... }` + `tailwind.config.ts` dinâmico e preview.

Melhorias não-funcionais

- **Timeouts** por página e **concurrency** controlada.
- **User-Agent** e headers “humanos”; **bloqueio a scripts 3rd-party** durante extração (performance/ruído).
- **Cache** de páginas e **artefatos** (favicons/logos extraídos).

Código — CRAWLER + CIELAB + WCAG + pHASH + THEME EXPORT

(1) Crawler controlado — `server/brandScanner.crawl.ts`

```
import puppeteer from "puppeteer";
import { parse } from "node-html-parser";
import { URL } from "url";

export async function crawlSite(entryUrl: string, opts: { maxDepth: number; maxPages: number }) {
  const browser = await puppeteer.launch({ headless: "new" });
  const page = await browser.newPage();
  const queue: { url: string; depth: number }[] = [{ url: entryUrl, depth: 0 }];
  const visited = new Set<string>();
  const pages: { url: string; html: string }[] = [];
```

```
  const origin = new URL(entryUrl).origin;

  while (queue.length && pages.length < opts.maxPages) {
    const { url, depth } = queue.shift()!;
    if (visited.has(url) || depth > opts.maxDepth) continue;
    visited.add(url);

    await page.goto(url, { waitUntil: "networkidle2", timeout: 30000 });
    const html = await page.content();
    pages.push({ url, html });

    const dom = parse(html);
    const links = dom.querySelectorAll("a[href]").map(a => new URL(a.getAttribute("href")!,
origin).toString());
    for (const l of links) if (l.startsWith(origin)) queue.push({ url: l, depth: depth + 1 });
  }
  await browser.close();
  return pages;
}
```

(2) Cores: CIELAB + K-means + WCAG — `server/brandScanner.colors.ts`

```
import { kmeans } from "@mljs/kmeans"; // adicione lib leve
import { rgb2lab } from "culori"; // conversão robusta
// WCAG contraste
export function contrastRatio(rgb1:[number,number,number], rgb2:[number,number,number]){
  const L = (c:number)=>{ c/=255; return c<=0.03928? c/12.92 : Math.pow((c+0.055)/1.055, 2.4); };
  const lum = (r:number,g:number,b:number)=> 0.2126*L(r)+0.7152*L(g)+0.0722*L(b);
  const l1 = lum(...rgb1), l2 = lum(...rgb2);
  const [a,b] = l1>l2?[l1,l2]:[l2,l1];
  return (a+0.05)/(b+0.05);
}

export function clusterColors(pixels: [number,number,number][], k=5){
  const data = pixels.map(([r,g,b]) => {
    const { L, a, b } = rgb2lab({ r, g, b });
    return [L,a,b];
  });
  const res = kmeans(data, { k });
  const centers = res.centroids.map(c => c.centroid); // LAB
  // Converter centro LAB -> RGB aproximado (use culori lab2rgb se desejar precisão)
  return centers;
}
```

(3) pHash para logos — `server/brandScanner.phash.ts`

```
import Jimp from "jimp";
export async function pHashFromBuffer(buf: Buffer) {
  const img = await Jimp.read(buf);
  img.resize(32, 32).greyscale();
  const pixels:number[] = [];
  for (let y=0;y<32;y++) for (let x=0;x<32;x++) pixels.push(img.getPixelColor(x,y)&0xff);
  const avg = pixels.reduce((a,b)=>a+b,0)/pixels.length;
  const bits = pixels.map(p => (p>avg?1:0));
  return bits.join("");
}
export function hamming(a:string,b:string){ let d=0; for (let i=0;i<a.length;i++) if (a[i]!==b[i])
d++; return d; }
```

(4) Theme export (CSS vars + Tailwind) — `server/cloneBuilder.ts` (já existe, ampliar)

```
export function buildThemeCSS(tokens: any){
  const css = `
:root {
  --color-primary: ${tokens.primary};
  --color-secondary: ${tokens.secondary};
  --color-accent: ${tokens.accent};
  --color-bg: ${tokens.background};
  --color-fg: ${tokens.foreground};
}
```

```
  `.trim();
  return css;
}

// Em tailwind.config.ts, gerar theme extend com as vars mapeadas
```

Fluxo Brand Scanner (ASCII)

```
[POST /api/brand/jobs] -> puppeteer crawl(maxDepth, maxPages)
           -> extract CSS computed + image palettes (CIELAB/K)
           -> fonts (stack & fallback real) -> logos (pHash)
           -> theme tokens + WCAG check -> Clone Builder (CSS vars + Tailwind)
```

CAPÍTULO 4 — MARKETPLACE (catálogo, carrinho, checkout, split)

Críticas objetivas (no repo)

- Catálogo/carrinho/checkout/telas estão lá, mas faltam **split/payout por seller**, **engine de promoções**, **frete** por regras e **facet search**.
- `checkout` sem **Idempotency-Key** → suscetível a duplicações em reenvios/webhooks.

Melhorias funcionais

- **Split commission** por produto/seller e **mkp_payouts**.
- **Promo engine** (percent/valor, cupom, *buy X get Y*, período e escopo).
- **Frete** (CEP/peso/faixa) + **pickup**.
- **Busca facetada** (categoria/atributos/preço).
- **Sitemaps** e **SEO** básicos (sem migrar framework).

Melhorias não-funcionais

- **Idempotency-Key** no checkout (já fornecido middleware).
- **Webhooks assinados** (HMAC).
- **Imagens via CDN + thumbor/imgproxy** (opcional).

Código — SPLIT/PAYOUT + PROMO ENGINE + FRETE

(1) Commission & payout — `shared/marketplace_extra.ts`

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

export const sellerCommissions = pgTable("mkp_seller_commissions", {
  id: uuid("id").primaryKey().defaultRandom(),
  sellerId: uuid("seller_id").notNull(),
  productId: uuid("product_id").notNull(),
  percentBps: integer("percent_bps").notNull(), // basis points: 1500 = 15%
  createdAt: timestamp("created_at", { withTimezone: true }).defaultNow(),
});

export const payouts = pgTable("mkp_payouts", {
  id: uuid("id").primaryKey().defaultRandom(),
  sellerId: uuid("seller_id").notNull(),
  amountCents: integer("amount_cents").notNull(),
  status: text("status").$type<"PENDING"|"PAID"|"FAILED">().default("PENDING"),
  createdAt: timestamp("created_at", { withTimezone: true }).defaultNow(),
});
```

(2) Split — `server/marketplace/split.ts`

