📘 MANUAL TÉCNICO DEFINITIVO — INTELIGÊNCIA ARTIFICIAL OPERACIONAL (FRIENDAPP)

🧠 CAMADA 01 — ENTRADA E PROPÓSITO MULTINÚCLEO (TÉCNICA)

🎯 Função

Definir a IA Operacional como **núcleo funcional distribuído** do FriendApp, responsável por performance, segurança, personalização e resiliência técnica.

📐 Fórmulas e Métricas Fundamentais

scoreeficienciaoperacional=(w1∗latenciamedia)+(w2∗uptime)−(w3∗erroscriticos)score_eficiencia_operacional = (w1 * latencia_media) + (w2 * uptime) - (w3 * erros_criticos)

scoreeficienciaoperacional=(w1∗latenciamedia)+(w2∗uptime)−(w3∗erroscriticos)

- **latencia_media** = tempo médio de resposta de APIs (ms)
- **uptime** = disponibilidade (%)
- **erros_criticos** = falhas fatais detectadas por módulo
- Pesos sugeridos: `w1 = 0.4`, `w2 = 0.5`, `w3 = 0.1`

🔧 Estrutura Multinúcleo

Núcleo	Função	Métrica Principal
IA Recomendação	Sugerir recursos e conexões	`precision@k` das sugestões
IA Performance	Reduzir latência e escalar clusters	`latencia_media < 250ms`
IA Segurança	Bloquear padrões anômalos	`taxa_falsos_positivos < 5%`
IA Jornada	Ajustar UX dinamicamente	`taxa_abandono_tela ↓`

📊 Modelo de Dados Inicial

```
CREATE TABLE ia_operacional_core (
    id SERIAL PRIMARY KEY,
    nucleo VARCHAR(50) NOT NULL,
    metrica VARCHAR(50),
    valor NUMERIC,
    timestamp TIMESTAMP DEFAULT NOW()
);
```

🌐 API — Estado Inicial da IA

Endpoint	Método	Função	Exemplo Response
`/api/ia/core/status`	GET	Retorna saúde da IA Operacional	`{ "uptime": 99.98, "latencia_media": 183 }`
`/api/ia/core/metrics`	GET	KPIs por núcleo	`{ "recomendacao": 0.82, "performance": 0.97 }`

📡 Logs Técnicos

- `core_health.log` → status de uptime e latência
- `core_anomaly.log` → falhas críticas e alertas de integridade

Exemplo:

```
{
  "nucleo": "performance",
  "latencia_media": 240,
  "uptime": 99.96,
  "alerta": false,
  "timestamp": "2025-09-01T12:30:21Z"
}
```

💡 Fechamento

A Camada 01 define a IA Operacional como núcleo central de sustentação funcional do FriendApp, **quantificável em métricas, logs e tabelas**, não apenas como conceito.

🧠 CAMADA 02 — ARQUITETURA DISTRIBUÍDA E ISOLAMENTO VIBRACIONAL

🎯 Objetivo

Definir a **topologia técnica da IA Operacional**: como os módulos são distribuídos em containers, como se comunicam via eventos assíncronos e como se mantém **isolados da camada vibracional (Aurah Kosmos)**.

🧬 Topologia Kubernetes

```
apiVersion: v1
kind: Pod
metadata:
  name: ia-recomendacao
spec:
  containers:
  - name: ia-recomendacao
    image: friendapp/ia-recomendacao:v1
    resources:
      limits:
        cpu: "2"
        memory: "4Gi"
---
apiVersion: v1
```

```
kind: Service
metadata:
  name: ia-recomendacao-svc
spec:
  selector:
    app: ia-recomendacao
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
```

- Cada **núcleo da IA** roda em pods isolados.
- Escalonamento automático via **Horizontal Pod Autoscaler**.
- **Node pools separados**: Recomendação, Segurança, Jornada, Performance.

📡 Event Bus (Kafka + RabbitMQ)

- Todo dado do app dispara **eventos assíncronos**.
- Cada módulo consome apenas os eventos relevantes.
- Evita bloqueio de UX com processamento síncrono.

Exemplo de Tópicos Kafka:

- `feed.new_post`
- `chat.message_sent`
- `user.session_update`
- `ia.anomaly_detected`

🔐 Camada Anti-Vibração

```
IF request.origin == aurah_kosmos:
    REJECT()
ELSE:
    ACCEPT()
```

- Nenhum módulo da IA Operacional acessa diretamente dados vibracionais crus.
- O isolamento é feito por **camada de proxy** que descarta qualquer chamada vinda da Aurah Kosmos.

📊 Fluxo de Comunicação

```
graph TD
A[Usuário interage no app] --> B[Evento gerado → Kafka]
B --> C1[IA Recomendação Pod]
B --> C2[IA Performance Pod]
B --> C3[IA Segurança Pod]
```

```
B --> C4[IA Jornada Pod]
C1 --> D1[Resposta assíncrona]
C2 --> D2[Logs de performance]
C3 --> D3[Trigger de bloqueio]
C4 --> D4[Alteração UX]
```

🗃️ Banco e Cache

Tecnologia	Função	Uso
Redis	Cache de sessões	Respostas em tempo real
PostgreSQL	Histórico de comportamento	Processamento assíncrono
ElasticSearch	Logs de eventos IA	Análise em tempo real
Firestore	Dados temporários	Feedback imediato

📐 Métricas Técnicas

- **latência_evento**: `< 200ms`
- **uptime_cluster**: `> 99.95%`
- **throughput_mensagens**: `> 50k eventos/s`
- **failover**: `< 2s`

💡 Fechamento

Esta camada define a **espinha dorsal da IA Operacional**: uma arquitetura distribuída, orientada a eventos, escalável por Kubernetes e **blindada contra contaminação vibracional**.

🧠 CAMADA 03 — IA DE RECOMENDAÇÃO FUNCIONAL (100% TÉCNICA)

1) Objetivo Técnico

Gerar **Top-K** sugestões de recursos/usuários/conteúdos por usuário em **tempo real** com pipeline **assíncrono** (Kafka) e ranking determinístico, respeitando políticas de segurança/ética e metas de qualidade (Precision@K, NDCG).

2) Modelo Matemático (Livro de Fórmulas)

2.1 Vetorização e Semelhança

- Vetor do usuário u∈Rd\mathbf{u}\in\mathbb{R}^du∈Rd, vetor do item/ação f∈Rd\mathbf{f}\in\mathbb{R}^df∈Rd.
- Similaridade cosseno:

simcos(u,f)=u·f∥u∥ ∥f∥\text{sim}_{\cos}(\mathbf{u},\mathbf{f})=\frac{\mathbf{u}\cdot\mathbf{f}}{\|\mathbf{u}\|\ \|\mathbf{f}\|}

simcos(u,f)=∥u∥ ∥f∥u·f

2.2 Fatoração de Matriz (CF)

r^u,f=μ+bu+bf+qfTpu\hat{r}_{u,f}= \mu + b_u + b_f + \mathbf{q}_f^\top \mathbf{p}_u

r^u,f=μ+bu+bf+qfTpu

onde μ\muμ é viés global, bu,bfb_u,b_fbu,bf vieses,
pu,qf∈Rk\mathbf{p}_u,\mathbf{q}_f\in\mathbb{R}^kpu,qf∈Rk.

2.3 Contexto Temporal (decay)

wt=e−λΔt(λ∈[0.01,0.2])w_t = e^{-\lambda \Delta t} \quad (\lambda \in [0.01,0.2])

wt=e−λΔt(λ∈[0.01,0.2])

2.4 Probabilidade de Clique (CTR)

P(click ∣ x)=σ(wTx),σ(z)=11+e−zP(\text{click}\,|\,\mathbf{x})=\sigma(\mathbf{w}^\top \mathbf{x}),\quad \sigma(z)=\frac{1}{1+e^{-z}}

P(click ∣ x)=σ(wTx),σ(z)=1+e−z1

x\mathbf{x}x: features (sim_\cos, r^\hat rr^, tempo, dispositivo, sessão).

2.5 Penalização de Risco/Segurança

penrisk=min(1, α·risk_scoreu,f)\text{pen}_{risk} = \min(1,\ \alpha\cdot \text{risk_score}_{u,f})

penrisk=min(1, α·risk_scoreu,f)

2.6 Exploração vs. Exploração (UCB)

ucbu,f=r¯u,f+clnNunu,f+1\text{ucb}_{u,f}=\bar{r}_{u,f} + c\sqrt{\frac{\ln N_u}{n_{u,f}+1}}

ucbu,f=r¯u,f+cnu,f+1lnNu

2.7 Re-ranking por Diversidade (MMR)

MMR(f)=λ·simcos(u,f)−(1−λ)·maxf′∈Ssimcos(f,f′)\text{MMR}(f)=\lambda\cdot\text{sim}_{\cos}(\mathbf{u},\mathbf{f})-(1-\lambda)\cdot\max_{f'\in S}\text{sim}_{\cos}(\mathbf{f},\mathbf{f'})

MMR(f)=λ·simcos(u,f)−(1−λ)·f′∈Smaxsimcos(f,f′)

SSS: conjunto já selecionado; λ∈[0.5,0.8]\lambda\in[0.5,0.8]λ∈[0.5,0.8].

2.8 Score Final de Ranking

scoreu,f=β1·simcos+β2·r^u,f+β3·P(click)+β4·ucbu,f+β5·wt − β6·penrisk\text{score}_{u,f}=
\beta_1 \cdot \text{sim}_{\cos} +
\beta_2 \cdot \hat{r}_{u,f} +
\beta_3 \cdot P(\text{click}) +
\beta_4 \cdot \text{ucb}_{u,f} +
\beta_5 \cdot w_t
\ - \ \beta_6 \cdot \text{pen}_{risk}

scoreu,f=β1·simcos+β2·r^u,f+β3·P(click)+β4·ucbu,f+β5·wt − β6·penrisk

Sugerido: β={0.25,0.20,0.25,0.10,0.10,0.10}\beta=\{0.25,0.20,0.25,0.10,0.10,0.10\}β={0.25,0.20,0.25,0.10,0.10,0.10}.

---

3) Pipeline Assíncrono (Tempo Real)

Tópicos Kafka

- `ux.view`, `ux.click`, `ux.dwell`, `feed.new_post`, `graph.edge_event`, `security.risk_update`, `model.feedback`.

Consumidores

- `recall-candidate-gen`: ANN/FAISS ou PGVector para top-N candidatos.

- `real-time-features`: atualiza features e contagens com decaimento.
- `ranker-service`: computa scoreu,f\text{score}_{u,f}scoreu,f, aplica MMR, grava Top-K em Redis.

SLA

- p95 **< 120 ms** (cache + ANN).
- Throughput alvo: **> 5k req/s** por zona.

---

4) Contratos de API

4.1 GET /api/ia/recomendacao

- **Query**: `user_id` (req), `k` (opt, default=10), `context` (opt: `feed|connect|feature`), `ab_bucket` (opt).
- **200 Response**

```
{
  "user_id": "u_123",
  "k": 10,
  "context": "feed",
  "items": [
    {
      "id": "f_882",
      "score": 0.8732,
      "features": {
        "sim_cos": 0.81,
        "r_hat": 0.62,
        "ctr_p": 0.57,
        "ucb": 0.11,
        "time_decay": 0.93,
        "risk_pen": 0.00
      },
      "reason": ["similar_usage", "fresh", "diverse"]
    }
  ],
  "latency_ms": 37
}
```

- **Códigos**: `200, 400 (invalid_user), 401, 404 (cold_start), 429, 500`.

4.2 POST /api/ia/feedback-sugestao

```
{
  "user_id": "u_123",
  "item_id": "f_882",
  "action": "click|dismiss|block",
  "context": "feed",
  "ts": "2025-09-02T14:43:21Z"
}
```

- **200**: `{ "ack": true }`

4.3 GET /api/ia/vetor-comportamento

- **200**:

```
{ "user_id":"u_123","embedding":[-0.014,0.233,...],"updated_at":"2025-09-02T14:39:01Z" }
```

Headers padrão

```
Authorization: Bearer <JWT>
X-Trace-Id: <uuid>
Content-Type: application/json
```

---

5) Modelo de Dados (DDL)

```
CREATE TABLE user_embedding (
  user_id VARCHAR PRIMARY KEY,
  dim SMALLINT NOT NULL DEFAULT 128,
  vec VECTOR(128),        -- PGVector
  updated_at TIMESTAMP NOT NULL DEFAULT NOW()
);

CREATE TABLE feature_embedding (
  feature_id VARCHAR PRIMARY KEY,
  type VARCHAR(24) NOT NULL,  -- user|content|feature
  vec VECTOR(128),
  updated_at TIMESTAMP NOT NULL DEFAULT NOW()
);

CREATE TABLE interaction_log (
  id BIGSERIAL PRIMARY KEY,
  user_id VARCHAR NOT NULL,
  item_id VARCHAR NOT NULL,
  action VARCHAR(16) NOT NULL, -- view|click|dismiss|block
  context VARCHAR(16) NOT NULL,
  ts TIMESTAMP NOT NULL DEFAULT NOW()
);
CREATE INDEX ix_interaction_user_ts ON interaction_log(user_id, ts DESC);

CREATE TABLE suggestion_log (
  id BIGSERIAL PRIMARY KEY,
  user_id VARCHAR NOT NULL,
  item_id VARCHAR NOT NULL,
  score NUMERIC(6,4) NOT NULL,
  features JSONB NOT NULL,
  context VARCHAR(16) NOT NULL,
  trace_id UUID NOT NULL,
```

```
  served_at TIMESTAMP NOT NULL DEFAULT NOW()
);
CREATE INDEX ix_suggestion_user_served ON suggestion_log(user_id, served_at DESC);
```

Redis (chaves)

- `rec:topk:{user_id}:{context}` → ZSET (item_id, score), TTL 5–15 min.
- `stat:ctr:{item_id}` → contagens decaídas.
- `feat:recent:{user_id}` → janela deslizante de eventos.

---

6) Seleção & Re-ranking (Pseudocódigo)

```
def topk_recommend(user_id, k=10, context="feed"):
    u = load_user_vec(user_id)                # PGVector
    C = ann_candidates(u, context, N=200)     # FAISS / ivfflat
    feats = build_features(user_id, C)        # sim_cos, r_hat, ctr_p, ucb, time_decay, risk_pen
    scores = {f: score(feats[f]) for f in C}  # fórmula 2.8
    S = mmr_rerank(u, scores, C, lambda_=0.7)  # diversidade
    S = policy_filter(S, context)             # bloqueio, cap de repetição
    cache_redis(user_id, context, S[:k])
    log_serving(user_id, S[:k])
    return S[:k]
```

Cold-Start

- Se `!u`: usar **popularidade contextual** + **similaridade por atributos declarados** (idade, região, device) + exploração maior (λ↓\lambda\downarrowλ↓, c↑c\uparrowc↑).

---

7) Treino/Atualização de Modelos

- **Offline (diário)**: fatoração (ALS/LightFM), ajuste de embeddings, treinos de CTR (logistic/GBDT).
- **Streaming**: atualização de contagens/ucb/decay via consumidores `model.feedback`.
- **Guardrails**: não promover modelos sem aprovação (Conselho de Guardiões) e avaliação A/B segura.

---

8) Métricas & SLAs

- **p95 latency** `/recomendacao`: **< 80 ms** (cache hit), **< 180 ms** (miss).
- **Precision@10 ≥ 0.25**, **NDCG@10 ≥ 0.32** (baseline inicial).
- **CTR uplift A/B** mínimo: **+3%** antes de promoção.
- **Taxa de repetição** no Top-K **< 30%** (diversidade).

---

9) Regras de Negócio & Compliance

- Filtro de risco: bloquear itens/usuários com `risk_score ≥ 0.8`.
- Cap de frequência por item: no máximo **2** aparições por sessão.

- Respeito a exclusões do usuário: **block/hidden** tem precedência absoluta.
- Rate limit: `60 req/min` por usuário (429).

---

10) Observabilidade

Logs (JSON-line)

```
{
  "type":"serve",
  "trace_id":"c8f6-...-91a",
  "user_id":"u_123",
  "context":"feed",
  "items":[["f_882",0.8732],["f_441",0.8451]],
  "latency_ms":37,
  "ab_bucket":"B",
  "ts":"2025-09-02T15:02:11Z"
}
```

Métricas Prometheus

- `rec_request_latency_ms{context}`, `rec_cache_hit_ratio`, `rec_diversity_index`.

---

11) Erros Padrão

- `400` invalid params; `401` auth; `404` cold_start (usar fallback); `409` conflito de estado; `429` rate limit; `500/502` falha de ranker/ANN.

---

12) Segurança

- JWT + RBAC; **Trace-ID** obrigatório; **PII minimizada** nos logs; anonimização para treino; quotas por IP/app.

🧠 CAMADA 04 — IA DE PERFORMANCE & INFRAESTRUTURA (100% TÉCNICA)

1) Objetivo

Garantir **baixa latência**, **alta disponibilidade** e **uso otimizado de recursos** em todos os módulos da IA Operacional. A camada de performance deve atuar de forma **proativa**, prevendo gargalos e escalando recursos antes que o usuário perceba lentidão.

---

2) Fórmulas e Métricas-Chave

2.1 Latência Média Global

latencia_media=∑i=1Ntresposta(i)N\text{latencia_media} = \frac{\sum_{i=1}^{N} t_{resposta}(i)}{N}

latencia_media=N∑i=1Ntresposta(i)

2.2 SLA de Latência

- p95: **< 250ms**
- p99: **< 400ms**

2.3 Taxa de Erro

taxa_erro=erros_5xx+erros_4xx_criticostotal_req\text{taxa_erro} = \frac{\text{erros_5xx}+\text{erros_4xx_criticos}}{\text{total_req}}

taxa_erro=total_reqerros_5xx+erros_4xx_criticos

Meta: **< 0.5%** no p95.