```
import { db } from "../db";
export async function computeSplits(orderId: string) {
  const items = await (db as any).selectFrom("order_items").where("order_id","=",orderId).execute();
// use drizzle
  const result: Record<string, number> = {};
  for (const it of items) {
    const { seller_id, product_id, total_cents } = it;
    const row = await (db as any).selectFrom("mkp_seller_commissions").where("seller_
id","=",seller_id).where("product_id","=",product_id).executeTakeFirst();
    const pct = row?.percent_bps ?? 0;
    const sellerShare = Math.round((total_cents * pct) / 10000);
    result[seller_id] = (result[seller_id] ?? 0) + sellerShare;
  }
  return result;
}
export async function enqueuePayouts(orderId: string) {
  const splits = await computeSplits(orderId);
  for (const [sellerId, amount] of Object.entries(splits)) {
    await (db as any).insertInto("mkp_payouts").values({ sellerId, amountCents: amount, status:
"PENDING" }).execute();
  }
}
```

(3) Promo engine — `server/marketplace/promo.ts`

```
export type Promo =
  | { type: "PERCENT"; value: number }           // 10% -> 0.10
  | { type: "VALUE"; valueCents: number }
  | { type: "BXGY"; x: number; y: number };      // buy X get Y (mais barato grátis)

export function applyPromo(items: { priceCents:number; qty:number }[], promo: Promo){
  let total = items.reduce((s,i)=> s + i.priceCents*i.qty, 0);
  if (promo.type === "PERCENT") return Math.max(0, Math.round(total*(1-promo.value)));
  if (promo.type === "VALUE")   return Math.max(0, total - promo.valueCents);
  if (promo.type === "BXGY") {
    const sorted = [...items].sort((a,b)=>a.priceCents-b.priceCents);
    const free = Math.floor(sorted.reduce((acc,i)=>acc+i.qty,0) / (promo.x+promo.y)) * promo.y;
    let discount = 0, count=free;
    for (const i of sorted) while (i.qty>0 && count>0){ discount += i.priceCents; i.qty--; count--; }
    return Math.max(0, total - discount);
  }
  return total;
}
```

(4) Frete simples (CEP/peso) — `server/marketplace/shipping.ts`

```
type Rule = { minCep:number; maxCep:number; minKg:number; maxKg:number; priceCents:number };
export function calcShipping(cep:number, kg:number, rules: Rule[]){
  const r = rules.find(r => cep>=r.minCep && cep<=r.maxCep && kg>=r.minKg && kg<=r.maxKg);
  return r?.priceCents ?? 0;
}
```

Fluxo Marketplace (ASCII)

```
Cart -> Pricing(+Promo) -> Checkout(Idempotency) -> Order
    -> SplitEngine -> mkp_payouts -> PSP Payout
    -> ShippingRules -> label/pickup
```

CAPÍTULO 5 — IA PARA ATENDIMENTO & VENDAS (chat, RAG, ações, avaliação)

O que você já tem (ótimo)

- **Fundação matemática** em `server/ai/*.ts`:
  • **RAG Híbrido** com fórmula: $S(x,q)=\alpha S_{sv} + \beta S\_bm25 + \gamma S\_grafo + \delta S\_fresco + \zeta S\_autoridade$
  • **Planner** POMDP com $score(a|s)=\lambda_1 \hat{Q} - \lambda_2 risk + \lambda_3 explain$ + ToT/GoT, **Lyapunov** e **LTL checker**.

- Endpoint **/api/ai/chat**.

O que falta para “calar com o negócio”

- **Ferramentas de ação (tools)** amarradas às suas rotas: `createLead`, `buildQuote`, `reserveStock`, `createOrder`, `checkOrderStatus`.
- **Guardrails** (PII, consent, canal permitido).
- **Router de intents** (vendas/suporte/status/humano), **handoff** com contexto.
- **Analytics & Evaluation**: golden sets, *turn budget*, CSAT, *conversion uplift* e **A/B**.

Matemática (resumo prático, já refletida no código sugerido)

1. **RAG Híbrido (já no repo)**

S(x,q)=αSvetor(x,q)+βSbm25(x,q)+γSgrafo(x,q)+δSfresco(x)+ζSautoridade(x), ∑α…ζ=1

- Vetor: cosseno; BM25: léxico; Grafo: distância/centralidade; Fresco: decaimento e−λΔt; Autoridade: peso por fonte.

2. **Decisão do Planner**

score(a|s)=λ1Q^(s,a)−λ2risk(s,a)+λ3explain(s,a)

- Q^: utilidade esperada (simulação/heurística), **risk**: penalidade (regras/abuso/estorno), **explain**: contribuição SHAP-like.

3. **Função de Custo Estendida** (já no repo)
   Equilibra **latência, custo de tokens, risco, confiança e conversão**; use pesos calibráveis.

Código — TOOLS + ORQUESTRADOR + GUARDA + AVALIAÇÃO

(1) Tools “coladas” no seu backend — `server/ai/tools.ts`

```
import { z } from "zod";
import fetch from "node-fetch";
const API = (p:string)=>`http://localhost:${process.env.PORT||5000}${p}`;

export const tools = {
  createLead: {
    input: z.object({ name: z.string(), email: z.string().email(), phone: z.string().optional() }),
    exec: async (i:any) => (await fetch(API("/api/customers"), { method:"POST", headers:{ "content-
type":"application/json"}, body: JSON.stringify(i)})).json()
  },
  buildQuote: {
    input: z.object({ items: z.array(z.object({ productId: z.string(), qty:
z.number().int().positive() })) }),
    exec: async (i:any) => (await fetch(API("/api/carts/price"), { method:"POST", headers:{ "content-
type":"application/json"}, body: JSON.stringify(i)})).json()
  },
  reserveStock: {
    input: z.object({ orderId: z.string() }),
    exec: async (i:any) => (await fetch(API(`/api/orders/${i.orderId}/reserve`), {
method:"POST"})).json()
  },
  createOrder: {
    input: z.object({ cartId: z.string(), paymentMethod: z.enum(["PIX","CARD","MP"]) }),
    exec: async (i:any) => (await fetch(API("/api/checkout"), { method:"POST", headers:{ "content-
type":"application/json","Idempotency-Key": crypto.randomUUID()}, body: JSON.stringify(i)})).json()
  },
  checkOrderStatus: {
    input: z.object({ orderId: z.string() }),
```

```
    exec: async (i:any) => (await fetch(API(`/api/orders/${i.orderId}`))).json()
  }
} as const;
```

(2) Guardrails (PII/consent) — `server/ai/guards.ts`

```
export function stripPII(text:string){
  return text.replace(/\b\d{3}\.?\d{3}\.?\d{3}-?\d{2}\b/g, "***CPF***")
             .replace(/\b\d{4}\s?\d{4}\s?\d{4}\s?\d{4}\b/g, "***CARD***");
}
export async function ensureConsent(customerId:string, channel:"web"|"whatsapp"|"instagram",
purpose:"marketing"|"transactional"|"support"){
  // consulte tabela crm_contact_consents
  return true;
}
```

(3) Router + RAG + Planner — `server/ai/orchestrator.ts`