2.4 SLO e Orçamento de Erros

erro_budget=(1−SLO)×total_req_mensal\text{erro_budget} = (1 - SLO) \times \text{total_req_mensal}

erro_budget=(1−SLO)×total_req_mensal

---

3) Autoscaling e Balanceamento

Kubernetes HPA (Horizontal Pod Autoscaler):

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: ia-performance-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: ia-performance
  minReplicas: 3
  maxReplicas: 50
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 70
  - type: Resource
    resource:
      name: memory
      target:
        type: Utilization
        averageUtilization: 75
```

Balanceamento

- **Istio** para roteamento inteligente.
- **Rate limiting** por namespace.
- **Priority queues** para rotas críticas (ex: `/api/ia/core/status`).

---

4) Predição de Gargalos (Streaming)

Modelo Regressão/ARIMA para previsão de latência

t^resposta(t+1)=α·tresposta(t)+β·tresposta(t−1)+ϵ\hat{t}_{resposta}(t+1)=\alpha \cdot t_{resposta}(t)+\beta \cdot t_{resposta}(t-1)+\epsilon

t^resposta(t+1)=α·tresposta(t)+β·tresposta(t−1)+ϵ

Consumidor Kafka: `perf.latency_stream`

---

5) Contratos de API — Health e Métricas

5.1 GET /api/ia/performance/health

- **200**:

```
{
  "status": "ok",
  "uptime": "99.982",
  "latency_p95": 183,
  "error_rate": 0.002
}
```

5.2 GET /api/ia/performance/metrics

```
{
  "cpu_utilization": 63,
  "memory_utilization": 72,
  "autoscale_events": 3,
  "requests_per_second": 5021,
  "queue_depth": 127
}
```

5.3 POST /api/ia/performance/simulate-load

- **Body**:

```
{ "qps": 10000, "duration_sec": 300 }
```

---

6) Estrutura de Dados

```
CREATE TABLE perf_metrics (
  id BIGSERIAL PRIMARY KEY,
  pod_id VARCHAR NOT NULL,
  cpu_usage NUMERIC(5,2),
  mem_usage NUMERIC(5,2),
  latency_p95 NUMERIC(6,2),
  error_rate NUMERIC(5,4),
  timestamp TIMESTAMP DEFAULT NOW()
);
```

```
CREATE INDEX ix_perf_timestamp ON perf_metrics(timestamp DESC);
```

Redis:

- `perf:qps:{region}` → contagem de QPS atualizada a cada 5s.
- `perf:latency:p95` → histograma com TTL curto.

---

7) Pseudocódigo — Orquestração

```
def monitorar_perf():
    metrics = coletar_metrics()
    if metrics['latency_p95'] > 250 or metrics['cpu_util'] > 70:
        escalar_pods(service="ia-performance")
    if metrics['error_rate'] > 0.005:
        alertar_devops()
```

---

8) Logs Técnicos

Formato JSON-line:

```
{
  "type": "perf_event",
  "service": "ia-performance",
  "latency_p95": 271,
  "cpu": 82,
  "mem": 69,
  "action": "scale_up",
  "timestamp": "2025-09-02T15:30:19Z"
}
```

---

9) Métricas Observáveis

- `perf_latency_ms{p="95"}`
- `perf_qps_total`
- `perf_cpu_utilization`
- `perf_autoscale_events_total`

---

10) Regras de Failover

- Queda de 1 nó → failover em < 2s.
- Multi-cloud redundancy (AWS + GCP).
- DB replicado cross-region.

💡 Fechamento da Camada

A IA de Performance & Infraestrutura garante que todos os outros módulos funcionem sem gargalos, com métricas claras, SLAs definidos e pipelines de autoajuste.

🧠 CAMADA 05 — IA DE SEGURANÇA OPERACIONAL (100% TÉCNICA)

1) Objetivo

Detectar **anomalias funcionais e técnicas**, bloquear **comportamentos suspeitos** e proteger **APIs, dados e fluxos** em tempo real.

A IA de Segurança é responsável por manter a **integridade operacional**, sem interferir diretamente na camada vibracional (Aurah Kosmos).

---

2) Fórmulas de Score de Risco

2.1 Score de Risco Operacional

score_risco=(w1·incoerencia)+(w2·hesitacao)+(w3·padrao_suspeito)−(w4·integridade_duc)score_risco = (w1 \cdot incoerencia) + (w2 \cdot hesitacao) + (w3 \cdot padrao_suspeito) - (w4 \cdot integridade_duc)

score_risco=(w1·incoerencia)+(w2·hesitacao)+(w3·padrao_suspeito)−(w4·integridade_duc)

- `incoerencia` = divergência entre ações consecutivas (%).
- `hesitacao` = (tresposta_user−tresposta_base)(t_{resposta_user} - t_{resposta_base}) (tresposta_user−tresposta_base).
- `padrao_suspeito` = probabilidade calculada por classificador (0–1).
- `integridade_duc` = score documental de verificação do usuário.
- Pesos recomendados: **w1=0.3, w2=0.2, w3=0.4, w4=0.1**.
- Limite crítico: **score_risco ≥ 0.8 → bloqueio imediato**.

---

3) Algoritmos de Detecção

Tipo	Técnica	Framework
Detecção de Anomalia	Isolation Forest, PyOD	Python/TensorFlow
Padrões Suspeitos	Random Forest supervisionado	Scikit-learn
Risco Temporal	ARIMA p/ detecção de repetição	statsmodels
Bot Detection	Análise de entropia + taxa de requests	Go + Kafka Consumer

---

4) Contratos de API

4.1 POST /api/ia/security/analyze

- **Body**:

```
{
  "user_id": "u_123",
  "ip": "201.11.22.33",
  "actions": ["login", "feed_open", "msg_send"],
  "ts": "2025-09-02T12:44:09Z"
```

```
}
```

- **200 Response**:

```
{
  "user_id": "u_123",
  "score_risco": 0.82,
  "status": "blocked",
  "reason": ["incoerencia_alta","bot_like_pattern"],
  "timestamp": "2025-09-02T12:44:09Z"
}
```

4.2 GET /api/ia/security/logs?user_id=u_123

- **200 Response**:

```
[
  {
    "event": "login_anomaly",
    "score": 0.76,
    "action": "challenge",
    "ts": "2025-09-01T21:13:02Z"
  }
]
```

---

5) Estrutura de Banco de Dados

```
CREATE TABLE security_logs (
  id BIGSERIAL PRIMARY KEY,
  user_id VARCHAR NOT NULL,
  ip_address INET,
  action VARCHAR(50),
  score_risco NUMERIC(5,3),
  status VARCHAR(20), -- allowed|challenge|blocked
  reason JSONB,
  created_at TIMESTAMP DEFAULT NOW()
);

CREATE INDEX ix_security_user_ts ON security_logs(user_id, created_at DESC);
```

Redis:

- `sec:score:{user_id}` → cache de risco (TTL 15min).
- `sec:blocked:ips` → blacklist temporária.

---

6) Pseudocódigo Simplificado

```
def avaliar_risco(user_id, actions, ip):
    incoerencia = calc_incoerencia(actions)
    hesitacao = medir_hesitacao(actions)
    padrao = classificador.predict_proba(actions)[1]
    integridade = buscar_integridade_duc(user_id)

    score = (0.3*incoerencia + 0.2*hesitacao + 0.4*padrao - 0.1*integridade)

    if score >= 0.8:
        bloquear(user_id, ip)
        log_event(user_id, score, "blocked")
    elif score >= 0.6:
        acionar_challenge(user_id)
        log_event(user_id, score, "challenge")
    else:
        permitir(user_id)
```

---

7) Logs Técnicos

Exemplo JSON-line:

```
{
  "type": "security_event",
  "user_id": "u_123",
  "ip": "201.11.22.33",
  "score_risco": 0.82,
  "status": "blocked",
  "reasons": ["pattern_bot","login_loop"],
  "ts": "2025-09-02T12:45:09Z"
}
```

---

8) Métricas Observáveis

- `sec_risk_score_avg`
- `sec_blocked_users_total`
- `sec_false_positive_rate`
- `sec_challenge_triggered_total`

---

9) Políticas de Segurança

- **Latência crítica** → resposta de bloqueio em **< 50ms**.
- **Falsos positivos** monitorados; meta **< 5%**.
- **Logs criptografados** (AES-256).

- **Tokens JWT invalidados** após bloqueio.

💡 Fechamento da Camada

A IA de Segurança Operacional garante a **blindagem funcional** do FriendApp, com base em **fórmulas de risco, algoritmos de anomalia, APIs de resposta rápida e logs rastreáveis**.

🧠 CAMADA 06 — IA DE JORNADA FUNCIONAL E UX ADAPTATIVA (100% TÉCNICA)

1) Objetivo

Monitorar a navegação do usuário, identificar **fricções e abandonos** e aplicar **ajustes dinâmicos de interface (UX)** de forma automática, garantindo fluidez de uso e redução de atrito.

---

2) Fórmulas e Scores de Fricção

2.1 Score de Fricção (SF)

$$SF = (w1 \cdot abandono) + (w2 \cdot hesitacao) + (w3 \cdot cliques\_inconclusivos) + (w4 \cdot scroll\_incompleto)$$

SF=(w1·abandono)+(w2·hesitacao)+(w3·cliques_inconclusivos)+(w4·scroll_incompleto)

- `abandono` = proporção de fluxos não concluídos.
- `hesitacao` = tempo médio de resposta acima do baseline.
- `cliques_inconclusivos` = cliques em botões inválidos/fechamento precoce.
- `scroll_incompleto` = percentual de páginas não roladas até 80%.
- Pesos sugeridos: w1=0.4, w2=0.3, w3=0.2, w4=0.1.

Thresholds:

- SF ≥ 0.7 → **Ajuste automático obrigatório**.
- 0.4 ≤ SF < 0.7 → **Sugestão leve de ajuste**.

---

3) Algoritmos de Ajuste UX

Tipo de Ajuste	Técnica	Framework
Reordenação de menus	A/B testing + reinforcement learning	TensorFlow RL
Sugestão de tela inicial	Clustering de navegação	KMeans
Substituição de texto	Multivariant A/B	Split.io
Reestruturação de fluxo	Graph traversal adaptativo	NetworkX
Scroll adaptativo	Heatmaps + thresholds	UX Engine

---

4) Contratos de API

4.1 GET /api/ia/jornada/analisar

- **Query**: `user_id`, `sessao_id`
- **200 Response**:

```
{
  "user_id": "u_123",
  "sessao_id": "s_991",
  "score_friccao": 0.74,
  "ajustes_recomendados": ["reordenar_menu", "alterar_tela_inicial"],
  "timestamp": "2025-09-02T15:21:01Z"
}
```

4.2 POST /api/ia/jornada/aplicar

- **Body**:

```
{
  "user_id": "u_123",
  "ajustes": [
    { "tipo": "reordenar_menu", "novo_layout": ["mapa","feed","chat"] },
    { "tipo": "substituir_texto", "componente": "cta", "valor": "Conectar Agora" }
  ]
}
```

- **200 Response**:

```
{ "status": "aplicado", "trace_id": "9f3d-889d-..." }
```

---

5) Estrutura de Banco

```
CREATE TABLE journey_sessions (
  sessao_id VARCHAR PRIMARY KEY,
  user_id VARCHAR NOT NULL,
  inicio TIMESTAMP NOT NULL,
  fim TIMESTAMP,
  abandono BOOLEAN DEFAULT FALSE
);

CREATE TABLE journey_events (
  id BIGSERIAL PRIMARY KEY,
  sessao_id VARCHAR NOT NULL,
  evento VARCHAR(50) NOT NULL,
  valor JSONB,
  ts TIMESTAMP DEFAULT NOW()
);

CREATE TABLE journey_adjustments (
  id BIGSERIAL PRIMARY KEY,
  user_id VARCHAR NOT NULL,
  tipo VARCHAR(50) NOT NULL,
```

```
  detalhe JSONB,
  aplicado_em TIMESTAMP DEFAULT NOW()
);
```

Redis:

- `journey:score:{user_id}` → SF atual (TTL: 30min).
- `journey:adjust:queue` → fila de ajustes pendentes.

---

6) Pseudocódigo

```
def avaliar_jornada(sessao):
    abandono = calc_abandono(sessao)
    hesitacao = tempo_medio_resposta(sessao)
    cliques_inconclusivos = contar_cliques_invalidos(sessao)
    scroll_incompleto = medir_scroll(sessao)

    SF = 0.4*abandono + 0.3*hesitacao + 0.2*cliques_inconclusivos + 0.1*scroll_incompleto

    if SF >= 0.7:
        aplicar_ajustes(sessao.user_id)
    elif SF >= 0.4:
        recomendar_ajustes(sessao.user_id)
```

---

7) Logs Técnicos

```
{
  "type": "journey_event",
  "user_id": "u_123",
  "sessao_id": "s_991",
  "score_friccao": 0.74,
  "ajustes": ["reordenar_menu"],
  "timestamp": "2025-09-02T15:27:59Z"
}
```

---

8) Métricas Observáveis

- `journey_abandono_rate`
- `journey_score_friccao_avg`
- `journey_adjustments_applied_total`
- `journey_ux_conversion_uplift`

---

9) SLAs

- Cálculo do **score_friccao**: < **100ms** por sessão.

- Aplicação de ajuste: < **200ms** após decisão.

💡 Fechamento da Camada

A IA de Jornada Funcional garante que cada navegação seja **otimizada em tempo real**, com métricas claras, logs técnicos rastreáveis e ajustes aplicados automaticamente.

🧠 CAMADA 07 — DADOS PROCESSADOS E ESTRUTURA DE APRENDIZADO (100% TÉCNICA)

1) Objetivo

Definir quais **dados brutos** são coletados, como são **armazenados**, **transformados em features** e usados pela IA Operacional em **modelos de aprendizado supervisionado, não supervisionado e reforço**.

---

2) Categorias de Dados

Categoria	Exemplos	Frequência de Captura	Destino
Funcionais	cliques, telas visitadas	tempo real	Redis + PostgreSQL
Temporais	hora, duração sessão, fusos	evento	PostgreSQL
Comportamentais	padrões de abandono, loops, hesitação	sessão completa	Firestore
Engajamento	curtidas, matches, tempo de rolagem	evento	PostgreSQL
Feedbacks	avaliações, bloqueios, denúncias	sob demanda	Firestore
Técnicos	erros, travamentos, latência	tempo real	ElasticSearch
Segurança	logins, IP, DUC/DCO, tokens	evento crítico	PostgreSQL Criptografado

---

3) Fórmulas de Atualização

3.1 Score de Engajamento

scoreengajamento(u)=w1·interacoes+w2·tempo_sessaow3+abandonoscore_{engajamento}(u) = \frac{w1 \cdot interacoes + w2 \cdot tempo_sessao}{w3 + abandono}

scoreengajamento(u)=w3+abandonow1·interacoes+w2·tempo_sessao

3.2 Decaimento Temporal

valorajustado=valor·e−λΔtvalor_{ajustado} = valor \cdot e^{-\lambda \Delta t}

valorajustado=valor·e−λΔt

3.3 Score de Reforço (Recompensa)

R(u,a)=clique(a)−penalidade(rejeicao)+bonus(diversidade)R(u,a) = clique(a) - penalidade(rejeicao) + bonus(diversidade)

R(u,a)=clique(a)−penalidade(rejeicao)+bonus(diversidade)

---

4) Pipeline de Ingestão (Assíncrono)

Kafka Tópicos

- `ux.events.click`
- `ux.events.view`

- `journey.abandono`
- `feed.engagement`
- `sec.login`
- `perf.latency`

Consumidores

- `feature-builder` → transforma eventos em vetores.
- `etl-persist` → grava no PostgreSQL.
- `real-time-cache` → atualiza Redis para uso imediato.

---

5) Estrutura de Banco de Dados

```
CREATE TABLE user_behavior_log (
  id BIGSERIAL PRIMARY KEY,
  user_id VARCHAR NOT NULL,
  evento VARCHAR(50) NOT NULL,
  valor JSONB,
  ts TIMESTAMP DEFAULT NOW()
);

CREATE TABLE feature_vectors (
  user_id VARCHAR NOT NULL,
  context VARCHAR(30),
  vec VECTOR(128),
  updated_at TIMESTAMP DEFAULT NOW(),
  PRIMARY KEY (user_id, context)
);

CREATE TABLE feedback_log (
  id BIGSERIAL PRIMARY KEY,
  user_id VARCHAR NOT NULL,
  item_id VARCHAR NOT NULL,
  action VARCHAR(20), -- click|dismiss|block
  ts TIMESTAMP DEFAULT NOW()
);
```

Redis:

- `feat:{user_id}:{context}` → vetor + TTL.
- `event:last:{user_id}` → último evento por usuário.

---

6) Pseudocódigo de Atualização

```
def processar_evento(evento):
    vetor = carregar_vetor(evento.user_id)
    features = extrair_features(evento)
    vetor_atualizado = atualizar(vetor, features, decay=True)
```

```
    salvar_pg(evento.user_id, vetor_atualizado)
    cache_redis(evento.user_id, vetor_atualizado)
```

---

7) Logs Técnicos

Exemplo JSON-line:

```
{
  "type": "data_ingest",
  "user_id": "u_123",
  "evento": "click",
  "valor": { "feature": "feed_open" },
  "context": "feed",
  "vec_norm": 0.882,
  "ts": "2025-09-02T16:19:43Z"
}
```

---

8) Métricas Observáveis

- `data_ingest_latency_ms` → latência de ingestão (<50ms).
- `feature_vector_updates_total` → número de vetores atualizados.
- `etl_persist_failures_total` → falhas de persistência.
- `redis_cache_hit_ratio` → alvo > 85%.

---

9) SLAs

- Processamento de evento: **< 50ms**.
- Persistência no PostgreSQL: **< 100ms**.
- Atualização em Redis: **instantânea (< 10ms)**.

💡 Fechamento da Camada

A estrutura de dados garante que toda ação do usuário seja **captada, transformada e disponibilizada em tempo real** para os módulos de IA, sustentando aprendizado contínuo e decisões instantâneas.

🧠 CAMADA 08 — MÓDULOS DA IA OPERACIONAL (100% TÉCNICA)

1) Objetivo

Descrever a **arquitetura modular da IA Operacional**, detalhando os núcleos independentes, suas responsabilidades, APIs dedicadas e a comunicação assíncrona entre eles.

---

2) Tabela de Módulos

Módulo	Responsabilidade Principal	Output Técnico	APIs Chave
IA Recomendação	Sugestões de recursos, conexões e conteúdos	Lista Top-K de itens com scores	`/api/ia/recomendacao`, `/feedback`
IA Performance	Monitorar e otimizar latência/infraestrutura	Métricas p95/p99, triggers de scaling	`/api/ia/performance/metrics`
IA Segurança	Detectar padrões suspeitos e fraudes	Score de risco, logs de bloqueio	`/api/ia/security/analyze`
IA Jornada	Ajustes de UX e fluxo de navegação	Score_friccao, alterações aplicadas	`/api/ia/jornada/analisar`
IA Vibracional	Leitura de frequência e estados (proxy Aurah)	Mapas de energia, alertas	`/api/ia/vibracional/status`
IA Moderadora	Detectar abuso/linguagem nociva	Flags, relatórios	`/api/ia/moderador/analyze`
IA Sensorial	Ajustes de feed e posts	Score vibracional de conteúdo	`/api/ia/feed/analisar`
IA Missões	Propor desafios e transmutação	Missões e recompensas	`/api/ia/missoes/gerar`
IA Global	Consolidar aprendizado intermodular	Vetores unificados, logs	`/api/ia/global/status`

3) Comunicação Entre Módulos

- **Event Bus (Kafka)** → cada módulo consome apenas eventos relevantes.
- **Redis** → cache compartilhado de scores e estados.
- **gRPC interno** → baixa latência para chamadas síncronas críticas (ex: segurança).

4) Fluxo de Interação