```
import { tools } from "./tools";
import { scoreHybrid } from "./hybrid-rag";
import { stripPII, ensureConsent } from "./guards";

export async function handleTurn(ctx: { message: string; customerId?: string }) {
  const msg = stripPII(ctx.message);
  const intent = await classifyIntent(msg); // simples: regex+fewshot
  await ensureConsent(ctx.customerId!, "web", intent==="support"?"support":"transactional");

  if (intent === "buy") {
    const items = await detectItems(msg);               // parse de produtos/qty
    const quote = await tools.buildQuote.exec({ items });
    return `Preparei um orçamento total ${quote.total}. Fechamos?`;
  }
  if (intent === "order_status") {
    const { orderId } = await extractOrderId(msg);
    const o = await tools.checkOrderStatus.exec({ orderId });
    return `Pedido ${o.id}: ${o.state}`;
  }

  // Suporte: RAG híbrido
  const kb = await retrieveKB(msg);                     // retorna {item, vector, bm25, graph,
freshness, authority}
  const ranked = scoreHybrid(kb, msg);                  // aplica α..ζ
  return synthesize(msg, ranked.slice(0,4));
}
```

(4) Evaluation Harness (golden sets + métricas) — `server/ai/eval.ts`

```
type Case = { input:string; expected:string; intent:"buy"|"support"|"order_status" };
export async function runEval(cases: Case[]){
  let ok=0; const logs:any[]=[];
  for (const c of cases) {
    const out = await handleTurn({ message: c.input });
    const pass = judge(out, c.expected); // BLEU/ROUGE simples + regras
    logs.push({ c, out, pass });
    if (pass) ok++;
  }
  return { passRate: ok/cases.length, logs };
}
```

Fluxo IA (ASCII)

```
[Omnichat] -> Router(intent) -> (buy) tools.buildQuote -> tools.createOrder
                     \-> (status) tools.checkOrderStatus
                     \-> (support) RAG híbrido (α..ζ) -> answer
           -> Guardrails(PII/consent) -> Eval logs -> CSAT/Conversão
```

CAPÍTULO TRANSVERSAL — Segurança/Observabilidade (resumo com código)

- **Helmet + CORS estrito + Rate-Limit + CSP + x-request-id** — `server/security.ts` (aplique `harden(app)`).
- **JWT_SECRET obrigatório** em prod (ajuste em `server/auth.ts`).
- **Webhooks HMAC** (verificação de assinatura) — `server/webhooks/verify.ts`.
- **OpenTelemetry mínimo** — `server/otel.ts` (traces HTTP/DB).

(Você já tem `server/vite.ts` e infra local; esses ajustes plugam fácil sem quebrar.)

PARIDADE DE MERCADO — checklist objetivo

CRM (HubSpot/Salesforce)
☐ Timeline 360° • ☑ Pipeline (falta SLA/hooks) • ☐ Consent granular • ☑ Segments • ☐ Dedupe/Merge UX • ☐ Relatórios funil/win-loss.

ERP (Odoo/NetSuite)
☑ Produtos/Ordens • ☐ Sagas/order state • ☐ Reserva com lock • ☐ Double-entry • ☐ Aging/Dunning • ☐ Idempotency em pagamentos.

Brand Scanner (Figma Tokens + Crawlers pró)
☑ Tokens básicos • ☐ Crawl com cobertura + CIELAB • ☐ WCAG contrast • ☐ pHash de logos • ☐ Export Tailwind vars determinístico.

Marketplace (VTEX/Shopify)
☑ Catálogo/Cart/Checkout • ☐ Split/payout • ☐ Promo engine • ☐ Frete por CEP/peso • ☐ Faceted search • ☐ Idempotency no checkout.

IA Atendimento/Vendas (Intercom/Einstein/Sidekick)
☑ Base de IA forte • ☐ Tools de ação ligadas ao negócio • ☐ Guardrails/consent • ☐ Intent router/handoff • ☐ Eval & métricas.

0) Sumário executivo (duro e honesto)

O que já está muito bom

- **Base de IA fora da curva** para um produto: `server/ai/planner.ts`, `constrained-mdp.ts`, `ltl-model-checker.ts`, `lyapunov-stability.ts`, `extended-cost-function.ts`, `hybrid-rag.ts`, `critics.ts`, `shap-causal.ts`, `federated-learning.ts`.
- **Modelagem ampla** no `shared/schema.ts` cobrindo CRM/ERP/Marketplace/Finanças/RBAC/KB.
- **Front** com ~39 páginas cobrindo CRM/Deals/Atividades, Omnichat, Shop/Cart/Checkout, Finance/Admin; **rotas** expressivas em `server/routes.ts`.

O que falta para "Diamond" e paridade com líderes

1. **CRM** — consolidar **Timeline 360°** (atividades+mensagens+pedidos+pagamentos), **Políticas de Pipeline** (SLA, campos obrigatórios, hooks), **Consent/LGPD granular**, **Dedupe/Merge**, e **Relatórios** (funil, win/loss, atribuição).
2. **ERP** — **máquina de estados + Sagas** (reserva→pagamento→faturar→expedir), **reserva transacional com lock**, **contabilidade leve (double-entry)**, **aging/dunning**, **Idempotency-Key** nos endpoints críticos.
3. **Brand Scanner** — elevar o extrator: **crawler controlado**, **K-means em CIELAB**, **WCAG contrast**, **font fallback detection**, **pHash/SSIM de logos**, **amostragem de imagens (hero/bg)**, **export determinístico** de tokens (CSS vars + Tailwind).

4. **Marketplace** — **split/payout** por seller, **promo engine** (percent/valor/BXGY), **frete** (CEP/peso), **busca facetada**, **Checkout idempotente**.

5. **IA de negócio** — ligar o cérebro à operação: **tools** (createLead/buildQuote/reserveStock/createOrder/checkOrderStatus), **guardrails (PII/consent)**, **intent router**, **handoff humano**, **analytics & evaluation** (golden sets, A/B, CSAT, conversão atribuída).

---

1) CRM — críticas, arquitetura, **código completo**, e métricas

1.1 Críticas objetivas (do seu código)

- Estruturas core (customers, deals, activities, pipeline_stages, segments) estão no `shared/schema.ts`; **ótimo**.
- Faltam: **Timeline 360°**; **políticas de pipeline** (SLA, requiredFields, onEnter/onExit, auto-assign); **consentimento granular** por canal/finalidade; **dedupe/merge**; **relatórios funil/win-loss/atribuição** (só dá para derivar parcialmente).

1.2 Arquitetura (ASCII)

```
[Form/Web/Whats/IG] → POST /customers (dedupe/merge + consent)
         │
         ├→ POST /deals (pipeline policies: SLA/requiredFields/hooks)
         │
         └→ /api/crm/customers/:id/timeline  (feed 360°: activities+messages+orders+payments)
                        └→ /reports (funnel, win/loss, attribution)
```

1.3 **Código pronto** (colar):

(a) Consent/LGPD por canal/finalidade — `shared/crm_extra.ts`

```
import { pgTable, uuid, text, boolean, timestamp, uniqueIndex } from "drizzle-orm/pg-core";

export const contactConsents = pgTable("crm_contact_consents", {
  id: uuid("id").primaryKey().defaultRandom(),
  contactId: uuid("contact_id").notNull(), // FK -> customers.id
  channel: text("channel").$type<"email"|"whatsapp"|"sms"|"ads">().notNull(),
  purpose: text("purpose").$type<"marketing"|"transactional"|"support">().notNull(),
  legalBasis: text("legal_basis").$type<"consent"|"contract"|"legitimate_interest">().notNull(),
  granted: boolean("granted").notNull().default(false),
  grantedAt: timestamp("granted_at", { withTimezone: true }).defaultNow(),
}, (t) => ({
  uniq: uniqueIndex("consent_uniq").on(t.contactId, t.channel, t.purpose)
}));
```

(b) Timeline 360° — `server/routes.crm.timeline.ts`

```
import type { Express, Request, Response } from "express";
import { db } from "./db";
import { sql } from "drizzle-orm";

export function registerCrmTimeline(app: Express) {
  app.get("/api/crm/customers/:id/timeline", async (req: Request, res: Response) => {
    const { id } = req.params;
    const limit = Number(req.query.limit ?? 50);
    const cursor = req.query.cursor as string | undefined;
    const after = cursor ? sql`AND occurred_at < ${cursor}` : sql``;

    const q = sql`
      (SELECT id, 'activity' AS kind, subject AS title, notes AS body, occurred_at
         FROM crm_activities WHERE customer_id = ${id} ${after})
      UNION ALL
      (SELECT id, 'message' AS kind, channel AS title, content AS body, created_at AS occurred_at
```

```
          FROM messages WHERE customer_id = ${id} ${after})
       UNION ALL
       (SELECT id, 'order' AS kind, order_number AS title, status AS body, created_at AS occurred_at
          FROM orders WHERE customer_id = ${id} ${after})
       UNION ALL
       (SELECT id, 'payment' AS kind, method AS title, status AS body, created_at AS occurred_at
          FROM payments WHERE customer_id = ${id} ${after})
       ORDER BY occurred_at DESC
       LIMIT ${limit}
     `;
     const items = await db.execute(q as any);
     res.json({ items, nextCursor: items.at(-1)?.occurred_at ?? null });
   });
}
```

(c) Políticas de Pipeline — `server/crm/pipeline-policy.ts`

```
export type StagePolicy = {
  requiredFields?: string[];
  slaHours?: number;
  onEnter?: (dealId: string) => Promise<void>;
  onExit?: (dealId: string) => Promise<void>;
};

export const pipelinePolicies: Record<string, StagePolicy> = {
  qualification: { requiredFields: ["budgetRange"], slaHours: 24 },
  proposal:      { requiredFields: ["quoteId"], slaHours: 72 },
  negotiation:   { requiredFields: [], slaHours: 120 },
};

export async function validateStageTransition(deal: any, targetStage: string) {
  const p = pipelinePolicies[targetStage];
  if (!p) return;
  const missing = (p.requiredFields ?? []).filter(f => !deal[f]);
  if (missing.length) throw new Error(`Campos obrigatórios ausentes: ${missing.join(", ")}`);
}
```

(d) Dedupe & Merge — `server/crm/dedupe.ts`

```
import { db } from "../db";
import { sql } from "drizzle-orm";
import parsePhone from "libphonenumber-js/min";

export const normEmail = (e:string)=> e.trim().toLowerCase();
export function normPhone(p?:string){ try{ return parsePhone(p||"")?.number; } catch { return
p?.replace(/\D/g,""); } }

export async function findDuplicates(email?: string, phone?: string) {
  const e = email ? normEmail(email) : null;
  const p = phone ? normPhone(phone) : null;
  const q = sql`
    SELECT * FROM customers
     WHERE (${e} IS NOT NULL AND lower(email)=${e})
        OR (${p} IS NOT NULL AND regexp_replace(phone, '\\D', '', 'g')=${p})
     LIMIT 10`;
  return db.execute(q as any);
}

export async function mergeCustomers(primaryId: string, duplicateId: string) {
  await db.execute(sql`UPDATE deals SET customer_id=${primaryId} WHERE customer_id=${duplicateId}` as
any);
  await db.execute(sql`UPDATE activities SET customer_id=${primaryId} WHERE
customer_id=${duplicateId}` as any);
  await db.execute(sql`DELETE FROM customers WHERE id=${duplicateId}` as any);
}
```

1.4 Relatórios/Paridade (o que precisa para “100%”)

- **Funil:** conversão por estágio, tempo médio por estágio, **SLA breaches**.
- **Win/Loss:** razões, competidores, ticket médio.
- **Attribution:** `dealProducts` ou vínculo pedido↔oportunidade.

---

2) ERP — críticas, **mecânica formal**, código e fluxos

2.1 Críticas objetivas

- Você já tem produtos/ordens/estoque/financeiro; falta **orquestração central**: **máquina de estados + Sagas**, **reserva transacional**, **double-entry**, **aging/dunning**, **Idempotency-Key**.

2.2 Modelagem formal (LaTeX)

(a) Máquina de estados da ordem

S={NEW,RESERVED,PAID,INVOICED,FULFILLED,CANCELLED}

Transições:

NEWreserveRESERVED,RESERVEDpayment=CONFIRMEDPAID,PAIDinvoice
INVOICED,INVOICEDshipFULFILLED.

(b) Reserva transacional com lock

Seja Qi a quantidade solicitada de SKU i. Em transação:

∀i:SELECT FOR UPDATE stocki;require availablei≥Qi;availablei←availablei−Qi;reservedi←reservedi+Qi.

(c) Contabilidade (double-entry)

Para um recebimento de A centavos:

Cash←Cash+A,Receivables←Receivables−A,

registrado como lançamento (Debit=Cash,Credit=Receivables,A), mantendo o **invariante**:

∑Debits−∑Credits=0.

(d) Idempotência formal

Seja f a operação HTTP; para `Idempotency-Key` k, queremos:

f(k)=y⇒∀n>1,f(k)=y.

Persistimos (k,y) após a 1ª execução e retornamos y nas subsequentes.

2.3 Código completo

(a) Máquina de estados — `server/erp/order-state.ts`