```
graph TD
A[Kafka UX Events] --> R[IA Recomendação]
A --> J[IA Jornada]
A --> S[IA Segurança]
A --> P[IA Performance]
R --> B[Redis Cache]
J --> B
S --> PG[(PostgreSQL Security)]
P --> E[(ElasticSearch Metrics)]
R --> API[/API Rest - Sugestões/]
J --> API
S --> API
P --> API
```

5) Estrutura de Banco de Dados Multi-Módulo

```
-- Logs de recomendações
CREATE TABLE rec_suggestions (
  id BIGSERIAL PRIMARY KEY,
  user_id VARCHAR NOT NULL,
```

```
  item_id VARCHAR NOT NULL,
  score NUMERIC(6,4),
  context VARCHAR(30),
  ts TIMESTAMP DEFAULT NOW()
);

-- Logs de segurança
CREATE TABLE sec_events (
  id BIGSERIAL PRIMARY KEY,
  user_id VARCHAR NOT NULL,
  score NUMERIC(5,3),
  status VARCHAR(20),
  reasons JSONB,
  ts TIMESTAMP DEFAULT NOW()
);

-- Ajustes de jornada
CREATE TABLE journey_adjustments (
  id BIGSERIAL PRIMARY KEY,
  user_id VARCHAR NOT NULL,
  tipo VARCHAR(30),
  detalhe JSONB,
  aplicado_em TIMESTAMP DEFAULT NOW()
);
```

Redis:

- `mod:{modulo}:{user_id}` → cache por módulo.
- `events:queue:{modulo}` → fila de eventos a processar.

---

6) Pseudocódigo de Orquestração

```
def processar_evento(evento):
    if evento.tipo in ["click","view","feed_post"]:
        publicar_kafka("rec.input", evento)
    if evento.tipo in ["abandono","scroll_incompleto"]:
        publicar_kafka("journey.input", evento)
    if evento.tipo in ["login","ip_change","request_loop"]:
        publicar_kafka("security.input", evento)
    publicar_kafka("performance.input", evento)
```

---

7) Logs Técnicos Intermodulares

Exemplo JSON-line:

```
{
  "type": "ia_interaction",
  "user_id": "u_234",
```

```
  "modules_triggered": ["rec","journey","security"],
  "latency_ms": 92,
  "trace_id": "d8b3c3c4-2b11-442b-94d2-cc123",
  "ts": "2025-09-02T16:45:32Z"
}
```

---

8) Métricas Observáveis

- `module_event_latency_ms{module}`
- `module_event_throughput_total{module}`
- `module_interaction_failures_total`
- `module_cache_hit_ratio{module}`

💡 Fechamento da Camada

A arquitetura modular garante **independência, resiliência e escalabilidade**. Cada módulo tem APIs próprias, dados isolados e se comunica por eventos assíncronos, evitando pontos únicos de falha.

🧠 CAMADA 09 — ALGORITMOS DE DECISÃO E COMPATIBILIDADE (100% TÉCNICA)

1) Objetivo

Definir matematicamente os algoritmos que determinam **compatibilidade, decisões adaptativas e correções automáticas** da IA Operacional. Essa camada fornece as fórmulas, pseudocódigos e logs necessários para execução direta pelos devs.

---

2) Fórmulas Matemáticas

2.1 Compatibilidade Vibracional + Social

score_compat(u,v)=α·simcos(u,v)+β·hist_interacoes(u,v)+γ·contexto(u,v)score_compat(u,v) = \alpha \cdot sim_{\cos}(\mathbf{u},\mathbf{v}) + \beta \cdot hist_interacoes(u,v) + \gamma \cdot contexto(u,v)

score_compat(u,v)=α·simcos(u,v)+β·hist_interacoes(u,v)+γ·contexto(u,v)

- `sim_cos` = similaridade cosseno entre vetores de uso.
- `hist_interacoes` = número de interações positivas (mensagens, matches).
- `contexto` = ajuste de região/idioma/dispositivo.
- Pesos sugeridos: α=0.5, β=0.3, γ=0.2.

2.2 Decisão Adaptativa (UX)

score_ajuste=(w1·abandono)+(w2·hesitacao)+(w3·erro_fluxo)score_ajuste = (w1 \cdot abandono) + (w2 \cdot hesitacao) + (w3 \cdot erro_fluxo)

score_ajuste=(w1·abandono)+(w2·hesitacao)+(w3·erro_fluxo)

Se `score_ajuste ≥ 0.6` → aplicar ajuste dinâmico (nova tela, botão direto).

2.3 Correção Energética

corrigir(u)={true,se vib(u)<0.3∨risco(u)≥0.8false,caso contra´riocorrigir(u) =
\begin{cases}
true, & \text{se } vib(u) < 0.3 \lor risco(u) \geq 0.8 \\
false, & \text{caso contrário}
\end{cases}

corrigir(u)={true,false,se vib(u)<0.3∨risco(u)≥0.8caso contra´rio

3) Algoritmos-Chave

Algoritmo	Função	Tipo de IA
MATCH_SOMA_V1	Soma de afinidades energéticas e sociais	Similaridade + Colab
ADAPT_FLOW	Ajuste de caminho em tempo real	RL + UX Engine
BLOQUEIO_DUAL	Pausa automática para incoerência	Anomaly Detection
LOGICA_KARMICA_AI	Reforço de padrões positivos	Reforço supervisionado
AURA_SWITCH	Alteração dinâmica de recomendações	Classificação temporal

4) Pseudocódigo

```
def calcular_compatibilidade(user_a, user_b):
    sim_cos = cosine_similarity(vec(user_a), vec(user_b))
    hist = historico_interacoes(user_a, user_b)
    contexto = contexto_ajuste(user_a, user_b)

    score = 0.5*sim_cos + 0.3*hist + 0.2*contexto

    if score >= 0.75:
        return "alta"
    elif score >= 0.5:
        return "media"
    else:
        return "baixa"
```

```
def ajuste_fluxo(sessao):
    abandono = calc_abandono(sessao)
    hesitacao = medir_hesitacao(sessao)
    erro = contar_erros(sessao)
    score = 0.4*abandono + 0.3*hesitacao + 0.3*erro

    if score >= 0.6:
        aplicar_ajuste(sessao)
```

5) Estrutura de Banco

```
CREATE TABLE compat_scores (
  id BIGSERIAL PRIMARY KEY,
```

```
  user_a VARCHAR NOT NULL,
  user_b VARCHAR NOT NULL,
  score NUMERIC(5,3),
  categoria VARCHAR(20), -- alta|media|baixa
  ts TIMESTAMP DEFAULT NOW()
);

CREATE TABLE ajustes_ux (
  id BIGSERIAL PRIMARY KEY,
  user_id VARCHAR NOT NULL,
  tipo VARCHAR(30),
  score NUMERIC(5,3),
  aplicado BOOLEAN DEFAULT FALSE,
  ts TIMESTAMP DEFAULT NOW()
);
```

---

6) Logs Técnicos

Exemplo JSON-line:

```
{
  "type": "compat_decision",
  "user_a": "u_123",
  "user_b": "u_456",
  "score": 0.78,
  "categoria": "alta",
  "reason": ["sim_cos=0.81","hist=0.65","contexto=0.52"],
  "ts": "2025-09-02T17:08:14Z"
}
```

---

7) Métricas Observáveis

- `compatibility_avg_score`
- `compatibility_high_matches_total`
- `ux_adjustments_triggered_total`
- `risk_corrections_applied_total`

---

8) SLAs

- Cálculo de compatibilidade: **< 50ms** por par.
- Ajuste de UX: **< 150ms** após trigger.
- Correção de risco: **< 50ms** resposta imediata.

💡 Fechamento da Camada

Esta camada traduz compatibilidade e decisões adaptativas em **fórmulas matemáticas, pseudocódigos e logs rastreáveis**, garantindo execução técnica consistente.

🧠 CAMADA 10 — APRENDIZADO CONTÍNUO E AUTOEVOLUÇÃO DA IA (100% TÉCNICA)

1) Objetivo

Definir o pipeline de **aprendizado contínuo** da IA Operacional, garantindo evolução dos modelos com dados em tempo real, sem comprometer a estabilidade do sistema. Inclui fórmulas de atualização, arquitetura de treino e **governança humana obrigatória**.

---

2) Fórmulas de Atualização

2.1 Atualização Online de Vetores

ut+1=(1−η)·ut+η·xt\mathbf{u}_{t+1} = (1 - \eta) \cdot \mathbf{u}_t + \eta \cdot \mathbf{x}_t

ut+1=(1−η)·ut+η·xt

- `η`: taxa de aprendizado (0.01–0.05).
- `x_t`: novo vetor de interação.

2.2 Ajuste de Peso de Reforço

wi,new=wi+α·(r−r^)w_{i,new} = w_i + \alpha \cdot (r - \hat{r})

wi,new=wi+α·(r−r^)

- `r`: recompensa observada (clique, aceitação).
- `\hat r`: previsão anterior.
- `α`: taxa de atualização (0.1).

2.3 Drift Detector (KS-Test)

DKS=max ⁡∣Fn(x)−F(x)∣D_{KS} = \max |F_{n}(x) - F(x)|

DKS=max ⁡∣Fn(x)−F(x)∣

Se `D_KS > 0.1` → alerta de drift do modelo.

---

3) Pipeline de Treino (Assíncrono)

Kafka Tópicos

- `model.feedback` → interações com outputs.
- `model.performance` → métricas de acurácia, latência.
- `model.drift` → eventos de desvio detectado.

Fluxo

1. Dados ingeridos em tempo real → `feature-store`.
2. Batch diário em Spark/Dataproc → treino offline.
3. Geração de **modelo candidato**.
4. Testes A/B + simulação em sandbox.
5. **Conselho de Guardiões** aprova/reprova.

6. Deploy gradual com **feature flag**.

---

4) Estrutura de Banco de Modelos

```
CREATE TABLE model_registry (
  id SERIAL PRIMARY KEY,
  nome VARCHAR(50) NOT NULL,
  versao VARCHAR(20) NOT NULL,
  status VARCHAR(20) DEFAULT 'candidate', -- candidate|approved|prod
  acuracia NUMERIC(5,3),
  drift_score NUMERIC(5,3),
  aprovado_por VARCHAR(50),
  criado_em TIMESTAMP DEFAULT NOW()
);
```

---

5) APIs

5.1 GET /api/ia/model/status

```
{
  "nome": "rec_model",
  "versao": "1.3.0",
  "status": "prod",
  "acuracia": 0.923,
  "drift_score": 0.07
}
```

5.2 POST /api/ia/model/promote

```
{ "nome": "rec_model", "versao": "1.4.0" }
```

- Exige **aprovação humana** → `403` se Conselho não validar.

---

6) Pseudocódigo do Ciclo

```
def ciclo_aprendizado():
    dados = coletar_feedback()
    modelo_candidato = treinar(dados)
    if drift_detectado(modelo_candidato):
        notificar_guardioes(modelo_candidato)
    if conselho_aprova(modelo_candidato):
        promover(modelo_candidato)
```

---

7) Logs Técnicos

```
{
  "type": "model_update",
  "model": "rec_model",
  "versao": "1.4.0",
  "acuracia": 0.927,
  "drift_score": 0.05,
  "status": "candidate",
  "ts": "2025-09-02T17:22:01Z"
}
```

---

8) Métricas Observáveis

- `model_accuracy{model}`
- `model_drift_score{model}`
- `model_candidate_total`
- `model_promotions_total`
- `model_rejected_total`

---

9) SLAs

- Drift detection: execução **< 500ms**.
- Novo modelo → tempo de treino: **< 6h batch**.
- Deploy gradual: 5% → 25% → 100% em 48h.

💡 Fechamento da Camada

O aprendizado contínuo é matematicamente definido, executado via pipelines assíncronos e validado por **governança humana obrigatória**, garantindo evolução sem perda de controle.

🧠 CAMADA 11 — FLUXOGRAMAS OPERACIONAIS E CICLO DE DECISÃO (100% TÉCNICA)

1) Objetivo

Formalizar os fluxos **end-to-end** de decisão da IA Operacional, desde a captura de eventos até a execução das ações. Definir triggers, prioridades, logs e diagramas técnicos que guiam os módulos.

---

2) Estrutura de Fluxo Geral

```
flowchart TD
A[Evento captado no app] --> B[Kafka - tópico relevante]
B --> C[Feature Builder]
C --> D[IA Módulo específico]
D --> E{Decisão?}
E -->|Sugestão| F[Top-K Output → Redis]
E -->|Correção| G[Trigger ajuste UX/Security]
```

```
E -->|Bloqueio| H[Log + Challenge]
F --> I[API de entrega]
G --> I
H --> I
I --> J[Feedback → Kafka]
J --> C
```

3) Tabela de Eventos → Decisão → Ação

Evento	IA Responsável	Decisão Tomada	Ação Executada
`feed.new_post`	IA Sensorial	Analisar vibração + risco	Expandir alcance ou limitar visibilidade
`journey.abandono`	IA Jornada	Score_friccao ≥ 0.7	Ajustar menu/tela inicial
`sec.login_anomaly`	IA Segurança	Score_risco ≥ 0.8	Bloqueio imediato
`chat.silence_90s`	IA Conexão	Trigger "hesitação"	Sugerir reconexão ou ritual
`perf.latency`	IA Performance	p95 > 250ms	Escalar pods / redistribuir carga

4) Regras de Escalonamento de Decisão

- **Alta prioridade**: Segurança (risco, fraude, colapso) → **resposta < 50ms**.
- **Média**: UX/Jornada (abandono, hesitação) → **resposta < 150ms**.
- **Baixa**: Sugestões/Feed → assíncrono, **p95 < 200ms**.

5) Logs de Ciclo de Decisão

Exemplo JSON-line:

```
{
  "type": "decision_cycle",
  "event": "journey.abandono",
  "user_id": "u_991",
  "module": "journey",
  "score": 0.74,
  "action": "ajuste_tela_inicial",
  "latency_ms": 112,
  "trace_id": "b4a7-77aa-91ff",
  "ts": "2025-09-02T17:41:18Z"
}
```

6) Pseudocódigo

```
def ciclo_decisao(evento):
    features = extrair_features(evento)
    score = calcular_score(evento.tipo, features)

    if evento.tipo == "security" and score >= 0.8:
```

```
    bloquear_usuario(evento.user_id)
elif evento.tipo == "journey" and score >= 0.6:
    aplicar_ajuste_ux(evento.user_id)
elif evento.tipo == "feed":
    gerar_sugestoes(evento.user_id)

log_decisao(evento, score)
enviar_feedback(evento, score)
```

---

7) Métricas Observáveis

- `decision_latency_ms{module}`
- `decision_triggered_total{event}`
- `decision_blocked_total`
- `decision_feedback_loop_ratio`

---

8) SLAs

- **Decisão crítica (segurança)** → resposta em **< 50ms**.
- **Fluxo adaptativo (UX)** → **< 150ms**.
- **Sugestões de feed** → **< 200ms** p95.

💡 Fechamento da Camada

O ciclo de decisão garante que cada evento recebido seja **classificado, processado e respondido** em tempo real, com logs e métricas para auditoria contínua.

🧠 CAMADA 12 — TIPOS DE DADOS CAPTADOS E LÓGICA DE TRATAMENTO (100% TÉCNICA)

1) Objetivo

Mapear **todos os tipos de dados captados pela IA Operacional**, definir a **lógica de limpeza, normalização e enriquecimento**, além de especificar os **fluxos de armazenamento assíncrono**.

---

2) Categorias de Dados Captados

Categoria	Exemplos	Frequência de Captura	Destino Primário
Funcionais	cliques, views, tempo de sessão	tempo real	Redis + PostgreSQL
Comportamentais	abandono, loops, hesitação	sessão completa	Firestore
Temporais	hora, fuso, duração, device	evento	PostgreSQL
Engajamento	curtidas, matches, comentários	evento	PostgreSQL
Feedbacks	avaliações, bloqueios, denúncias	sob demanda	Firestore
Técnicos	erros 5xx, latência, falhas de pods	tempo real	ElasticSearch + Grafana
Segurança	login, IP, device fingerprint, tokens	evento crítico	PostgreSQL Criptografado

---

3) Fórmulas e Tratamento de Dados

3.1 Normalização de Tempo de Resposta

tnorm=tresposta−μσt_{norm} = \frac{t_{resposta} - \mu}{\sigma}

tnorm=σtresposta−μ

3.2 Score de Qualidade de Sessão

Qsessao=1−abandono+erroseventostotaisQ_{sessao} = 1 - \frac{abandono + erros}{eventos_{totais}}

Qsessao=1−eventostotaisabandono+erros

3.3 Decaimento de Evento

pesoevento=e−λ·Δt,λ∈[0.01,0.2]peso_{evento} = e^{-\lambda \cdot \Delta t},\quad \lambda \in [0.01,0.2]

pesoevento=e−λ·Δt,λ∈[0.01,0.2]

3.4 Score de Integridade de Dados

I=eventos_validoseventos_captadosI = \frac{eventos_validos}{eventos_captados}

I=eventos_captadoseventos_validos

Meta: I ≥ **0.95**.

---

4) Pipeline de Processamento (Assíncrono)

Kafka Tópicos

- `ux.raw_events` → captura inicial.
- `ux.clean_events` → dados normalizados.
- `ux.feature_vectors` → envio para Feature Store.
- `ux.error_events` → monitoramento em Elastic.

Consumidores

- `data-cleaner` → remove duplicados, valida schemas.
- `feature-builder` → transforma em vetores.
- `etl-writer` → grava em PostgreSQL/Firestore.

---

5) Estrutura de Banco