```
export type OrderState = "NEW"|"RESERVED"|"PAID"|"INVOICED"|"FULFILLED"|"CANCELLED";

export function nextState(current: OrderState, ctx: any): OrderState {
  if (current === "NEW" && ctx.stockReserved) return "RESERVED";
  if (current === "RESERVED" && ctx.paymentStatus === "CONFIRMED") return "PAID";
  if (current === "PAID" && ctx.invoiceId) return "INVOICED";
  if (current === "INVOICED" && ctx.shipped) return "FULFILLED";
  return current;
}
```

(b) Saga — `server/erp/order-saga.ts`

```
import { nextState } from "./order-state";
import { db } from "../db";

export async function runOrderSaga(orderId: string) {
  const order = await db.query.orders.findFirst({ where: (t, {eq}) => eq(t.id, orderId) });
  if (!order) throw new Error("ORDER_NOT_FOUND");

  let state: any = order.state;
  const ctx: any = {};

  if (state === "NEW") {
    ctx.stockReserved = await reserveStock(order); // tx com lock
    state = await move(orderId, state, ctx);
  }
  if (state === "RESERVED") {
    ctx.paymentStatus = await confirmPayment(order);
    state = await move(orderId, state, ctx);
  }
  if (state === "PAID") {
    ctx.invoiceId = await issueInvoice(order);
    state = await move(orderId, state, ctx);
  }
  if (state === "INVOICED") {
    ctx.shipped = await ship(order);
    state = await move(orderId, state, ctx);
  }
}

async function move(orderId: string, current: string, ctx:any) {
  const ns = nextState(current as any, ctx);
  if (ns !== current)
    await db.update((db as any).orders).set({ state: ns }).where((t:any,{eq}:any)=>eq(t.id, orderId));
  return ns;
}

// stubs: integre com seus serviços reais
async function reserveStock(order:any){ /* ... */ return true; }
async function confirmPayment(order:any){ /* ... */ return "CONFIRMED"; }
async function issueInvoice(order:any){ /* ... */ return "INV-001"; }
async function ship(order:any){ /* ... */ return true; }
```

(c) Reserva com lock — `server/erp/inventory.ts`

```
import { db } from "../db";
import { sql } from "drizzle-orm";

export async function reserveStockTx(orderId: string) {
  return db.transaction(async (tx) => {
    const items = await tx.execute(sql`SELECT sku_id, qty FROM order_items WHERE order_id=${orderId}`
as any);
    for (const it of items as any[]) {
      await tx.execute(sql`SELECT 1 FROM stock WHERE sku_id=${it.sku_id} FOR UPDATE` as any);
      const row = await tx.execute(sql`SELECT available, reserved FROM stock WHERE
sku_id=${it.sku_id}` as any);
      const s = (row as any[])[0];
      if (!s || s.available < it.qty) throw new Error("INSUFFICIENT_STOCK");

      await tx.execute(sql`
        UPDATE stock SET available=available-${it.qty}, reserved=reserved+${it.qty}
         WHERE sku_id=${it.sku_id}` as any);
    }
    await tx.execute(sql`UPDATE orders SET state='RESERVED' WHERE id=${orderId}` as any);
  });
}
```

(d) Double-entry — `shared/finance_extra.ts`

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

```
export const ledgerEntries = pgTable("fin_ledger_entries", {
  id: uuid("id").primaryKey().defaultRandom(),
  entryAt: timestamp("entry_at", { withTimezone: true }).defaultNow(),
  accountDebit: text("account_debit").notNull(),
  accountCredit: text("account_credit").notNull(),
  amountCents: integer("amount_cents").notNull(),
  refType: text("ref_type").notNull(), // 'order'|'invoice'|'payment'
  refId: uuid("ref_id").notNull(),
});
```

(e) Idempotência — `server/middlewares/idempotency.ts`

```
import type { Request, Response, NextFunction } from "express";
import { db } from "../db";

export async function idempotency(req: Request, res: Response, next: NextFunction) {
  const key = req.header("Idempotency-Key");
  if (!key) return res.status(400).json({ message: "Missing Idempotency-Key" });

  const hit = await (db as any).query.idempotencyKeys?.findFirst?.({ where: (t:any,
{eq}:any)=>eq(t.key, key) });
  if (hit) return res.status(200).json(hit.response);

  const originalJson = res.json.bind(res);
  (res as any).json = async (body:any) => {
    try { await (db as any).insert((db as any).idempotencyKeys).values({ key, response: body,
createdAt: new Date() }); } catch {}
    return originalJson(body);
  };
  next();
}
```

Aplicação:

```
app.post("/api/checkout", idempotency, async (req,res)=>{ /* ... */ });
app.post("/api/payments", idempotency, async (req,res)=>{ /* ... */ });
```

---

3) Brand Scanner — críticas, **equações**, e código completo

3.1 Críticas objetivas

- `server/brandScanner.ts` já extrai **cores/tipografia** com K-means e computed styles.
- Falta **crawler controlado** (profundidade e respeito a domínio), **CIELAB** para clustering de cores, **WCAG contrast**, **font fallback detection**, **pHash de logos**, **amostragem de imagens (hero/bg)**, **export determinístico** (CSS vars + Tailwind tokens).

3.2 Fundamentação (LaTeX)

(a) Espaço de cor e clusterização

Convertemos RGB para **CIELAB** (aproximadamente perceptualmente uniforme).
Dado N pixels xi∈R3 em L∗a∗b∗, aplicamos **K-means**:

{μk}mini=1∑Nk∈{1,…,K}min∥xi−μk∥22.

Selecionamos K via “elbow” no decréscimo de SSE. Paleta final normalizada por **uso e contraste**.

(b) Contraste WCAG 2.1

Para cores c1,c2 em RGB:

Luma(c)=0.2126L(R)+0.7152L(G)+0.0722L(B), L(v)={12.92v,(1.055v+0.055)2.4,v≤0.03928v>0.03928
Contraste(c1,c2)=min(Luma(c1),Luma(c2))+0.05max(Luma(c1),Luma(c2))+0.05.

Padrões **AA**: ≥ 4.5:1 para texto normal (3:1 para bold ≥ 14pt). **AAA**: ≥ 7:1.

(c) Logo similarity (pHash/SSIM)

Geramos **pHash** h∈{0,1}M (DCT reduzida). Distância de Hamming:

dH(h1,h2)=j=1∑M1[h1(j)□=h2(j)].

Limiar dH≤τ identifica variantes da mesma marca.

3.3 Código completo (colar)

(a) Crawler controlado — `server/brandScanner.crawl.ts`