```
CREATE TABLE raw_events (
  id BIGSERIAL PRIMARY KEY,
  user_id VARCHAR,
  tipo VARCHAR(50),
  payload JSONB,
  valido BOOLEAN DEFAULT TRUE,
  ts TIMESTAMP DEFAULT NOW()
);

CREATE TABLE cleaned_events (
  id BIGSERIAL PRIMARY KEY,
  user_id VARCHAR,
  tipo VARCHAR(50),
```

```
  features JSONB,
  peso NUMERIC(5,4),
  ts TIMESTAMP DEFAULT NOW()
);

CREATE TABLE session_quality (
  sessao_id VARCHAR PRIMARY KEY,
  user_id VARCHAR,
  q_sessao NUMERIC(5,4),
  eventos INT,
  abandonos INT,
  erros INT,
  ts TIMESTAMP DEFAULT NOW()
);
```

Redis:

- `event:last:{user_id}` → último evento válido.
- `sessao:qualidade:{sessao_id}` → cache do Q_sessao.

---

6) Pseudocódigo de Limpeza

```
def processar_evento(evento):
    if not validar_schema(evento):
        log_evento_invalido(evento)
        return

    evento['peso'] = calcular_peso(evento['timestamp'])
    salvar_pg("cleaned_events", evento)
    atualizar_cache(evento['user_id'], evento)
```

---

7) Logs Técnicos

Exemplo JSON-line:

```
{
  "type": "data_event",
  "user_id": "u_789",
  "evento": "click",
  "valido": true,
  "peso": 0.93,
  "q_sessao": 0.88,
  "trace_id": "83ac-11d9",
  "ts": "2025-09-02T18:04:27Z"
}
```

---

8) Métricas Observáveis

- `eventos_validos_total`
- `taxa_integridade_dados` (meta > 95%)
- `data_ingest_latency_ms` (meta < 50ms)
- `etl_falhas_total`

---

9) SLAs

- Ingestão → **< 50ms/evento**.
- Persistência → **< 100ms**.
- Taxa de integridade mínima → **95%**.

💡 Fechamento da Camada

Todos os dados captados são tratados por **pipelines assíncronos**, normalizados matematicamente, enriquecidos com pesos de tempo e persistidos em bancos híbridos, prontos para alimentar modelos de decisão.

🧠 CAMADA 13 — OUTPUTS INTELIGENTES DA IA E AÇÕES SUGERIDAS (100% TÉCNICA)

1) Objetivo

Formalizar os **outputs que a IA Operacional gera** a partir de decisões internas: sugestões, correções de fluxo, bloqueios e missões. Definir fórmulas de cálculo, contratos de API, estrutura de logs e métricas.

---

2) Fórmulas de Decisão

2.1 Score de Sugestão

$$score_{sugestao} = \beta_1 \cdot compat(u,f) + \beta_2 \cdot engajamento(u) + \beta_3 \cdot tempo\_sessao - \beta_4 \cdot risco(u,f)$$

$$score_{sugestao} = \beta_1 \cdot compat(u,f) + \beta_2 \cdot engajamento(u) + \beta_3 \cdot tempo\_sessao - \beta_4 \cdot risco(u,f)$$

- Pesos típicos: β1=0.4, β2=0.3, β3=0.2, β4=0.1.
- Limite de output: se `score_sugestao ≥ 0.65`.

2.2 Score de Correção UX

$$score_{correcao} = abandono \cdot 0.5 + hesitacao \cdot 0.3 + erros\_fluxo \cdot 0.2$$

$$score_{correcao} = abandono \cdot 0.5 + hesitacao \cdot 0.3 + erros\_fluxo \cdot 0.2$$

- Correção se `score_correcao ≥ 0.6`.

2.3 Score de Pausa Energética

$$pause(u) = \begin{cases} 1, & vib(u) < 0.3 \lor rejeicoes(u) > 5 \\ \text{caso contrário} & pause(u) = \end{cases}$$

$$\begin{cases} 1, & vib(u) < 0.3 \lor rejeicoes(u) > 5 \\ \end{cases}$$

0, & caso\ contrário
\end{cases}

pause(u)={1,0,vib(u)<0.3∨rejeicoes(u)>5caso contra´rio

---

3) Tipos de Outputs

Tipo	Exemplo Técnico	Contexto
Sugestão Ativa	Top-K conexões/recursos → API `/recomendacao`	Feed, Matching
Correção de Fluxo	Ajustes de UX: tela inicial, botões, menus	Jornada
Missão Sensorial	Geração de missão com ID + instruções	Missões
Pausa Energética	Output para modo neutro	Vibracional
Alerta Técnico	Log oculto enviado ao módulo de segurança/perf.	Segurança
Sinalização Interna	Trigger interno (Webhook → outro módulo)	Cross-IA

---

4) Contratos de API

4.1 GET /api/ia/outputs

- **Query**: `user_id`, `context` (feed|journey|security).
- **200 Response**:

```
{
  "user_id": "u_456",
  "context": "feed",
  "outputs": [
    {
      "tipo": "sugestao",
      "item_id": "f_902",
      "score": 0.842,
      "trace_id": "12ab-88ff",
      "features": { "compat": 0.77, "engajamento": 0.68, "risco": 0.05 }
    }
  ],
  "latency_ms": 92
}
```

4.2 POST /api/ia/outputs/feedback

- **Body**:

```
{
  "user_id": "u_456",
  "output_id": "out_882",
  "action": "accept|reject|ignore",
  "ts": "2025-09-02T18:21:19Z"
}
```

---

5) Estrutura de Banco de Dados

```
CREATE TABLE ia_outputs (
  id BIGSERIAL PRIMARY KEY,
  user_id VARCHAR NOT NULL,
  tipo VARCHAR(30) NOT NULL, -- sugestao|correcao|missao|pausa|alerta
  payload JSONB,
  score NUMERIC(6,4),
  context VARCHAR(30),
  trace_id UUID,
  criado_em TIMESTAMP DEFAULT NOW()
);

CREATE TABLE ia_output_feedback (
  id BIGSERIAL PRIMARY KEY,
  output_id BIGINT REFERENCES ia_outputs(id),
  user_id VARCHAR NOT NULL,
  action VARCHAR(20), -- accept|reject|ignore
  ts TIMESTAMP DEFAULT NOW()
);
```

Redis:

- `outputs:recent:{user_id}:{context}` → ZSET com últimos outputs.

---

6) Pseudocódigo

```
def gerar_output(user_id, contexto):
    score = calcular_score(user_id, contexto)
    if score >= threshold(contexto):
        output = criar_output(user_id, contexto, score)
        salvar_pg(output)
        cache_redis(user_id, output)
        return output
    else:
        return None
```

---

7) Logs Técnicos

Exemplo JSON-line:

```
{
  "type": "ia_output",
  "user_id": "u_456",
  "context": "journey",
  "output_type": "correcao",
  "score": 0.71,
  "trace_id": "cd88-9981",
```

```
  "payload": { "ajuste": "tela_inicial" },
  "latency_ms": 127,
  "ts": "2025-09-02T18:27:11Z"
}
```

8) Métricas Observáveis

- `outputs_total{tipo}`
- `output_feedback_accept_ratio`
- `output_latency_ms{context}`
- `outputs_blocked_by_risk_total`

9) SLAs

- Geração de outputs críticos: **< 100ms**.
- Feedback processado: **< 50ms**.
- Taxa de aceitação mínima: **≥ 25%** para manter modelo ativo.

💡 Fechamento da Camada

Os outputs da IA são **contratualizados por API, persistidos em banco híbrido e logados em JSON-line**, garantindo rastreabilidade e governança técnica.

🧠 CAMADA 14 — ALGORITMOS PREDITIVOS E LÓGICAS ADAPTATIVAS (100% TÉCNICA)

1) Objetivo

Especificar os **modelos preditivos e adaptativos** utilizados pela IA Operacional para antecipar comportamentos, prever estados futuros e ajustar outputs de forma dinâmica, mantendo a experiência fluida e escalável.

2) Fórmulas e Modelos

2.1 Predição de Estado Vibracional (Regressão Temporal)

$$vib_{t+1} = \alpha \cdot vib_t + \beta \cdot vib_{t-1} + \epsilon$$

$$vibt+1=α·vibt+β·vibt−1+ϵ$$

2.2 Predição de Ação (Logistic Regression)

$$P(click \mid x) = \sigma(\mathbf{w}^\top \mathbf{x}),\quad \sigma(z)=\frac{1}{1+e^{-z}}$$

$$P(click ∣ x)=σ(wTx),σ(z)=11+e−zP(click|x) = \sigma(\mathbf{w}^\top \mathbf{x}),\quad \sigma(z)=\frac{1}{1+e^{-z}}$$

$$P(click ∣ x)=σ(wTx),σ(z)=1+e−z1$$

- Features: tempo_sessao, compat_score, engajamento, risco.

2.3 Lógica de Oscilação

$$osc(u)=max(vibjanela)−min(vibjanela)media(vibjanela)osc(u) = \frac{\max(vib_{janela}) - \min(vib_{janela})}{media(vib_{janela})}$$

osc(u)=media(vibjanela)max(vibjanela)−min(vibjanela)

- Se `osc(u) > 0.5` → estado instável, ativar modo de proteção.

2.4 Predição de Abandono (Survival Analysis – Kaplan-Meier)

S(t)=∏ti≤t(1−dini)S(t) = \prod_{t_i \leq t} \left(1 - \frac{d_i}{n_i}\right)

S(t)=ti≤t∏(1−nidi)

- `S(t)`: probabilidade do usuário continuar até tempo t.

---

3) Algoritmos-Chave

Algoritmo	Função	Técnica
VibraFlowPredictor	Predição de curva vibracional	ARIMA/Regressão
AutoJourneyMapper	Ajuste automático de telas	Reinforcement Learning
MatchSentienceAI	Antecipar conexões prováveis	Classificação probabilística
OscillationDetector	Detectar instabilidade de comportamento	Variância + LSTM
AuraConvergenceEngine	Prever compatibilidade futura	Deep Pattern Matching

---

4) Pipeline Preditivo Assíncrono

Kafka Tópicos

- `predict.vibration` → inputs vibracionais.
- `predict.journey` → dados de abandono e fluxo.
- `predict.match` → interações sociais.

Consumidores

- `predictor-vibe` → gera previsão t+1 para cada usuário ativo.
- `journey-mapper` → simula fluxo alternativo de UX.
- `match-predictor` → calcula probabilidades de conexões futuras.

---

5) Pseudocódigo

```
def prever_estado(user_id):
    vib_hist = carregar_vibracao(user_id, janela=10)
    modelo = ARIMA(vib_hist, order=(2,1,0))
    previsao = modelo.forecast(steps=1)
    if previsao < 0.3:
        ativar_protecao(user_id)
    return previsao
```

```
def prever_abandono(sessao):
    features = extrair_features(sessao)
    prob = survival_model.predict(features)
    if prob < 0.4:
        sugerir_missao_reconexao(sessao.user_id)
```

```
    return prob
```

6) Estrutura de Banco

```
CREATE TABLE predictions (
  id BIGSERIAL PRIMARY KEY,
  user_id VARCHAR NOT NULL,
  tipo VARCHAR(30), -- vibe|journey|match
  valor NUMERIC(6,4),
  modelo VARCHAR(50),
  criado_em TIMESTAMP DEFAULT NOW()
);

CREATE INDEX ix_predictions_user_ts ON predictions(user_id, criado_em DESC);
```

Redis:

- `predict:vibe:{user_id}` → última previsão vibracional.
- `predict:journey:{sessao_id}` → probabilidade de abandono.

7) Logs Técnicos

```
{
  "type": "prediction",
  "user_id": "u_222",
  "modelo": "VibraFlowPredictor",
  "valor": 0.28,
  "status": "alerta",
  "trace_id": "73ff-92ab",
  "ts": "2025-09-02T18:46:13Z"
}
```

8) Métricas Observáveis

- `prediction_latency_ms{model}`
- `prediction_accuracy{model}`
- `prediction_drift_score`
- `prediction_alerts_triggered_total`

9) SLAs

- Predição vibracional: **< 200ms** por usuário.
- Ajuste de jornada: **< 150ms**.
- Drift check: execução **< 1s** batch/hora.

💡 Fechamento da Camada

Os algoritmos preditivos garantem que a IA **não apenas reaja**, mas **antecipe padrões e oscilações**, ajustando fluxos antes que o usuário perceba instabilidade.

🧠 CAMADA 15 — SISTEMA DE RISCO VIBRACIONAL E CORREÇÃO DE ANOMALIAS (100% TÉCNICA)

1) Objetivo

Monitorar em tempo real **oscilações vibracionais e comportamentais**, detectar **anomalias críticas** e aplicar **mecanismos automáticos de correção**, garantindo estabilidade e proteção do ecossistema.

---

2) Fórmulas de Score de Risco

2.1 Score de Risco Vibracional

scorerisco_vib(u)=(w1·incoerencia)+(w2·hesitacao)+(w3·oscilacao)−(w4·integridade_duc)score_{risco_vib}(u) = (w1 \cdot incoerencia) + (w2 \cdot hesitacao) + (w3 \cdot oscilacao) - (w4 \cdot integridade_duc)

scorerisco_vib(u)=(w1·incoerencia)+(w2·hesitacao)+(w3·oscilacao)−(w4·integridade_duc)

- `incoerencia`: divergência entre respostas (0–1).
- `hesitacao`: atraso médio de resposta em ms normalizado.
- `oscilacao`: variação relativa em janela temporal.
- `integridade_duc`: score documental/verificação.
- Pesos: w1=0.3, w2=0.2, w3=0.4, w4=0.1.
- **Threshold**: `≥ 0.8 → bloqueio`, `0.6–0.79 → desafio`, `<0.6 → normal`.

2.2 Score de Oscilação

osc(u)=max(vibjanela)−min(vibjanela)media(vibjanela)osc(u) = \frac{\max(vib_{janela}) - \min(vib_{janela})}{media(vib_{janela})}

osc(u)=media(vibjanela)max(vibjanela)−min(vibjanela)

2.3 Correção Automática

corrigir(u)= {"ritual_recalibracao",scorerisco≥0.8"ajuste_ux",0.6≤scorerisco<0.8"nenhum",scorerisco<0.6corrigir(u) = \begin{cases} "ritual_recalibracao", & score_{risco} \geq 0.8 \\ "ajuste_ux", & 0.6 \leq score_{risco} < 0.8 \\ "nenhum", & score_{risco} < 0.6 \end{cases}

corrigir(u)=⌊ ⌋ {

⌈"ritual_recalibracao","ajuste_ux","nenhum",scorerisco≥0.80.6≤scorerisco<0.8scorerisco<0.6

---

3) Mecanismos de Correção

Mecanismo	Quando Ativado	Ação Técnica
Ritual de Recalibração	Score ≥ 0.8	Gera missão via `/api/ia/missoes`
Isolamento Sensorial	Oscilação crítica (>0.6)	Feed entra em modo neutro
Reprogramação de Fluxo	Rejeições sucessivas	Jornada redirecionada
Espelhamento de Conteúdo	Padrão tóxico repetido	Conteúdo modificado p/ autorreflexão
Escudo Coletivo	Buraco vibracional (n>100)	Trigger de missão coletiva

4) APIs de Risco e Correção

4.1 POST /api/ia/risk/analyze

- **Body**:

```
{
  "user_id": "u_999",
  "features": {
    "incoerencia": 0.44,
    "hesitacao": 0.31,
    "oscilacao": 0.55,
    "integridade_duc": 0.72
  }
}
```

- **200 Response**:

```
{
  "score_risco": 0.81,
  "status": "blocked",
  "acao": "ritual_recalibracao",
  "trace_id": "f4b1-88cd",
  "ts": "2025-09-02T19:03:12Z"
}
```

4.2 POST /api/ia/risk/correct

- **Body**:

```
{
  "user_id": "u_999",
  "acao": "ritual_recalibracao"
}
```

- **200**:

```
{ "ack": true }
```

5) Estrutura de Banco

```
CREATE TABLE risk_events (
  id BIGSERIAL PRIMARY KEY,
  user_id VARCHAR NOT NULL,
  score NUMERIC(5,3),
  status VARCHAR(20),
  acao VARCHAR(30),
  features JSONB,
  ts TIMESTAMP DEFAULT NOW()
);

CREATE TABLE risk_collective (
  id BIGSERIAL PRIMARY KEY,
  grupo_id UUID NOT NULL,
  usuarios INT,
  score_medio NUMERIC(5,3),
  acao VARCHAR(30),
  ts TIMESTAMP DEFAULT NOW()
);
```

Redis:

- `risk:score:{user_id}` → cache score atual.
- `risk:collective:{grupo}` → score coletivo em tempo real.

---

6) Pseudocódigo

```
def analisar_risco(user_id, features):
    score = 0.3*features["incoerencia"] + 0.2*features["hesitacao"] \
          + 0.4*features["oscilacao"] - 0.1*features["integridade_duc"]

    if score >= 0.8:
        aplicar("ritual_recalibracao", user_id)
    elif score >= 0.6:
        aplicar("ajuste_ux", user_id)
    else:
        return "ok"

    log_event(user_id, score)
    return score
```

---

7) Logs Técnicos

```
{
  "type": "risk_event",
  "user_id": "u_999",
```

```
  "score": 0.81,
  "acao": "ritual_recalibracao",
  "features": { "incoerencia": 0.44, "hesitacao": 0.31, "oscilacao": 0.55 },
  "trace_id": "f4b1-88cd",
  "ts": "2025-09-02T19:05:41Z"
}
```

---

8) Métricas Observáveis

- `risk_score_avg`
- `risk_blocked_total`
- `risk_challenges_total`
- `risk_collective_events_total`

---

9) SLAs

- Análise de risco: **< 80ms**.
- Aplicação de correção: **< 150ms**.
- Detecção de buraco coletivo: agregação em **≤ 2s**.

💡 Fechamento da Camada

O sistema de risco vibracional permite **detecção quantitativa de incoerências e anomalias**, com **respostas automáticas configuráveis** e logs auditáveis para supervisão.

🧠 CAMADA 16 — LOGS TÉCNICOS, FEEDBACKS E RECALIBRAÇÃO DA IA (100% TÉCNICA)

1) Objetivo

Definir a infraestrutura de **logging técnico** e **feedback em tempo real** que alimentam os mecanismos de **recalibração automática** da IA Operacional. Essa camada garante **auditoria, rastreabilidade e ajustes contínuos** nos modelos.

---

2) Fórmulas de Recalibração

2.1 Atualização de Peso por Feedback

$$w_{novo} = w_{antigo} + \alpha \cdot (resposta\_user - previsao\_ia)$$

$$w_{novo} = w_{antigo} + \alpha \cdot (resposta\_user - previsao\_ia)$$

- α = taxa de aprendizado incremental (0.05–0.2).
- $resposta\_user \in \{0=negativo, 1=positivo\}$.
- $previsao\_ia$ = probabilidade atribuída pelo modelo.

2.2 Score de Impacto de Feedback

impacto=feedbacks_positivosfeedbacks_totaisimpacto = \frac{feedbacks_positivos}{feedbacks_totais}

impacto=feedbacks_totaisfeedbacks_positivos

2.3 Trigger de Recalibração

trigger={true,impacto<0.25∨drift>0.1false,caso contra´riotrigger =
\begin{cases}
true, & impacto < 0.25 \lor drift > 0.1 \\
false, & caso\ contrário
\end{cases}

trigger={true,false,impacto<0.25∨drift>0.1caso contra´rio

---

3) Estrutura de Logs

Logs Técnicos

Log	Conteúdo	Destino
`ia_decision.log`	decisão, score, contexto	PostgreSQL + S3
`feedback_result.log`	aceitação/rejeição	PostgreSQL
`loop_detect.log`	repetição de eventos	Redis + Elastic
`user_recalibration.log`	eventos de ajuste de pesos	PostgreSQL

Formato JSON-line

```
{
  "type": "decision",
  "user_id": "u_123",
  "context": "feed",
  "score": 0.77,
  "output": "sugestao_conexao",
  "feedback": "reject",
  "recalibrado": true,
  "ts": "2025-09-02T19:25:43Z"
}
```

---

4) Estrutura de Banco

```
CREATE TABLE ia_feedback (
  id BIGSERIAL PRIMARY KEY,
  user_id VARCHAR NOT NULL,
  output_id BIGINT,
  acao VARCHAR(20), -- accept|reject|ignore
  score NUMERIC(6,4),
  contexto VARCHAR(30),
  ts TIMESTAMP DEFAULT NOW()
);

CREATE TABLE ia_recalibration (
```

```
  id BIGSERIAL PRIMARY KEY,
  user_id VARCHAR NOT NULL,
  modelo VARCHAR(50),
  peso_antigo NUMERIC(6,4),
  peso_novo NUMERIC(6,4),
  trigger BOOLEAN,
  ts TIMESTAMP DEFAULT NOW()
);
```

Redis:

- `feedback:pending:{user_id}` → fila de feedbacks não processados.
- `recalib:score:{modelo}` → último score agregado.

---

5) APIs

5.1 POST /api/ia/feedback

- **Body**:

```
{
  "user_id": "u_123",
  "output_id": "out_222",
  "acao": "reject",
  "context": "feed",
  "ts": "2025-09-02T19:27:13Z"
}
```

- **200 Response**:

```
{ "ack": true, "impacto": 0.21, "trigger_recalibracao": true }
```

5.2 GET /api/ia/recalibration/status?modelo=rec_model

- **200 Response**:

```
{
  "modelo": "rec_model",
  "impacto": 0.23,
  "drift": 0.12,
  "ultima_recalibracao": "2025-09-01T21:44:08Z",
  "status": "em_execucao"
}
```

---

6) Pseudocódigo

```
def processar_feedback(user_id, output_id, acao, score):
    impacto = calcular_impacto(output_id)
    if impacto < 0.25:
        trigger_recalibracao(modelo="rec_model")
    atualizar_logs(user_id, output_id, acao, score, impacto)
```

---

7) Métricas Observáveis

- `feedback_accept_ratio`
- `feedback_latency_ms`
- `recalibration_triggers_total`
- `model_weight_updates_total`

---

8) SLAs

- Processamento de feedback: **< 50ms**.
- Trigger de recalibração: **< 200ms** após evento.
- Taxa mínima de aceitação: **≥ 25%**.

💡 Fechamento da Camada

A camada de **logs, feedbacks e recalibração** garante que a IA aprenda continuamente, **ajustando pesos matematicamente** e mantendo a rastreabilidade de todas as decisões.

🧠 CAMADA 17 — PAINEL DE COMANDO MULTINÚCLEO DA IA (100% TÉCNICA)

1) Objetivo

Descrever o **painel central de orquestração da IA Operacional**, responsável por monitorar, sincronizar e controlar os **múltiplos núcleos especializados** (Recomendação, Jornada, Segurança, Performance, etc.), com permissões claras e logs intermodulares.

---

2) Estrutura de Orquestração

- **Painel Central (Orchestrator)** → microserviço centralizado que recebe métricas e outputs de todos os núcleos.
- Comunicação:
  - **Kafka** → ingestão de eventos assíncronos.
  - **gRPC interno** → chamadas síncronas críticas (ex: segurança).
  - **Redis Pub/Sub** → sincronização em tempo real de métricas.

---

3) Módulos Integrados

Núcleo	Dados Recebidos	Ação Emitida
IA Recomendação	Vetores de comportamento, feedback	Top-K outputs de sugestão

Núcleo	Dados Recebidos	Ação Emitida
IA Jornada	Eventos de fricção e abandono	Ajustes de UX (menus, telas)
IA Segurança	Logs de risco, login, IPs	Bloqueio, challenge, blacklist
IA Performance	Métricas p95, uso de CPU, erro	Escalar pods, redistribuir carga
IA Sensorial	Conteúdos e posts	Score vibracional e alcance
IA Missões	Score de risco, vibração, engajamento	Missões personalizadas e coletivas
IA Moderadora	Linguagem e interações sociais	Flags, silenciamento, moderação assistida

4) Fluxograma Central

```
graph TD
A[Eventos captados] --> B[Kafka Topics]
B --> C[Orchestrator Service]
C --> D1[IA Recomendação]
C --> D2[IA Jornada]
C --> D3[IA Segurança]
C --> D4[IA Performance]
C --> D5[IA Sensorial]
C --> D6[IA Missões]
D1 --> E[Outputs Top-K]
D2 --> E
D3 --> E
D4 --> E
D5 --> E
D6 --> E
E --> F[Redis Cache / API Delivery]
```

5) Estrutura de Banco

```
CREATE TABLE orchestrator_logs (
  id BIGSERIAL PRIMARY KEY,
  modulo VARCHAR(30) NOT NULL,
  evento VARCHAR(50) NOT NULL,
  input JSONB,
  output JSONB,
  trace_id UUID NOT NULL,
  ts TIMESTAMP DEFAULT NOW()
);

CREATE TABLE orchestrator_metrics (
  id BIGSERIAL PRIMARY KEY,
  modulo VARCHAR(30),
  latencia_ms NUMERIC(6,2),
  throughput INT,
  status VARCHAR(20),
  ts TIMESTAMP DEFAULT NOW()
```

```
);
```

Redis:

- `orch:status:{modulo}` → último status por núcleo.
- `orch:metrics` → agregados de performance.

---

6) APIs do Painel

6.1 GET /api/ia/orch/status

```
{
 "modulos": [
  {"nome": "recomendacao", "status": "ok", "latencia_ms": 72},
  {"nome": "seguranca", "status": "alerta", "latencia_ms": 44}
 ],
 "ts": "2025-09-02T19:44:09Z"
}
```

6.2 GET /api/ia/orch/logs?modulo=seguranca

```
[
 {
  "modulo": "seguranca",
  "evento": "login_anomaly",
  "output": {"acao": "blocked"},
  "trace_id": "ab88-7121",
  "ts": "2025-09-02T19:42:11Z"
 }
]
```

---

7) Pseudocódigo

```
def orquestrar_evento(evento):
    modulo = identificar_modulo(evento)
    resultado = chamar_modulo(modulo, evento)
    salvar_log(modulo, evento, resultado)
    atualizar_metricas(modulo, resultado)
    return resultado
```

---

8) Logs Intermodulares

```
{
 "type": "orchestrator_event",
 "modulo": "jornada",
```

```
  "evento": "abandono_detectado",
  "input": {"sessao_id": "s_882"},
  "output": {"ajuste": "tela_inicial"},
  "latency_ms": 138,
  "trace_id": "d199-12ff",
  "ts": "2025-09-02T19:46:28Z"
}
```

---

9) Métricas Observáveis

- `orch_event_latency_ms{modulo}`
- `orch_event_throughput_total`
- `orch_failures_total{modulo}`
- `orch_status{modulo}`

---

10) SLAs

- Latência média de orquestração: **< 100ms**.
- Throughput mínimo: **10k eventos/s por região**.
- Failover automático entre clusters em **≤ 2s**.

💡 Fechamento da Camada

O painel multinúcleo garante que a IA funcione como **um sistema distribuído, mas coordenado**, com logs, métricas e auditoria de ponta a ponta.

🧠 CAMADA 18 — PSEUDOCÓDIGOS OPERACIONAIS DE DECISÃO ENERGÉTICA (100% TÉCNICA)

1) Objetivo

Fornecer **pseudocódigos formais** que descrevem como a IA Operacional processa inputs, aplica fórmulas e toma decisões em tempo real. Estes códigos servem como **referência direta para implementação**, conectando módulos distintos.

---

2) Pseudocódigo — Compatibilidade de Usuários

```
def calcular_compatibilidade(user_a, user_b):
    vec_a = carregar_vetor(user_a)
    vec_b = carregar_vetor(user_b)

    sim_cos = cosine_similarity(vec_a, vec_b)
    hist = historico_interacoes(user_a, user_b)
    contexto = calcular_contexto(user_a, user_b)

    score = 0.5*sim_cos + 0.3*hist + 0.2*contexto
```

```
    if score >= 0.75:
        return {"categoria": "alta", "score": score}
    elif score >= 0.5:
        return {"categoria": "media", "score": score}
    else:
        return {"categoria": "baixa", "score": score}
```

---

3) Pseudocódigo — Ajuste de Jornada

```
def avaliar_jornada(sessao):
    abandono = calcular_abandono(sessao)
    hesitacao = tempo_resposta_medio(sessao)
    erros = contar_erros_fluxo(sessao)

    score_ajuste = 0.4*abandono + 0.3*hesitacao + 0.3*erros

    if score_ajuste >= 0.6:
        aplicar_ajuste(sessao.user_id)
        registrar_log("ajuste_aplicado", sessao.user_id, score_ajuste)
```

---

4) Pseudocódigo — Análise de Risco

```
def analisar_risco(user_id, features):
    score = 0.3*features["incoerencia"] \
          + 0.2*features["hesitacao"] \
          + 0.4*features["oscilacao"] \
          - 0.1*features["integridade_duc"]

    if score >= 0.8:
        bloquear_usuario(user_id)
        log_event("blocked", user_id, score)
    elif score >= 0.6:
        aplicar_challenge(user_id)
        log_event("challenge", user_id, score)
    else:
        permitir(user_id)
```

---

5) Pseudocódigo — Recalibração com Feedback

```
def processar_feedback(user_id, output_id, acao, score_previsto):
    impacto = calcular_impacto(output_id)
    diferenca = (1 if acao=="accept" else 0) - score_previsto

    if impacto < 0.25 or abs(diferenca) > 0.5:
```

```
        trigger_recalibracao("rec_model")

    atualizar_peso("rec_model", diferenca)
    salvar_feedback(user_id, output_id, acao, impacto)
```

6) Pseudocódigo — Orquestração Multinúcleo

```
def orquestrar_evento(evento):
    modulo = roteador_modulo(evento.tipo)
    resultado = chamar_modulo(modulo, evento)

    atualizar_metricas(modulo, resultado)
    salvar_log(modulo, evento, resultado)

    enviar_feedback_loop(evento, resultado)
    return resultado
```

7) Logs Técnicos — Saída de Pseudocódigos

Formato JSON-line:

```
{
  "type": "pseudocode_exec",
  "modulo": "seguranca",
  "acao": "blocked",
  "user_id": "u_888",
  "score": 0.82,
  "trace_id": "fa33-9121",
  "latency_ms": 47,
  "ts": "2025-09-02T19:58:22Z"
}
```

8) Métricas Observáveis

- `pseudocode_exec_latency_ms{modulo}`
- `pseudocode_exec_total{acao}`
- `pseudocode_exec_errors_total`

9) SLAs

- Execução de pseudocódigo crítico: **< 50ms**.
- Execução de pseudocódigo adaptativo: **< 150ms**.
- Taxa de falhas de execução: **< 1%**.

💡 Fechamento da Camada

Os pseudocódigos consolidam a lógica da IA em **estruturas técnicas claras**, permitindo implementação direta e validação de decisões em tempo real.

🧠 CAMADA 19 — ESTRUTURA DE APIs E ENDPOINTS DA IA OPERACIONAL (100% TÉCNICA)

1) Objetivo

Formalizar os **contratos de API** da IA Operacional, com endpoints, parâmetros, payloads de entrada/saída, códigos de erro e políticas de segurança. Essas APIs são a **interface oficial** entre a IA e os demais módulos do FriendApp.

---

2) Padrões de API

- **REST + JSON**
- **Autenticação**: JWT (expiração curta, 15min).
- **Headers obrigatórios**:
  - `Authorization: Bearer <token>`
  - `X-Trace-Id: <uuid>`
  - `Content-Type: application/json`
- **Timeout**: 300ms (interno), 1s (externo).
- **Rate limit**: 60 req/min por usuário.

---

3) Principais Endpoints

3.1 Core

- `GET /api/ia/core/status` → status geral da IA.
- **200 Response**:

```
{
  "uptime": "99.97",
  "latency_ms": 182,
  "throughput": 5002,
  "ts": "2025-09-02T20:11:19Z"
}
```

3.2 Recomendação

- `GET /api/ia/recomendacao?user_id=u123&k=10&context=feed`
- **200 Response**:

```
{
  "user_id": "u123",
  "context": "feed",
```

```
  "items": [
    { "id": "f_882", "score": 0.872, "trace_id": "a88c-99fa" }
  ]
}
```

- `POST /api/ia/feedback-sugestao`
- **Body**:

```
{ "user_id": "u123", "item_id": "f_882", "action": "accept" }
```

3.3 Segurança

- `POST /api/ia/security/analyze`
- **Body**:

```
{
  "user_id": "u123",
  "ip": "201.22.11.4",
  "actions": ["login","feed_open"]
}
```

- **200 Response**:

```
{
  "score_risco": 0.81,
  "status": "blocked",
  "acao": "challenge"
}
```

3.4 Jornada

- `GET /api/ia/jornada/analisar?user_id=u123&sessao=s88`
- **200 Response**:

```
{
  "score_friccao": 0.74,
  "ajustes": ["reordenar_menu","tela_inicial"]
}
```

- `POST /api/ia/jornada/aplicar`
- **Body**:

```
{
  "user_id": "u123",
  "ajustes": [{ "tipo": "reordenar_menu", "novo_layout": ["mapa","feed","chat"] }]
}
```

3.5 Risco Vibracional

- `POST /api/ia/risk/analyze`
- **Body**:

```
{
  "user_id": "u456",
  "features": { "incoerencia": 0.33, "hesitacao": 0.21, "oscilacao": 0.55 }
}
```

- **200 Response**:

```
{ "score_risco": 0.72, "status": "challenge", "acao": "ajuste_ux" }
```

---

4) Estrutura de Erros

Código	Descrição	Ação
400	Parâmetro inválido	Corrigir request
401	Token inválido/expirado	Reautenticação
403	Acesso negado (sem permissão)	Bloqueio
404	Dados não encontrados	Fallback
409	Conflito de estado	Retry
429	Rate limit excedido	Backoff exponencial
500	Erro interno	Alertar DevOps

---

5) Estrutura de Banco para Tracing

```
CREATE TABLE api_traces (
  id BIGSERIAL PRIMARY KEY,
  endpoint VARCHAR(100),
  user_id VARCHAR,
  trace_id UUID,
  status_code INT,
  latency_ms NUMERIC(6,2),
  payload JSONB,
  created_at TIMESTAMP DEFAULT NOW()
);
```

Redis:

- `api:rate:{user_id}` → contagem para rate limit.

---

6) Logs Técnicos

```
{
  "endpoint": "/api/ia/recomendacao",
  "user_id": "u123",
  "status_code": 200,
  "latency_ms": 91,
  "trace_id": "a88c-99fa",
  "payload": { "k": 10, "context": "feed" },
  "ts": "2025-09-02T20:17:44Z"
}
```

---

7) Métricas Observáveis

- `api_request_latency_ms{endpoint}`
- `api_error_rate{endpoint}`
- `api_requests_total{endpoint}`
- `api_rate_limit_exceeded_total`

---

8) SLAs

- Latência p95 por endpoint: **< 200ms**.
- Disponibilidade ≥ **99.95%**.
- Erro rate ≤ **0.5%**.

💡 Fechamento da Camada

As APIs da IA Operacional estão **formalizadas em contratos técnicos completos**, com payloads, erros, logs e SLAs definidos para execução real.

🧠 CAMADA 20 — BANCO DE DADOS DA IA OPERACIONAL (DDL, ÍNDICES, PARTIÇÕES, CACHE) — **100% TÉCNICA**

1) Objetivo

Projetar a camada de persistência e acesso com **PostgreSQL (OLTP/analítico leve)**, **Redis (cache/filas rápidas)**, **Firestore (estado efêmero)** e **ElasticSearch (observabilidade/logs)**, com **CDC→Kafka**, particionamento temporal e políticas de retenção.

---

2) Topologia de Dados (visão rápida)

- **PostgreSQL (`ia_ops`)**: fonte de verdade; particionado por tempo; extensões `pgvector`, `uuid-ossp`, `pgcrypto`.

- **Redis**: ZSETs para Top-K, chaves TTL para sessões/medidas; Lua para operações atômicas.
- **Firestore**: coleções efêmeras de sessão/UX (TTL curto).
- **ElasticSearch**: índices de logs técnicos e tracing.
- **S3/Cold storage**: arquivos parquet para histórico longo (ETL batch).
- **CDC**: Debezium → Kafka (`db.ia_ops.*`) para stream near-real-time.

---

3) Inicialização PostgreSQL (schema + extensões)