```
import puppeteer from "puppeteer";
import { parse } from "node-html-parser";
import { URL } from "url";

export async function crawlSite(entryUrl: string, opts: { maxDepth: number; maxPages: number }) {
  const browser = await puppeteer.launch({ headless: "new" });
  const page = await browser.newPage();
  const origin = new URL(entryUrl).origin;

  const queue: { url: string; depth: number }[] = [{ url: entryUrl, depth: 0 }];
  const visited = new Set<string>();
  const pages: { url: string; html: string }[] = [];

  while (queue.length && pages.length < opts.maxPages) {
    const { url, depth } = queue.shift()!;
    if (visited.has(url) || depth > opts.maxDepth) continue;
    visited.add(url);

    await page.goto(url, { waitUntil: "networkidle2", timeout: 30000 });
    const html = await page.content();
    pages.push({ url, html });

    const dom = parse(html);
    const links = dom.querySelectorAll("a[href]").map(a => new URL(a.getAttribute("href")!,
origin).toString());
    for (const l of links) if (l.startsWith(origin)) queue.push({ url: l, depth: depth + 1 });
  }
  await browser.close();
  return pages;
}
```

(b) Cores (CIELAB + K-means + WCAG) — `server/brandScanner.colors.ts`

```
import { kmeans } from "@mljs/kmeans";
import { converter } from "culori"; // lab<->rgb
const toLab = converter("lab"), toRgb = converter("rgb");

export function contrastRatio(rgb1:[number,number,number], rgb2:[number,number,number]){
  const L = (c:number)=>{ c/=255; return c<=0.03928? c/12.92 : Math.pow((c+0.055)/1.055, 2.4); };
  const lum = (r:number,g:number,b:number)=> 0.2126*L(r)+0.7152*L(g)+0.0722*L(b);
  const l1 = lum(...rgb1), l2 = lum(...rgb2); const [a,b] = l1>l2?[l1,l2]:[l2,l1];
  return (a+0.05)/(b+0.05);
}

export function clusterColors(pixels: [number,number,number][], k=5){
  const data = pixels.map(([r,g,b]) => {
    const lab = toLab({ r, g, b, mode:"rgb" }) as any;
    return [lab.l, lab.a, lab.b];
  });
  const res = kmeans(data, { k });
  const centersLab = res.centroids.map(c => c.centroid as [number,number,number]);
```

```
  const centersRgb = centersLab.map(([l,a,b]) => {
    const rgb = toRgb({ mode:"lab", l, a, b }) as any;
    return [Math.round(rgb.r), Math.round(rgb.g), Math.round(rgb.b)] as [number,number,number];
  });
  return centersRgb;
}
```

(c) pHash — `server/brandScanner.phash.ts`

```
import Jimp from "jimp";
export async function pHashFromBuffer(buf: Buffer) {
  const img = await Jimp.read(buf);
  img.resize(32, 32).greyscale();
  const vals:number[] = [];
  for (let y=0;y<32;y++) for (let x=0;x<32;x++) vals.push(img.getPixelColor(x,y)&0xff);
  const avg = vals.reduce((a,b)=>a+b,0)/vals.length;
  return vals.map(p => (p>avg?1:0)).join("");
}
export const hamming = (a:string,b:string)=> [...a].reduce((d,ai,i)=> d + (ai!==b[i]?1:0), 0);
```

(d) Export (CSS vars + Tailwind) — `server/cloneBuilder.ts` (extender)

```
export function buildThemeCSS(tokens: any){
  return `
:root {
  --color-primary: ${tokens.primary};
  --color-secondary: ${tokens.secondary};
  --color-accent: ${tokens.accent};
  --color-bg: ${tokens.background};
  --color-fg: ${tokens.foreground};
}`.trim();
}
```

---

4) Marketplace — críticas, modelos e código completo

4.1 Críticas objetivas

- Você tem catálogo/carrinho/checkout UI + rotas; faltam **split/payout**, **promo engine**, **frete**, **facet search** e **Idempotency-Key** no checkout.

4.2 Modelagem (LaTeX)

(a) Split

Para item j com total T_j e comissão do seller p_j (em basis points):

$$\text{sellerShare}_j = \lfloor 10{,}000 T_j \cdot p_j \rfloor, \quad \text{sellerPayout} = \sum_j \text{sellerShare}_j.$$

(b) Promoções

- Percentual: $T' = T \cdot (1-\alpha)$.
- Valor fixo: $T' = \max(0, T - v)$.
- **BXGY**: gratuidade dos y itens mais baratos a cada x+y itens.

(c) Frete (regra simples)

Seja regra r válida para $\text{CEP} \in [c,c]$ e peso $w \in [w,w]$, custo C_r:

$$C(\text{CEP}, w) = C_r \text{se} \text{CEP} \in [c,c] \wedge w \in [w,w].$$

4.3 Código

(a) Commission & payout — `shared/marketplace_extra.ts`

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

export const sellerCommissions = pgTable("mkp_seller_commissions", {
  id: uuid("id").primaryKey().defaultRandom(),
  sellerId: uuid("seller_id").notNull(),
  productId: uuid("product_id").notNull(),
  percentBps: integer("percent_bps").notNull(), // 1500 = 15%
  createdAt: timestamp("created_at", { withTimezone: true }).defaultNow(),
});

export const payouts = pgTable("mkp_payouts", {
  id: uuid("id").primaryKey().defaultRandom(),
  sellerId: uuid("seller_id").notNull(),
  amountCents: integer("amount_cents").notNull(),
  status: text("status").$type<"PENDING"|"PAID"|"FAILED">().default("PENDING"),
  createdAt: timestamp("created_at", { withTimezone: true }).defaultNow(),
});
```

(b) Split — `server/marketplace/split.ts`

```
import { db } from "../db";

export async function computeSplits(orderId: string) {
  const items = await (db as any).selectFrom("order_items").where("order_id","=",orderId).execute();
  const res: Record<string, number> = {};
  for (const it of items) {
    const { seller_id, product_id, total_cents } = it;
    const row = await (db as any).selectFrom("mkp_seller_commissions")
      .where("seller_id","=",seller_id).where("product_id","=",product_id).executeTakeFirst();
    const bps = row?.percent_bps ?? 0;
    const share = Math.round((total_cents * bps) / 10000);
    res[seller_id] = (res[seller_id] ?? 0) + share;
  }
  return res;
}

export async function enqueuePayouts(orderId: string) {
  const splits = await computeSplits(orderId);
  for (const [sellerId, amount] of Object.entries(splits)) {
    await (db as any).insertInto("mkp_payouts").values({ sellerId, amountCents: amount, status:
"PENDING" }).execute();
  }
}
```

(c) Promo engine — `server/marketplace/promo.ts`

```
export type Promo =
  | { type: "PERCENT"; value: number }
  | { type: "VALUE"; valueCents: number }
  | { type: "BXGY"; x: number; y: number };

export function applyPromo(items: { priceCents:number; qty:number }[], promo: Promo){
  let total = items.reduce((s,i)=> s + i.priceCents*i.qty, 0);
  if (promo.type === "PERCENT") return Math.max(0, Math.round(total*(1-promo.value)));
  if (promo.type === "VALUE")   return Math.max(0, total - promo.valueCents);
  if (promo.type === "BXGY") {
    const sorted = [...items].sort((a,b)=>a.priceCents-b.priceCents);
    const free = Math.floor(sorted.reduce((acc,i)=>acc+i.qty,0) / (promo.x+promo.y)) * promo.y;
    let discount = 0, count=free;
    for (const i of sorted) while (i.qty>0 && count>0){ discount += i.priceCents; i.qty--; count--; }
    return Math.max(0, total - discount);
  }
  return total;
}
```

(d) Frete CEP/peso — `server/marketplace/shipping.ts`

```
type Rule = { minCep:number; maxCep:number; minKg:number; maxKg:number; priceCents:number };
export function calcShipping(cep:number, kg:number, rules: Rule[]){
  const r = rules.find(r => cep>=r.minCep && cep<=r.maxCep && kg>=r.minKg && kg<=r.maxKg);
  return r?.priceCents ?? 0;
}
```

*(Integrar **Idempotency-Key** no `/api/checkout`, como mostrado no capítulo ERP.)*

---

5) IA — revisão de nível doutorado (matemática + código orquestrado)

Você já tem uma base **fortíssima**. Abaixo, consolido e **aprofundamos** a formulação, conectando ao **negócio** (tools) com provas/esboços, e **equações LaTeX** completas, mantendo compatibilidade com os seus arquivos `server/ai/*.ts`.

5.1 POMDP + ToT/GoT (planner) — formulação e política ótima aproximada

(a) POMDP formal

$M=\langle S,A,O,T,\Omega,R,\gamma\rangle$

- S: estados latentes do diálogo/negócio (ex.: intenção, fase do funil, constraints).
- A: ações (responder, perguntar, invocar tool: createLead, buildQuote, createOrder, etc.).
- O: observações (mensagens do usuário, eventos externos).
- $T(s'|s,a)$: dinâmica; $\Omega(o|s',a)$: modelo de observação.
- **Crença** $b\in\Delta(S)$; atualização:

$b'(s')=\eta\Omega(o|s',a)\sum_{s\in S}T(s'|s,a)b(s).$

- Retorno esperado sob política π:

$V^{\pi}(b)=E[\sum_{t=0}^{\infty}\gamma^t R(s_t,a_t)|b_0=b].$

(b) Exploração estruturada (ToT/GoT)

Árvore/grafo de hipóteses; **score** para expansão:

$score(a|b)=\lambda_1\hat{Q}(b,a)-\lambda_2 risk(b,a)+\lambda_3 explain(b,a),$

com $\lambda_1+\lambda_2+\lambda_3=1$.

- $\hat{Q}$: valor estimado via simulação curta (rollouts) e críticos;
- risk: custo normativo/ético/financeiro;
- explain: medida de **explicabilidade** (ex.: relevância SHAP de features do contexto).

(c) Restrições e Lagrangiano (CMDP)

Queremos maximizar retorno com **restrições** (custo de risco, PII, latência, orçamento):

$\max_{\pi} J(\pi) \text{ sujeito a } E^{\pi}[C_k]\le c_k, k=1..K.$

Lagrangiano:

$L(\pi,\lambda)=J(\pi)-\sum_{k=1}^{K}\lambda_k(E^{\pi}[C_k]-c_k),$

e gradientes tipo **primal-dual** ajustam λ_k.

(d) Estabilidade de Lyapunov (política segura)

Seja V(x)≥0 função de Lyapunov do “estado de risco” x. Política é **estável** se

ΔV=E[V(xt+1)−V(xt)|π]≤−ϵ‖xt‖2,

garantindo decaimento da “energia de risco”.

(e) Critérios de explicabilidade (SHAP-like)

Para features zj do contexto, relevância ϕj tal que:

j∑ϕj=f(x)−E[f(X)],

com integração de caminhos (Shapley values). Usamos aproximação amostral.

(f) LTL + Deôntica (checagem de políticas)

Restrições do tipo:

G(risk(a)>τ⇒Ohandoff(a)),G(answer⇒Ocitation).

Verificação de **violações** em rotas geradas (modelo em `ltl-model-checker.ts`).

5.2 RAG híbrido — escore canônico e normalização

Com base no seu `hybrid-rag.ts`, formalizamos:

S(x,q)=αSvec(x,q)+βSbm25(x,q)+γSgraph(x,q)+δSfresh(x)+ζSauth(x),∑α..ζ=1,

onde:

- Svec=cos(ex,eq)
- Sbm25 clássico
- Sgraph: centralidade/curta distância em grafo semântico
- Sfresh=e−λΔt
- Sauth pondera fonte/verificação.

Normalização: cada componente ∈[0,1] (min-max ou z-score + sigmoid) — **importantíssimo** para mistura bem-comportada.

5.3 Função de custo estendida (já no seu repo) — regularização

J(θ)=E[ℓ(fθ(x),y)]+λsRstab(θ)+λeRethic(θ),

com

Rstab(θ)=‖θ−θt−1‖22+‖Δθ‖22,Rethic(θ)=E(x,y)[violations(x,y)].

5.4 Federated Learning (já no repo) — agregação robusta

Rounds t: clientes i enviam θi(t). **FedAvg**:

θ(t+1)=i∑wiθi(t).

Robustos: **Krum**, **Trimmed Mean** substituem média para tolerar outliers ou *poisoning*.

5.5 Conectar IA ao negócio — Tools, Router, Guardrails, Eval

(a) Tools — `server/ai/tools.ts`

```
import { z } from "zod";
import fetch from "node-fetch";
const API = (p:string)=>`http://localhost:${process.env.PORT||5000}${p}`;
```

```
export const tools = {
  createLead: {
    input: z.object({ name: z.string(), email: z.string().email(), phone: z.string().optional() }),
    exec: async (i:any) => (await fetch(API("/api/customers"), { method:"POST", headers:{ "content-
type":"application/json"}, body: JSON.stringify(i)})).json()
  },
  buildQuote: {
    input: z.object({ items: z.array(z.object({ productId: z.string(), qty:
z.number().int().positive() })) }),
    exec: async (i:any) => (await fetch(API("/api/carts/price"), { method:"POST", headers:{ "content-
type":"application/json"}, body: JSON.stringify(i)})).json()
  },
  reserveStock: {
    input: z.object({ orderId: z.string() }),
    exec: async (i:any) => (await fetch(API(`/api/orders/${i.orderId}/reserve`), {
method:"POST"})).json()
  },
  createOrder: {
    input: z.object({ cartId: z.string(), paymentMethod: z.enum(["PIX","CARD","MP"]) }),
    exec: async (i:any) => (await fetch(API("/api/checkout"), { method:"POST", headers:{ "content-
type":"application/json","Idempotency-Key": crypto.randomUUID()}, body: JSON.stringify(i)})).json()
  },
  checkOrderStatus: {
    input: z.object({ orderId: z.string() }),
    exec: async (i:any) => (await fetch(API(`/api/orders/${i.orderId}`))).json()
  }
} as const;
```