```
CREATE SCHEMA IF NOT EXISTS ia_ops;
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";
CREATE EXTENSION IF NOT EXISTS pgcrypto;
CREATE EXTENSION IF NOT EXISTS vector;        -- pgvector

-- Roles mínimas
CREATE ROLE ia_admin LOGIN;
CREATE ROLE ia_app   LOGIN;
CREATE ROLE ia_ro;

GRANT USAGE ON SCHEMA ia_ops TO ia_app, ia_ro;
```

---

4) Tabelas-Chave (particionadas por mês, `created_at`)

> Padrão: tabela “pai” + partições FOR VALUES FROM ('YYYY-MM-01') TO ('YYYY-MM-01').
>
> Índices por `user_id`, `created_at DESC`, e GIN/IVFFLAT quando aplicável.

4.1 Embeddings de Usuário/Item (pgvector)

```
CREATE TABLE ia_ops.user_embedding (
  user_id VARCHAR PRIMARY KEY,
  dim SMALLINT NOT NULL CHECK (dim IN (64,128,256)),
  vec vector(128) NOT NULL,
  updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

-- Vetores de features/conteúdos
CREATE TABLE ia_ops.feature_embedding (
  feature_id VARCHAR PRIMARY KEY,
  type VARCHAR(24) NOT NULL,   -- user|content|feature
  vec vector(128) NOT NULL,
  updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

-- Índices ANN (IVFFLAT) — requer REINDEX após carga inicial
CREATE INDEX ON ia_ops.user_embedding USING ivfflat (vec vector_cosine_ops) WITH (lists = 10
0);
CREATE INDEX ON ia_ops.feature_embedding USING ivfflat (vec vector_cosine_ops) WITH (lists = 1
```

```
00);
```

4.2 Interações (evento granular)

```
CREATE TABLE ia_ops.interaction_log (
  id BIGSERIAL,
  user_id VARCHAR NOT NULL,
  item_id VARCHAR NOT NULL,
  action VARCHAR(16) NOT NULL,       -- view|click|dismiss|block
  context VARCHAR(16) NOT NULL,      -- feed|connect|feature
  meta JSONB,
  created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
  PRIMARY KEY (id, created_at)
) PARTITION BY RANGE (created_at);

-- partição exemplo mês corrente
CREATE TABLE ia_ops.interaction_log_2025_09
  PARTITION OF ia_ops.interaction_log FOR VALUES
  FROM ('2025-09-01') TO ('2025-10-01');

CREATE INDEX ix_interaction_user_ts ON ia_ops.interaction_log_2025_09 (user_id, created_at DES
C);
CREATE INDEX ix_interaction_item_ts ON ia_ops.interaction_log_2025_09 (item_id, created_at DES
C);
CREATE INDEX ix_interaction_meta_gin ON ia_ops.interaction_log_2025_09 USING GIN (meta jsonb
_path_ops);
```

4.3 Sugestões Servidas (serving log)

```
CREATE TABLE ia_ops.suggestion_log (
  id BIGSERIAL,
  user_id VARCHAR NOT NULL,
  item_id VARCHAR NOT NULL,
  score NUMERIC(6,4) NOT NULL,
  features JSONB NOT NULL,
  context VARCHAR(16) NOT NULL,
  trace_id UUID NOT NULL DEFAULT uuid_generate_v4(),
  created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
  PRIMARY KEY (id, created_at)
) PARTITION BY RANGE (created_at);

-- índices típicos
CREATE INDEX ix_sugg_user_ts ON ia_ops.suggestion_log_2025_09 (user_id, created_at DESC);
CREATE INDEX ix_sugg_ctx_ts  ON ia_ops.suggestion_log_2025_09 (context, created_at DESC);
CREATE INDEX ix_sugg_feat_gin ON ia_ops.suggestion_log_2025_09 USING GIN (features jsonb_pa
th_ops);
```

4.4 Segurança (eventos e decisões)

```
CREATE TABLE ia_ops.security_events (
  id BIGSERIAL,
  user_id VARCHAR,
  ip INET,
  score_risco NUMERIC(5,3),
  status VARCHAR(20),              -- allowed|challenge|blocked
  reasons JSONB,
  created_at TIMESTAMPTZ DEFAULT NOW(),
  PRIMARY KEY (id, created_at)
) PARTITION BY RANGE (created_at);

CREATE INDEX ix_sec_user_ts ON ia_ops.security_events_2025_09 (user_id, created_at DESC);
CREATE INDEX ix_sec_status_ts ON ia_ops.security_events_2025_09 (status, created_at DESC);
```

4.5 Jornada/UX

```
CREATE TABLE ia_ops.journey_sessions (
  sessao_id VARCHAR PRIMARY KEY,
  user_id VARCHAR NOT NULL,
  started_at TIMESTAMPTZ NOT NULL,
  ended_at   TIMESTAMPTZ,
  abandono   BOOLEAN DEFAULT FALSE
);

CREATE TABLE ia_ops.journey_events (
  id BIGSERIAL PRIMARY KEY,
  sessao_id VARCHAR NOT NULL REFERENCES ia_ops.journey_sessions(sessao_id),
  event VARCHAR(50) NOT NULL,
  value JSONB,
  created_at TIMESTAMPTZ DEFAULT NOW()
);

CREATE TABLE ia_ops.journey_adjustments (
  id BIGSERIAL PRIMARY KEY,
  user_id VARCHAR NOT NULL,
  tipo VARCHAR(50) NOT NULL,
  detail JSONB,
  applied_at TIMESTAMPTZ DEFAULT NOW()
);
```

4.6 Outputs e Feedback

```
CREATE TABLE ia_ops.ia_outputs (
  id BIGSERIAL PRIMARY KEY,
  user_id VARCHAR NOT NULL,
  tipo VARCHAR(30) NOT NULL,       -- sugestao|correcao|missao|pausa|alerta
```

```
  payload JSONB,
  score NUMERIC(6,4),
  context VARCHAR(30),
  trace_id UUID NOT NULL DEFAULT uuid_generate_v4(),
  created_at TIMESTAMPTZ DEFAULT NOW()
) PARTITION BY RANGE (created_at);

CREATE TABLE ia_ops.ia_output_feedback (
  id BIGSERIAL PRIMARY KEY,
  output_id BIGINT NOT NULL,
  user_id VARCHAR NOT NULL,
  action VARCHAR(20),              -- accept|reject|ignore
  created_at TIMESTAMPTZ DEFAULT NOW(),
  FOREIGN KEY (output_id) REFERENCES ia_ops.ia_outputs(id)
);

CREATE INDEX ix_out_user_ts ON ia_ops.ia_outputs_2025_09 (user_id, created_at DESC);
CREATE INDEX ix_outfb_user_ts ON ia_ops.ia_output_feedback (user_id, created_at DESC);
```

4.7 Performance/Tracing

```
CREATE TABLE ia_ops.perf_metrics (
  id BIGSERIAL PRIMARY KEY,
  service VARCHAR(48) NOT NULL,
  cpu NUMERIC(5,2),
  mem NUMERIC(5,2),
  latency_p95 NUMERIC(6,2),
  error_rate NUMERIC(5,4),
  created_at TIMESTAMPTZ DEFAULT NOW()
);

CREATE TABLE ia_ops.api_traces (
  id BIGSERIAL PRIMARY KEY,
  endpoint VARCHAR(120),
  user_id VARCHAR,
  trace_id UUID,
  status_code INT,
  latency_ms NUMERIC(6,2),
  payload JSONB,
  created_at TIMESTAMPTZ DEFAULT NOW()
);
CREATE INDEX ix_api_endpoint_ts ON ia_ops.api_traces (endpoint, created_at DESC);
CREATE INDEX ix_api_trace_id     ON ia_ops.api_traces (trace_id);
```

4.8 Predições/Modelos

```
CREATE TABLE ia_ops.predictions (
  id BIGSERIAL PRIMARY KEY,
```

```
  user_id VARCHAR NOT NULL,
  tipo VARCHAR(30),          -- vibe|journey|match
  valor NUMERIC(6,4),
  model VARCHAR(50),
  created_at TIMESTAMPTZ DEFAULT NOW()
);
CREATE INDEX ix_pred_user_ts ON ia_ops.predictions (user_id, created_at DESC);

CREATE TABLE ia_ops.model_registry (
  id SERIAL PRIMARY KEY,
  nome VARCHAR(50) NOT NULL,
  versao VARCHAR(20) NOT NULL,
  status VARCHAR(20) DEFAULT 'candidate', -- candidate|approved|prod
  accuracy NUMERIC(5,3),
  drift_score NUMERIC(5,3),
  approved_by VARCHAR(50),
  created_at TIMESTAMPTZ DEFAULT NOW()
);
```

---

5) Políticas de Partição & Retenção

- **Granularidade**: mensal por padrão; diário para `api_traces` / `perf_metrics` em ambientes de alto volume.
- **Retenção**:
  - `api_traces`, `perf_metrics`: 90 dias (rotação + `DROP PARTITION`).
  - `interaction_log`, `suggestion_log`, `security_events`: 12 meses on-line → arquiva em S3 (Parquet).
  - **GDPR/Delete**: função `erase_user(user_id)` que apaga/anonimiza em todas as tabelas (FK on delete cascade + pseudonimização).

---

6) Segurança, RLS e PII

- **Coluna PII** só em tabelas dedicadas (ex.: `user_profile_pii`) com `RLS` habilitado; demais usam `user_id` pseudonimizado.
- Ativar **RLS** onde necessário:

```
ALTER TABLE ia_ops.api_traces ENABLE ROW LEVEL SECURITY;
CREATE POLICY ro_by_service ON ia_ops.api_traces
  FOR SELECT TO ia_ro USING (current_setting('app.service') = service);
```

- Criptografia de campos sensíveis com `pgcrypto` (ex.: IP, device_fingerprint).
- **At-Rest**: discos criptografados (cloud KMS). **In-Transit**: TLS 1.2+.

---

7) CDC → Kafka (Debezium)

- Conectores: `dbserver1.ia_ops.interaction_log`, `...suggestion_log`, `...security_events`, `...api_traces`.
- Tópicos: `db.ia_ops.interaction_log`, etc.

- **Chaves**: `user_id` para compactação por usuário onde aplicável.
- Consumidores: `feature-builder`, `real-time-analytics`, `anomaly-detector`.

---

8) Redis (chaves, TTL e scripts)

- **Padrões de chave**
  - `rec:topk:{user_id}:{context}` → ZSET (score), TTL 5–15 min.
  - `stat:ctr:{item_id}` → HASH contagens decaídas.
  - `journey:score:{user_id}` → SF corrente, TTL 30 min.
  - `risk:score:{user_id}` → TTL 15 min.
- **Lua (atomizar update+expire)**

```
-- KEYS[1]=key, ARGV[1]=score, ARGV[2]=item_id, ARGV[3]=ttl
redis.call('ZADD', KEYS[1], ARGV[1], ARGV[2])
redis.call('EXPIRE', KEYS[1], ARGV[3])
return 1
```

- **Eviction**: `allkeys-lru`, alertas quando hit-ratio < 85%.

---

9) Firestore (efêmero)

- Coleções:
  - `/sessions/{sessaoId}`: estado UX em tempo real (TTL 24–48h).
  - `/ux_adjust/{userId}`: fila de ajustes aplicados (TTL 7d).
- **Rules**: somente serviços internos (`ia_app`) com service account.
- **Mirror** de sessão em PostgreSQL via job periódico para análises.

---

10) ElasticSearch (observabilidade)

- Índices: `logs-ia-YYYY.MM.DD`, ILM com hot-warm-delete (30/60/90d).
- Mappings principais: `@timestamp`, `trace_id`, `service`, `latency_ms`, `level`, `payload` (flatten).
- Dashboards: latência por endpoint, taxa de erro, correlação `trace_id` ↔ `api_traces`.

---

11) Conexão, Pool e DR

- **PgBouncer**: `transaction` pooling; `max_client_conn` conforme QPS.
- **Replicação**: 1 réplica por região + 1 cross-region (async).
- **Backups**: PITR (WAL archiving) + snapshots diários; testes de restore semanais.
- **Failover**: DNS/Proxy + promoção de réplica < 2 min.

---

12) Exemplos de Consultas Otimizadas

```
-- Últimas 50 sugestões servidas ao usuário
SELECT item_id, score, created_at
```

```
FROM ia_ops.suggestion_log
PARTITION FOR (DATE_TRUNC('month', NOW()))  -- opcional, em PG14+
WHERE user_id = 'u_123'
ORDER BY created_at DESC
LIMIT 50;

-- CTR aproximado por item (7 dias)
SELECT item_id,
  SUM(CASE WHEN action='click' THEN 1 ELSE 0 END)::float /
  NULLIF(SUM(CASE WHEN action='view' THEN 1 ELSE 0 END),0) AS ctr
FROM ia_ops.interaction_log
WHERE created_at >= NOW() - INTERVAL '7 days'
GROUP BY item_id
ORDER BY ctr DESC
LIMIT 100;
```

---

13) Migrações & Versionamento

- Ferramenta: **Flyway** ou **Liquibase**.
- Padrão: `V{yyyyMMddHHmm}__descricao.sql`.
- Regras:
  - **Nunca** alterar coluna sem etapa de “backfill+shadow column+switch”.
  - Índices IVFFLAT requerem `REINDEX CONCURRENTLY` após carregamento.

---

14) Métricas & SLAs de Dados

- **p95 SELECT** em OLTP: < **120ms**.
- **CDC lag** (Debezium): < **2s**.
- **Hit-ratio Redis**: ≥ **85%**.
- **Integridade de partições**: checagem diária; `VACUUM` / `ANALYZE` noturnos.

---

15) Policies de Privacidade/Exclusão

- Função `ia_ops.erase_user('u_123')` varre tabelas particionadas;
- Substitui `user_id` por hash irreversível onde retenção estatística é necessária;
- Mantém **tracing sem PII** para auditoria.

✅ Resultado

Camada de dados **executável e auditável**: DDL completo com particionamento, índices, cache Redis, EF-states no Firestore, observabilidade no Elastic, CDC para Kafka e políticas de segurança/retention/DR prontas para operação.

🧠 CAMADA 21 — ARQUITETURA ESCALONÁVEL COM CONTAINERS E KUBERNETES (100% TÉCNICA)

1) Objetivo

Padronizar **deploy, segurança, observabilidade e autoscaling** dos módulos da IA Operacional (recomendação, jornada, segurança, performance, etc.) em **Kubernetes** com **Istio (mTLS)**, **HPA v2**, **KEDA (Kafka lag)**, **PDB**, **NetworkPolicy**, **RBAC** e **quotas de namespace**.

---

2) Padrões por Módulo (ex.: `ia-recomendacao`)

- **Deployment**: rolling update, probes, requests/limits, anti-affinity, spread por zona.
- **Service**: `ClusterIP` porta 8080 gRPC/HTTP.
- **Ingress**: Istio `VirtualService` + `DestinationRule` (mTLS, retries, circuit breaker).
- **Autoscaling**: HPA (CPU/Mem) + KEDA (Kafka consumer lag).
- **Resiliência**: `PodDisruptionBudget`, `maxUnavailable`.
- **Segurança**: `runAsNonRoot`, `readOnlyRootFilesystem`, `NET_RAW` drop, secrets via `envFrom`.
- **Rede**: `NetworkPolicy` egress only DB/Kafka/Redis/Elastic.
- **Observabilidade**: Prometheus scrape, OpenTelemetry tracing, logs STDOUT.

---

3) Manifests (mínimos executáveis)

3.1 Namespace, Quotas e Limites

```
apiVersion: v1
kind: Namespace
metadata: { name: ia-ops, labels: { istio-injection: enabled } }
---
apiVersion: v1
kind: ResourceQuota
metadata: { name: rq-ia-ops, namespace: ia-ops }
spec:
  hard: { requests.cpu: "200", limits.cpu: "400", requests.memory: 400Gi, limits.memory: 800Gi, po
ds: "800" }
---
apiVersion: v1
kind: LimitRange
metadata: { name: lr-ia-ops, namespace: ia-ops }
spec:
  limits:
  - type: Container
    default: { cpu: "500m", memory: "1Gi" }
    defaultRequest: { cpu: "250m", memory: "512Mi" }
```

3.2 ServiceAccount, RBAC

```
apiVersion: v1
kind: ServiceAccount
metadata: { name: ia-recomendacao-sa, namespace: ia-ops }
---
apiVersion: rbac.authorization.k8s.io/v1
```

```
kind: Role
metadata: { name: ia-read-config, namespace: ia-ops }
rules:
- apiGroups: [""]
  resources: ["configmaps","secrets"]
  verbs: ["get","list","watch"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata: { name: ia-recomendacao-rb, namespace: ia-ops }
subjects: [{ kind: ServiceAccount, name: ia-recomendacao-sa, namespace: ia-ops }]
roleRef: { kind: Role, name: ia-read-config, apiGroup: rbac.authorization.k8s.io }
```

3.3 ConfigMap & Secret

```
apiVersion: v1
kind: ConfigMap
metadata: { name: ia-recomendacao-cm, namespace: ia-ops }
data:
  KAFKA_BROKERS: "kafka-0:9092,kafka-1:9092"
  KAFKA_TOPICS: "ux.view,ux.click,ux.dwell,feed.new_post"
  REDIS_ADDR: "redis.ia-ops.svc:6379"
  PG_DSN: "postgres://ia_app@pg.ia-ops.svc:5432/ia_ops"
  ANN_INDEX: "ivfflat"
---
apiVersion: v1
kind: Secret
metadata: { name: ia-recomendacao-sec, namespace: ia-ops }
type: Opaque
stringData:
  PG_PASSWORD: "****"
  JWT_PUBLIC_KEY: "-----BEGIN PUBLIC KEY-----\n..."
```

3.4 Deployment + Service (istio sidecar injetado)

```
apiVersion: apps/v1
kind: Deployment
metadata: { name: ia-recomendacao, namespace: ia-ops }
spec:
  replicas: 3
  strategy: { type: RollingUpdate, rollingUpdate: { maxUnavailable: 1, maxSurge: 1 } }
  selector: { matchLabels: { app: ia-recomendacao } }
  template:
    metadata:
      labels: { app: ia-recomendacao, tier: backend, module: rec }
      annotations:
        prometheus.io/scrape: "true"
        prometheus.io/port: "8080"
```

```
        prometheus.io/path: "/metrics"
        sidecar.istio.io/inject: "true"
    spec:
      serviceAccountName: ia-recomendacao-sa
      priorityClassName: high-priority
      securityContext: { fsGroup: 2000 }
      containers:
      - name: app
        image: friendapp/ia-recomendacao:1.14.3
        imagePullPolicy: IfNotPresent
        ports: [{ containerPort: 8080, name: http }]
        envFrom:
        - configMapRef: { name: ia-recomendacao-cm }
        - secretRef: { name: ia-recomendacao-sec }
        resources:
          requests: { cpu: "500m", memory: "1Gi" }
          limits:   { cpu: "2",    memory: "4Gi" }
        securityContext:
          runAsUser: 1001
          runAsNonRoot: true
          readOnlyRootFilesystem: true
          allowPrivilegeEscalation: false
          capabilities: { drop: ["ALL"] }
        readinessProbe: { httpGet: { path: "/healthz", port: 8080 }, periodSeconds: 5, timeoutSeconds:
2, failureThreshold: 6 }
        livenessProbe:  { httpGet: { path: "/livez",  port: 8080 }, periodSeconds: 10, timeoutSeconds: 2,
failureThreshold: 3 }
        startupProbe:   { httpGet: { path: "/startupz", port: 8080 }, failureThreshold: 30, periodSecond
s: 3 }
      nodeSelector: { workload: "compute-std" }
      tolerations:
      - key: "preemptible"  # opcional
        operator: "Exists"
        effect: "NoSchedule"
      affinity:
        podAntiAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
          - weight: 100
            podAffinityTerm:
              topologyKey: "kubernetes.io/hostname"
              labelSelector: { matchLabels: { app: ia-recomendacao } }
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: topology.kubernetes.io/zone
                operator: In
                values: ["zone-a","zone-b","zone-c"]
      topologySpreadConstraints:
      - maxSkew: 1
```

```
      topologyKey: topology.kubernetes.io/zone
      whenUnsatisfiable: ScheduleAnyway
      labelSelector: { matchLabels: { app: ia-recomendacao } }
---
apiVersion: v1
kind: Service
metadata: { name: ia-recomendacao-svc, namespace: ia-ops }
spec:
  selector: { app: ia-recomendacao }
  ports: [{ name: http, port: 8080, targetPort: 8080 }]
  type: ClusterIP
```

3.5 HPA v2 (CPU/Mem)

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata: { name: ia-recomendacao-hpa, namespace: ia-ops }
spec:
  scaleTargetRef: { apiVersion: apps/v1, kind: Deployment, name: ia-recomendacao }
  minReplicas: 3
  maxReplicas: 60
  behavior:
    scaleUp: { stabilizationWindowSeconds: 0, policies: [{ type: Percent, value: 100, periodSeconds:
60 }] }
    scaleDown: { stabilizationWindowSeconds: 300, policies: [{ type: Percent, value: 50, periodSeco
nds: 60 }] }
  metrics:
  - type: Resource
    resource: { name: cpu, target: { type: Utilization, averageUtilization: 70 } }
  - type: Resource
    resource: { name: memory, target: { type: Utilization, averageUtilization: 75 } }
```

3.6 KEDA (Kafka Consumer Lag)

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata: { name: ia-recomendacao-keda, namespace: ia-ops }
spec:
  scaleTargetRef: { name: ia-recomendacao }
  minReplicaCount: 3
  maxReplicaCount: 100
  cooldownPeriod: 120
  triggers:
  - type: kafka
    metadata:
      bootstrapServers: "kafka-0:9092,kafka-1:9092"
      consumerGroup: "rec-cg"
      topic: "ux.view"
```

```
      lagThreshold: "5000"      # escala quando lag > 5k
```

3.7 PDB, NetworkPolicy

```
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata: { name: ia-recomendacao-pdb, namespace: ia-ops }
spec:
  minAvailable: 2
  selector: { matchLabels: { app: ia-recomendacao } }
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata: { name: ia-recomendacao-egress, namespace: ia-ops }
spec:
  podSelector: { matchLabels: { app: ia-recomendacao } }
  policyTypes: ["Egress"]
  egress:
  - to: [{ namespaceSelector: { matchLabels: { name: ia-ops } }, podSelector: { matchLabels: { app:
"redis" } } }]
    ports: [{ protocol: TCP, port: 6379 }]
  - to: [{ namespaceSelector: { matchLabels: { name: ia-ops } }, podSelector: { matchLabels: { app:
"pg" } } }]
    ports: [{ protocol: TCP, port: 5432 }]
  - to: [{ ipBlock: { cidr: 10.0.0.0/16 } }]  # Kafka/Elastic internos
    ports: [{ protocol: TCP, port: 9092 }, { protocol: TCP, port: 9200 }]
```

3.8 Istio: VirtualService + DestinationRule (mTLS, retries)

```
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata: { name: ia-recomendacao-dr, namespace: ia-ops }
spec:
  host: ia-recomendacao-svc.ia-ops.svc.cluster.local
  trafficPolicy:
    tls: { mode: ISTIO_MUTUAL }
    outlierDetection:
      consecutive5xxErrors: 5
      interval: 5s
      baseEjectionTime: 30s
      maxEjectionPercent: 50
    connectionPool:
      http: { http1MaxPendingRequests: 1024, maxRequestsPerConnection: 100 }
      tcp:  { maxConnections: 1000 }
---
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata: { name: ia-recomendacao-vs, namespace: ia-ops }
```

```
spec:
  hosts: ["ia-recomendacao.ia.svc"]  # internal host via mesh
  http:
  - route:
    - destination: { host: ia-recomendacao-svc.ia-ops.svc.cluster.local, port: { number: 8080 } }
      weight: 100
    retries: { attempts: 2, perTryTimeout: 200ms, retryOn: "5xx,connect-failure,reset" }
    timeout: 500ms
```

---

4) Classes de Nós (node pools)

- **compute-std**: serviços HTTP/gRPC (recomendação, jornada).
- **compute-ml**: preditores pesados (LSTM/ARIMA) com `n2-highmem`.
- **io-optimized**: consumidores Kafka/ETL alto throughput.
- **observability**: Elastic/Prometheus/OTel Collector.

---

5) Observabilidade

- **Prometheus**: scrape via annotations; métricas expostas:
  - `request_latency_ms{route}`
  - `kafka_consumer_lag` (KEDA/Exporter)
  - `cache_hit_ratio`
- **Tracing**: OpenTelemetry SDK → OTLP → Collector → Jaeger/Tempo.
- **Logs**: STDOUT → FluentBit → Elastic; retenção 30/60/90d por índice.

---

6) Segurança e Conformidade

- **Pod Security Standard: restricted** (não-root, CAP drop, FS read-only).
- **Secrets** via `Secret` + IAM/KMS; rotação a cada 90 dias.
- **Istio mTLS** strict.
- **RBAC de menor privilégio**.
- **NetworkPolicy** default deny-all com egress whitelists por módulo.

---

7) Desdobramentos Operacionais

- **SLOs**:
  - p95 `/<mod>/health`: **<150ms**; disponibilidade **≥99.95%**.
  - Erro 5xx por módulo: **<0.3%** rolling 7d.
- **Erro Budget** mensal: `1 - SLO`. Gate de deploy se budget esgotado.
- **Rollout**: canary 5%→25%→100% com Istio weights; rollback automático por `outlierDetection`.
- **Backpressure**: fila interna + `429` com `Retry-After`.

---

8) Jobs de Manutenção (exemplos)

```
apiVersion: batch/v1
kind: CronJob
metadata: { name: rec-ann-reindex, namespace: ia-ops }
spec:
 schedule: "0 3 * * *"
 jobTemplate:
   spec:
    template:
      spec:
        serviceAccountName: ia-recomendacao-sa
        restartPolicy: OnFailure
        containers:
        - name: psql
          image: bitnami/pgpool:latest
          command: ["sh","-c","psql \"$PG_DSN\" -c 'REINDEX INDEX CONCURRENTLY ia_ops.user_e
mbedding_vec_ivfflat'"]
          envFrom:
          - configMapRef: { name: ia-recomendacao-cm }
          - secretRef: { name: ia-recomendacao-sec }
```

---

9) Comandos Operacionais (referência)

```
# rollout e status
kubectl -n ia-ops rollout restart deploy/ia-recomendacao
kubectl -n ia-ops rollout status  deploy/ia-recomendacao

# verificar HPA
kubectl -n ia-ops get hpa ia-recomendacao-hpa

# lag Kafka (via KEDA)
kubectl -n ia-ops get scaledobject ia-recomendacao-keda
```

---

10) SLAs & Testes

- **Readiness p95** pós-deploy < **45s**.
- **MTTR** por módulo crítico < **5 min** (auto-heal + rollback).
- **Chaos tests**: semanal (kill pod, node drain, broker down).
- **Load tests**: mensal; alvo `>20k req/s` por zona (módulos HTTP).

💡 Fechamento da Camada

Arquitetura de execução **padronizada e auditável**: manifests completos (Deploy/Service/HPA/KEDA/PDB/NetworkPolicy/Istio), segurança restrita, observabilidade integrada e práticas de rollout/rollback com SLOs e erro budget.

🧠 CAMADA 22 — LOGS VIBRACIONAIS OCULTOS E PAINÉIS INTERNOS (100% TÉCNICA)

1) Objetivo

Definir a infraestrutura de **logs ocultos** para rastrear métricas vibracionais e funcionais e os **painéis internos de IA** usados por DevOps, Data Science e Conselho de Guardiões. Essa camada garante **observabilidade total**, mas sem exposição direta ao usuário final.

---

2) Tipos de Logs Vibracionais

Log	Conteúdo Principal	Armazenamento Primário	TTL
`vibe_log`	score vibracional em cada interação	Firebase + Redis	24h
`oscillation_log`	variação de vibração por janela temporal	PostgreSQL + Kafka (stream)	7 dias
`collective_log`	estados médios de grupos/coletivos	PostgreSQL	30 dias
`feed_log`	score vibracional de conteúdos/postagens	ElasticSearch	90 dias
`correction_log`	ações corretivas disparadas pela IA	PostgreSQL + S3	1 ano

---

3) Estrutura de Banco (DDL)

```
CREATE TABLE vibe_log (
  id BIGSERIAL PRIMARY KEY,
  user_id VARCHAR NOT NULL,
  score NUMERIC(5,3),
  contexto VARCHAR(30),
  trace_id UUID,
  created_at TIMESTAMPTZ DEFAULT NOW()
);

CREATE TABLE oscillation_log (
  id BIGSERIAL PRIMARY KEY,
  user_id VARCHAR NOT NULL,
  variacao NUMERIC(5,3),
  janela INTERVAL,
  trace_id UUID,
  created_at TIMESTAMPTZ DEFAULT NOW()
);

CREATE TABLE collective_log (
  id BIGSERIAL PRIMARY KEY,
  grupo_id UUID NOT NULL,
  usuarios INT,
  score_medio NUMERIC(5,3),
  variacao NUMERIC(5,3),
  acao_disparada VARCHAR(50),
  created_at TIMESTAMPTZ DEFAULT NOW()
);
```

ElasticSearch:

- Índices diários → `logs-vibe-YYYY.MM.DD`.
- ILM (Index Lifecycle Management): hot (30d) → warm (60d) → delete (90d).

---

4) Logs Técnicos — Exemplo JSON

```
{
  "type": "vibe_event",
  "user_id": "u_333",
  "score": 0.27,
  "contexto": "feed",
  "trace_id": "a7b9-88df",
  "oscillation": 0.62,
  "correction": "ritual_recalibracao",
  "ts": "2025-09-02T20:33:11Z"
}
```

---

5) Painéis Internos

5.1 Painel Vibracional Global

- Fonte: `vibe_log`, `oscillation_log`.
- Métricas: score médio por região, % em risco, alertas ativos.
- Dashboard: Grafana + mapas de calor.

5.2 Painel de Anomalias

- Fonte: `oscillation_log`, `correction_log`.
- Gráficos: anomalias por módulo, triggers disparados, taxa de correção.
- Alertas automáticos: Slack/Telegram via webhook.

5.3 Painel de Aprendizado

- Fonte: `feedback_log`, `model_registry`.
- Mostra evolução dos modelos e taxa de aceitação de outputs.

5.4 Painel de Guardiões (Governança)

- Fonte: `model_registry`, `collective_log`.
- Exibe status de modelos candidatos, relatórios de drift, ações do Conselho.

---

6) APIs de Observabilidade

- `GET /api/ia/logs/vibe?user_id=u123&range=24h`
- **200 Response**:

```
[
  { "score": 0.82, "contexto": "chat", "ts": "2025-09-02T20:12:01Z" },
  { "score": 0.65, "contexto": "feed", "ts": "2025-09-02T20:14:22Z" }
```

```
]
```

- `GET /api/ia/painel/anomalias?janela=7d`
- **200 Response**:

```
{
  "total_anomalias": 4312,
  "correcoes_aplicadas": 1203,
  "oscillation_avg": 0.47
}
```

---

7) Métricas Observáveis

- `vibe_score_avg{region}`
- `oscillation_index_avg`
- `collective_events_triggered_total`
- `correction_applied_total{type}`

---

8) SLAs

- Registro de log em banco: **< 50ms**.
- Disponibilidade dos painéis: **99.9%**.
- Atraso máximo em métricas agregadas: **≤ 5s**.

💡 Fechamento da Camada

Os **logs vibracionais ocultos** e **painéis internos** garantem que a IA seja **auditável, transparente e monitorada em tempo real**, sem expor dados sensíveis ao usuário final.

🧠 CAMADA 23 — INTEGRAÇÃO COM IAs NEURAIS E ASSISTENTES VIRTUAIS (FUTURO EXPANDIDO, 100% TÉCNICA)

1) Objetivo

Definir como a IA Operacional poderá se integrar, no futuro, com **IAs neurais externas (BCI/assistentes virtuais)** e **dispositivos de biofeedback**, mantendo segurança, controle e governança.

---

2) Cenários de Integração

Sistema Futuro	Integração Técnica	Propósito
Assistentes Virtuais (Alexa, Siri, Google Assistant)	Webhook + API REST	Sincronizar estados funcionais e contextuais do usuário
BCI (Brain-Computer Interface)	WebSocket + protocolo binário encriptado	Captar sinais neurais e mapear em vetores vibracionais

Sistema Futuro	Integração Técnica	Propósito
Sensores Wearables (HRV, EDA, EEG simplificado)	Bluetooth Low Energy (BLE) + Gateway IoT → Kafka	Enviar sinais fisiológicos para análise vibracional
IA Cognitiva Externa	API GraphQL + assinaturas Kafka	Predições de comportamento futuro e cruzamento de padrões

3) Estrutura de Protocolos

- **VibeRelay Protocol (proposto)**
  - Baseado em **gRPC** com **mTLS**.
  - Mensagens em Protobuf: `NeuralSignal`, `BioFeedback`, `VirtualCommand`.
  - Latência alvo: **< 50ms** (streaming).

Exemplo Protobuf

```
message NeuralSignal {
  string user_id = 1;
  repeated float eeg_vector = 2;
  int64 timestamp = 3;
}
message BioFeedback {
  string user_id = 1;
  float heart_rate = 2;
  float eda_level = 3;
  int64 timestamp = 4;
}
```

4) APIs Futuras

4.1 POST /api/ia/neural/input

```
{
  "user_id": "u_555",
  "eeg_vector": [0.22, -0.15, 0.33],
  "ts": "2025-09-02T20:51:11Z"
}
```

- **200 Response**:

```
{ "ack": true, "mapped_vibe_score": 0.68 }
```

4.2 POST /api/ia/assistente/command

```
{
  "assistant": "alexa",
  "command": "friendapp abrir feed",
```

```
  "context": "voz"
}
```

5) Banco de Dados de Integração

```
CREATE TABLE neural_signals (
  id BIGSERIAL PRIMARY KEY,
  user_id VARCHAR NOT NULL,
  eeg_vector vector(64),
  mapped_score NUMERIC(5,3),
  created_at TIMESTAMPTZ DEFAULT NOW()
);

CREATE TABLE biofeedback_signals (
  id BIGSERIAL PRIMARY KEY,
  user_id VARCHAR NOT NULL,
  heart_rate NUMERIC(5,2),
  eda NUMERIC(5,2),
  created_at TIMESTAMPTZ DEFAULT NOW()
);
```

Redis:

- `neural:last:{user_id}` → última leitura EEG.
- `bio:last:{user_id}` → último feedback fisiológico.

6) Pseudocódigo

```
def integrar_neural(user_id, eeg_vector):
    score = mapear_para_vibracao(eeg_vector)
    salvar_pg(user_id, eeg_vector, score)
    atualizar_cache(user_id, score)
    if score < 0.3:
        disparar_alerta(user_id)
```

7) Logs Técnicos

```
{
  "type": "neural_integration",
  "user_id": "u_555",
  "eeg_vector": [0.22, -0.15, 0.33],
  "mapped_vibe_score": 0.68,
  "trace_id": "a199-77df",
  "ts": "2025-09-02T20:55:27Z"
```