(b) Guardrails — `server/ai/guards.ts`

```
export function stripPII(text:string){
  return text.replace(/\b\d{3}\.?\d{3}\.?\d{3}-?\d{2}\b/g, "***CPF***")
             .replace(/\b\d{4}\s?\d{4}\s?\d{4}\s?\d{4}\b/g, "***CARD***");
}
export async function ensureConsent(customerId:string, channel:"web"|"whatsapp"|"instagram",
purpose:"marketing"|"transactional"|"support"){
  // consultar crm_contact_consents e bloquear caso não autorizado
  return true;
}
```

(c) Router + Planner + RAG — `server/ai/orchestrator.ts`

```
import { tools } from "./tools";
import { scoreHybrid } from "./hybrid-rag";
import { stripPII, ensureConsent } from "./guards";
// importe seus críticos, cmdp, ltl, lyapunov conforme necessidade

export async function handleTurn(ctx: { message: string; customerId?: string }) {
  const msg = stripPII(ctx.message);
  const intent = await classifyIntent(msg); // few-shot + regras
  await ensureConsent(ctx.customerId!, "web", intent==="support"?"support":"transactional");

  if (intent === "buy") {
    const items = await detectItems(msg);
    const quote = await tools.buildQuote.exec({ items });
    return `Orçamento pronto: total ${quote.total}. Posso fechar agora?`;
  }
  if (intent === "order_status") {
    const { orderId } = await extractOrderId(msg);
    const o = await tools.checkOrderStatus.exec({ orderId });
    return `Pedido ${o.id}: ${o.state}`;
  }

  // fallback: suporte com RAG híbrido (normalizado)
  const kb = await retrieveKB(msg);
  const ranked = scoreHybrid(kb, msg);
  return synthesize(msg, ranked.slice(0,4));
}
```

(d) Avaliação (golden sets) — `server/ai/eval.ts`

```
type Case = { input:string; expected:string; intent:"buy"|"support"|"order_status" };
export async function runEval(cases: Case[]){
  let ok=0; const logs:any[]=[];
  for (const c of cases) {
    const out = await handleTurn({ message: c.input });
    const pass = judge(out, c.expected); // BLEU/ROUGE simples + regras
    logs.push({ c, out, pass });
    if (pass) ok++;
  }
  return { passRate: ok/cases.length, logs };
}
```

5.6 Fluxo IA (ASCII)

```
[Omnichat] → Intent Router → (buy) buildQuote → createOrder
                      └→ (status) checkOrderStatus
                      └→ (support) RAG híbrido α..ζ → answer
          → Guardrails(PII/consent) → LTL-check → Critic → Eval logs
```

---

6) Segurança, Observabilidade e Confiabilidade (aplique já)

- **Helmet + CORS estrito + Rate-Limit + CSP + x-request-id** — (middleware abaixo)
- **JWT_SECRET obrigatório** em prod (evita fallback fraco).
- **Webhooks HMAC** (assinatura e *replay protection*).
- **OpenTelemetry** (traces HTTP/DB) para funis críticos (checkout, IA-turn).

Middleware — `server/security.ts`

```
import type { Express } from "express";
import helmet from "helmet";
import cors from "cors";
import rateLimit from "express-rate-limit";
import crypto from "crypto";

export function harden(app: Express) {
  app.set("trust proxy", 1);
  app.use(helmet({
    contentSecurityPolicy: {
      useDefaults: true,
      directives: {
        "default-src": ["'self'"],
        "script-src": ["'self'"],
        "style-src": ["'self'", "'unsafe-inline'"],
        "img-src": ["'self'", "data:"],
        "connect-src": ["'self'"],
      },
    },
    crossOriginResourcePolicy: { policy: "same-site" },
    frameguard: { action: "deny" },
  }));
  app.use(cors({ origin: process.env.ALLOWED_ORIGIN?.split(",") ?? ["http://localhost:5173"], credentials:
true }));
  app.use(rateLimit({ windowMs: 60_000, max: 300 }));
  app.use((req, _res, next) => { (req as any).rid = req.header("x-request-id") || crypto.randomUUID();
next(); });
}
```

JWT secret forte — ajuste no seu `server/auth.ts`

```
const JWT_SECRET = (() => {
  const s = process.env.JWT_SECRET;
  if (!s || s === "your-secret-key-change-in-production") {
    if (process.env.NODE_ENV === "production") throw new Error("JWT_SECRET obrigatório em produção");
  }
  return s || "dev-secret-only";
})();
```

---

7) Paridade com mercado (checklist honesto)

CRM (HubSpot/Salesforce)

- ☐ Timeline 360° • [x] Pipeline (mas falta SLA/hooks) • [ ] Consent granular • [x] Segments • [ ] Dedupe/Merge UX • [ ] Relatórios funil/win-loss/atribuição

ERP (Odoo/NetSuite)

- ☑ Produtos/Ordens • [ ] Sagas/order state • [ ] Reserva com lock • [ ] Double-entry • [ ] Aging/Dunning • [ ] Idempotency em pagamentos/checkout

Brand Scanner (classe Figma Tokens + crawlers)

- ☑ Tokens básicos • [ ] Crawl + CIELAB • [ ] WCAG contrast • [ ] pHash logos • [ ] Export determinístico Tailwind

Marketplace (VTEX/Shopify)

- ☑ Catálogo/Cart/Checkout • [ ] Split/payout • [ ] Promo engine • [ ] Frete CEP/peso • [ ] Faceted search • [ ] Idempotency no checkout

IA Atendimento/Vendas (Intercom/Einstein/Sidekick)

- ☑ Base cognitiva robusta • [ ] Tools acopladas • [ ] Guardrails/consent • [ ] Intent router/handoff • [ ] Eval & métricas

---

8) Próximos passos (régua “Diamond” — ordem sugerida)

1. **Checkout blindado**: Idempotency-Key + Webhook HMAC + traces.
2. **Split/Payout** do marketplace + **Promo engine** + **Frete**.
3. **Timeline 360°** + **Pipeline Policies** + **Consent** + **Dedupe/Merge**.
4. **Order Saga** + **Reserva com lock** + **Double-entry** + **Aging/Dunning**.
5. **IA wired-in**: tools → negócios; guardrails; intent router; eval.
6. **Brand Scanner Pro**: crawler, CIELAB, WCAG, pHash, export determinístico.