```
}
```

8) Segurança

- **DUC obrigatório** para qualquer dispositivo externo.
- **Criptografia**: TLS 1.3 + AES-256 para dados em trânsito.
- **Sandbox de integração** → IA externa nunca atua diretamente, apenas fornece sinais/sugestões.
- **Gate humano** para ativações de modelos novos (Conselho de Guardiões).

9) SLAs

- Processamento de sinal neural: **< 80ms**.
- Integração com assistentes virtuais: **< 200ms**.
- Buffer máximo de biofeedback: **≤ 2s**.

💡 Fechamento da Camada

A integração futura da IA Operacional com **IAs neurais, assistentes virtuais e sensores** é suportada por **protocolos, bancos e APIs seguras**, mas sempre sob controle humano e compliance estrito.

CAMADA 24 — MAPA DE INTEGRAÇÕES E DEPENDÊNCIAS DO ECOSSISTEMA INTELIGENTE (100% TÉCNICA)

1) Objetivo

Consolidar todas as **entradas, saídas, fluxos e dependências** entre os módulos da IA Operacional e os demais sistemas do FriendApp. Essa camada garante que os devs saibam exatamente **de onde vêm os dados, para onde vão e como trafegam**.

2) Integrações por Módulo

Módulo	Entradas Principais	Saídas / Consumo	Dependências Críticas
IA Recomendação	Kafka (`ux.click` , `ux.view`), vetores PGVector	Redis (Top-K), API `/recomendacao`	PG (vetores), Redis, Kafka
IA Jornada	Kafka (`journey.abandono`), eventos UX	API `/jornada/aplicar` , logs ajustes	Firestore, PG, Redis
IA Segurança	Kafka (`sec.login` , `sec.ip_change`)	API `/security/analyze` , bloqueios	PG criptografado, Redis, Elastic
IA Performance	Métricas Prometheus, Kafka (`perf.latency`)	API `/performance/metrics` , scaling	Kubernetes HPA/KEDA
IA Sensorial	`feed.new_post` , `feed.engagement`	API `/feed/analisar` , ajustes feed	Elastic, Redis
IA Missões	Scores de risco (IA Segurança), vibrações	API `/missoes/gerar` , triggers	PG, Redis, Orchestrator
IA Moderadora	Kafka (`chat.message` , `feed.comment`)	API `/moderador/analyze` , flags	Elastic, PG, Redis

Módulo	Entradas Principais	Saídas / Consumo	Dependências Críticas
IA Global	Logs intermodulares	API `/global/status`, relatórios	Todos os módulos

3) Fluxos Assíncronos (Kafka)

- **Entradas**
  - `ux.view`, `ux.click`, `ux.scroll` → IA Recomendação + Jornada
  - `journey.abandono` → IA Jornada
  - `sec.login`, `sec.ip_change` → IA Segurança
  - `feed.new_post` → IA Sensorial
  - `chat.message_sent` → IA Moderadora
- **Saídas**
  - `rec.topk.generated` → Redis + API
  - `journey.adjustments` → Firestore + API
  - `security.blocked` → Redis + PG
  - `missions.generated` → API + Redis

4) Dependências Técnicas

Origem	Destino	Tipo	Forma de Integração
App (mobile/web)	Kafka	Evento	Producer SDK
Kafka	IA Módulos	Evento	Consumer assíncrono
IA Módulos	Redis	Cache	Pub/Sub + TTL
IA Módulos	PostgreSQL	Persistência	CDC → Kafka
IA Módulos	ElasticSearch	Observabilidade	Logstash/FluentBit
IA Módulos	API REST/gRPC	Resposta ao app	HTTP/2 + mTLS
Orchestrator	Todos os Módulos	Coordenação	gRPC interno

5) Logs de Integração

Exemplo JSON-line:

```
{
  "type": "integration_event",
  "origem": "ia-recomendacao",
  "destino": "redis",
  "acao": "cache_topk",
  "status": "success",
  "latency_ms": 42,
  "trace_id": "88ac-22fe",
  "ts": "2025-09-02T21:03:17Z"
}
```

6) Métricas Observáveis

- `integration_latency_ms{origem,destino}`
- `integration_failures_total{modulo}`
- `integration_throughput_total{canal}`
- `integration_cache_hit_ratio`

---

7) SLA de Integrações

- Latência de integração crítica (Kafka → IA → Redis): **< 150ms**.
- Disponibilidade ≥ **99.95%**.
- Falha de integração → fallback automático em **< 1s**.

---

8) Diagrama de Integrações (Simplificado)

```
graph LR
App —>|Eventos UX| Kafka
Kafka —> Recomendacao
Kafka —> Jornada
Kafka —> Seguranca
Kafka —> Sensorial
Kafka —> Moderadora
Recomendacao —> Redis
Jornada —> Firestore
Seguranca —> PG
Seguranca —> Redis
Sensorial —> Elastic
Missoes —> Redis
Todos —> Orchestrator
Orchestrator —> APIs
```

💡 Fechamento da Camada

A Camada 24 consolida todas as **dependências do ecossistema da IA**, em **mapas de entrada/saída, bancos, caches e fluxos Kafka**, permitindo que os devs entendam o **caminho completo de cada dado**.

🧠 CAMADA 25 — PSEUDOCÓDIGO MESTRE DO CICLO DA IA + SLOs GLOBAIS (100% TÉCNICA)

1) Objetivo

Formalizar o **ciclo fim-a-fim** da IA Operacional (captura → features → decisão por módulo → política → entrega → feedback → aprendizado), com **pseudocódigo executável**, **contratos de mensageria**, **políticas de resiliência** e **SLOs globais**.

---

2) Ciclo Mestre (visão geral)

```
flowchart LR
A[App Event] --> B[Kafka topics]
B --> C[Feature Builder]
C --> D{Router/Policy}
D -->|security| S[IA Segurança]
D -->|journey| J[IA Jornada]
D -->|rec| R[IA Recomendação]
D -->|perf| P[IA Performance]
S --> E[Policy Gate]
J --> E
R --> E
P --> E
E --> F[Serving: Redis/API]
F --> G[Tracing+Logs]
G --> H[Feedback → Kafka]
H --> I[Model Ops: Train/Eval/Promote]
```

---

3) Pseudocódigo Mestre (idempotente, com tracing e política)

```
# Contexto comum
from typing import Dict, Any
TraceId = str

def handle_event(evt: Dict[str, Any], trace_id: TraceId):
    # 1) Idempotência (dedupe por event_id + source)
    if dedupe_seen(evt["id"]):
        log_skip(evt, trace_id)
        return OK("duplicate")

    # 2) Extração/normalização de features (com decaimento temporal)
    feats = feature_builder(evt)  # O(μs) + cache

    # 3) Roteamento por tipo + prioridade
    mod = route(evt["type"])      # security>journey>rec>perf
    pri = priority(evt["type"])

    # 4) Execução do módulo (com timeout e budget de latência)
    with budget(pri) as b:
        if mod == "security":
            dec = security_decide(feats)
        elif mod == "journey":
            dec = journey_decide(feats)
        elif mod == "rec":
            dec = rec_rank(feats)
        elif mod == "perf":
            dec = perf_adjust(feats)
        else:
```

```
        return ERR("unknown_module")

    # 5) Policy Gate (compliance, risco, caps, diversidade)
    dec2 = apply_policies(dec, evt["user_id"])

    # 6) Persistência mínima + entrega (Redis/API) com TTL
    out = serve(dec2, user_id=evt["user_id"], context=evt.get("context"))
    persist_minimal(evt, dec2, out, trace_id)

    # 7) Telemetria e feedback loop (assíncrono)
    emit_feedback(evt, dec2, out, trace_id)

    return OK("served", latency_ms=lat())
```

Observações técnicas

- **Idempotência**: chave `event_id` + `producer_id` em Redis `SETNX` com TTL curto.
- **Orçamento de latência**: `budget(priority)` aplica deadlines diferentes (crit vs. soft).
- **Policy Gate**: aplica **risk caps**, **MMR/diversidade**, **frequency capping**, **blocklist** do usuário.
- **Serving**: `ZADD rec:topk:{user}:{ctx}` (TTL 5–15 min) e resposta HTTP/gRPC.

---

4) Contratos de Mensageria (Kafka/Avro)

- **Padrão de tópicos**: `domain.event_type.v1`
- **Garantia**: *at-least-once*; ordenação por *partition-key* (ex.: `user_id`).
- **Schema (Avro) — `ux.view.v1` (exemplo)**:

```
{
 "type":"record","name":"UxViewV1","namespace":"friendapp.ux",
 "fields":[
  {"name":"event_id","type":"string"},
  {"name":"user_id","type":"string"},
  {"name":"screen","type":"string"},
  {"name":"ts","type":{"type":"long","logicalType":"timestamp-millis"}}
 ]
}
```

Idempotência no consumo

```
def consume(msg):
    if redis.setnx(f"dedupe:{msg.event_id}", 1):
        redis.expire(f"dedupe:{msg.event_id}", 600)
        process(msg)
    else:
        return  # drop duplicate
```

---

5) Políticas de Resiliência

- **Retries**: exponencial com *jitter* (100–400ms, máx 3).
- **Circuit breaker**: abre após 5× 5xx em 30s; meia-vida 60s.
- **Fallbacks**:
  - Recomendação: *popularidade contextual* + cache anterior.
  - Jornada: presets de UX (layout básico) + texto padrão.
  - Segurança: `challenge` quando análise indisponível.
- **Backpressure**: fila interna + `429 Retry-After` no edge.
- **Time-budget**: críticas (segurança) ≤ 50ms; UX ≤ 150ms; rec ≤ 200ms.

---

6) SLOs Globais (agregados)

Área	SLO p95	Dispo	Erro 5xx	Observações
/recomendacao	≤ 200 ms	≥99.95%	≤0.5%	Cache ≥85%
/jornada/	≤ 150 ms	≥99.95%	≤0.5%	Ajuste atômico
/security/analyze	≤ 50 ms	≥99.99%	≤0.2%	Preemptivo
Ingestão Kafka→Serve	≤ 150 ms (end-to-end)	≥99.95%	n/a	Lag < 5k (KEDA)
CDC Debezium	Lag ≤ 2 s	≥99.9%	n/a	Alertas > 5 s
Model Ops promote	≤ 48 h (batch)	n/a	n/a	Gate humano

Error budget mensal = `1 - SLO`. Release é bloqueado se budget esgotar.

---

7) Testes (gate de release)

- **Contract** (Pact): produtor/consumidor das APIs e tópicos.
- **Load** (k6/Locust): alvo ≥ 20k req/s por zona; picos 2×.
- **Chaos**: matar pod, isolar broker, aumentar lag; SLOs não podem romper.
- **E2E sintético**: cenários frios/quentes; verificação de idempotência.
- **Segurança**: fuzz de payload, JWT inválido, RBAC negativo.

---

8) Observabilidade Global

- **Tracing**: W3C `traceparent` propagado; OpenTelemetry → Jaeger/Tempo.
- **Métricas padrão**: `_latency_ms`, `_error_rate`, `kafka_consumer_lag`, `cache_hit_ratio`.
- **Logs**: JSON-line com `trace_id`, `user_id` pseudonimizado, `context`.

Exemplo:

```
{"type":"serve","endpoint":"/api/ia/recomendacao","trace_id":"b1c-...","latency_ms":91,"status":200}
```

---

9) Segurança & Compliance

- JWT + mTLS interno; RBAC por rota.

- PII isolada; `user_id` pseudonimizado nos logs.
- DUC/DCO verificados antes de outputs sensíveis.
- Criptografia AES-256 at-rest; TLS 1.3 in-transit.

---

10) Model Ops (resumo operacional)

- Registro: `model_registry` (status: candidate/approved/prod).
- Promoção: A/B + Conselho de Guardiões; rollback automático por queda de CTR/Precision@K > 5% do baseline.
- Drift: KS-test horário; alerta > 0.1.

---

11) Runbooks (incidentes críticos)

Incidente	Detecção	Mitigação imediata	Pós-mortem
Lag Kafka alto (>5k)	`keda` + alerta	Escalar consumers; pausar lote não crítico	RCA + tuning
p95 /recomendacao > 200ms	Prometheus alerta	Forçar cache preset; reduzir K no Top-K	Ajustar ANN
Falha /security/analyze	Health fail + 5xx	`challenge` default; circuit open 60s	Testes e rollback
CDC Debezium > 5s	Metric lag	Reconciliar jobs; drenagem controlada	Capacity plan

---

🔗 SEÇÃO FINAL — INTEGRAÇÕES & DEPENDÊNCIAS CÍCLICAS (100% TÉCNICA)

> Documentação de/para para que nenhum dev pergunte “depende de quê?” ou “de onde vem o dado?”. Contém matrizes de integrações, falhas e fallbacks, responsáveis e contratos.

A) Matriz “De → Para” (módulos × canais)

De (Origem)	Para (Destino)	Canal	Contrato	Observações
App (web/mobile)	Kafka `ux.*`	Producer	Avro `UxViewV1`	Partição por `user_id`
App	`/api/ia/*`	HTTP/gRPC	REST v1	JWT + `X-Trace-Id`
IA Recomendação	Redis `rec:topk:*`	Cache	ZSET (score,item)	TTL 5–15 min
IA Jornada	Firestore `/ux_*`	Efêmero	JSON doc	TTL 7d
IA Segurança	PG `security_events`	OLTP	DDL v1	RLS + criptografia
IA Performance	Prometheus/Elastic	Métricas/Logs	OpenMetrics / JSON	Retenção 30–90d
Orchestrator	Módulos IA	gRPC interno	Protobuf v1	mTLS, RBAC
Debezium (CDC)	Kafka `db.ia_ops.*`	Stream	JSON/Avro	Lag ≤ 2 s
RA (AR Engine)	Kafka `ra.session`	Evento	Avro `RaSessionV1`	Integração presencial
Wearables/IoT	Gateway → Kafka	Evento	Protobuf/JSON	Buffer ≤ 2 s

---

B) Dependências por Recurso (SPOF e fallback)

Recurso	Consumidores	Tipo	SLO	Fallback
Redis	rec, journey, security	Cache	p95 < 5 ms	Preset em memória + reduzir K

Recurso	Consumidores	Tipo	SLO	Fallback
PG `ia_ops`	todos	OLTP	p95 < 120 ms	Fila local + retry + degradação read-only
Kafka	todos	Bus	lag < 5k	Buffer local + KEDA scale-up
Elastic	perf, moderação	Logs	99.9%	Buffer stdout + reenvio
Firestore	jornada	Efêmero	99.9%	Persistir mínimo em PG

C) Matriz de Falhas → Detecção → Mitigação

Falha	Detecção	Mitigação técnica
Redis down	`ping` falha / timeouts	Circuit breaker + cache em memória + TTL curto
PG conexão saturada	p95 SELECT ↑ / pool esgotado	PgBouncer hard cap; backoff; priorizar rotas críticas
Kafka broker indisponível	lag ↑ + erro consumer	Rebalance; aumentar partições; KEDA scale; DLQ opcional
Elastic ingest congestiona	fila > limiar	Reduzir sampling logs; ILM warm; compressão
CDC parou (Debezium)	lag > 5s	Restart connector; replay WAL; reconciliar offsets

D) Contratos por API/Tópico (resumo)

Interface	Versão	Produtor	Consumidor	Esquema/Contrato
`/api/ia/recomendacao`	v1	rec	app/gateway	OpenAPI v1
`/api/ia/jornada/*`	v1	journey	app/gateway	OpenAPI v1
`/api/ia/security/*`	v1	security	app/gateway	OpenAPI v1
`ux.view`	v1	app	rec/journey	Avro `UxViewV1`
`feed.new_post`	v1	app	sensorial	Avro `FeedPostV1`
`db.ia_ops.interaction`	v1	debezium	feature-builder	JSON/Avro
`ra.session` (RA)	v1	RA eng.	orchestrator/journey	Avro `RaSessionV1`

E) Ownership (RACI)

Módulo/Interface	R (responsável)	A (aprova)	C (consulta)	I (informa)
Recomendação (/rec, ux.*)	IA-Rec Team	Orchestrator Lead	Data Science, Segurança	SRE
Jornada (/jornada, ux.*)	IA-Journey Team	Orchestrator Lead	Produto, DS	SRE
Segurança (/security)	IA-Sec Team	CISO/Guardião	Jurídico, Orchestrator	SRE
Performance (perf.*)	SRE	Orchestrator Lead	Todos	Produto
RA (`ra.session`)	RA/AR Team	Orchestrator Lead	Journey, DS	SRE

F) Sequência de Inicialização / Desligamento

Startup (ordem):

1. Kafka, Redis, PG, Elastic, Firestore.
2. Debezium (CDC).
3. IA módulos (sem tráfego) → *warm-up* (preload ANN, caches).

4. Orchestrator + Istio routes (peso 0%).
5. Canary 5% → 25% → 100% (health OK).

Shutdown (ordem):

1. Congelar ingest não crítica; drenar filas.
2. Orchestrator bloqueia novos tráfegos.
3. Módulos IA → *preStop* (flush logs/caches).
4. Parar CDC; persistência final.

G) Legenda Técnica (ícones/emoji)

- 🔌 **API/HTTP** | 📨 **Kafka** | 🧰 **Cache/Redis** | 🗄️ **DB/PG** | 📈 **Métricas/Logs** | 🧪 **Testes** | 🛡️ **Segurança** | 🧭 **Orchestrator** | 🕶️ **RA/AR**

H) Checklist de Integração (por feature)

- 🔌 **APIs** definidas (OpenAPI) e testadas (contract).
- 📨 **Tópicos Kafka** criados, com schema versionado.
- 🧰 **Caches** com TTL e chaves padronizadas.
- 🗄️ **DDL** aplicado + índices + RLS.
- 📈 **Métricas** e **tracing** expostos.
- 🛡️ **RBAC** e **mTLS** revisados.
- 🧪 **Load/Chaos** ok; SLOs dentro do budget.

I) RA (Realidade Aumentada) — Integração mínima

- Evento `ra.session` (start/stop, posição, ponteiros de interação).
- Jornada consome `ra.session` para **ajuste de UX** in-situ.
- Segurança monitora padrões anômalos (loop/latência).
- Armazenamento: `ra_session_log` (PG particionado) + cache de sessão (Redis).

✅ Encerramento Técnico

- **Camada 25** entregue com **ciclo mestre, políticas e SLOs globais**.
- **Seção Final de Integrações & Dependências Cíclicas** entregue com **matrizes, ownership, falhas e contratos**.