

➡ MANUAL TÉCNICO DEFINITIVO — MAPA DE FREQUÊNCIA (FRIENDAPP)

⚡ CAMADA 01 — VISÃO E PROPÓSITO DO MAPA DE FREQUÊNCIAS

🎯 Objetivo

O **Mapa de Frequências** é a **interface coletiva de energia** do FriendApp. Ele transforma a atividade individual de cada usuário, a vibração dos locais parceiros e os eventos ativos em uma **visualização única, imersiva e dinâmica**.

O propósito é permitir que cada pessoa veja:

1. **A sua própria frequência individual** (check-ins, intenções, oscilações).
2. **O estado vibracional coletivo de uma região** (bairro, cidade, hotspots).
3. **A energia de locais e eventos parceiros**, com camadas de contexto e relevância.

✨ Diferenciais

- **Não é um mapa comum:** em vez de mostrar apenas “onde” as pessoas estão, mostra **como elas estão**.
- **Escalabilidade Global:** arquitetado em **pré-agregações geográficas (geohashing)** que permitem atualizar estados coletivos sem sobrecarregar o sistema.
- **Tempo Real Inteligente:** pontos individuais fluem em tempo real (via Firestore), enquanto zonas e estados são recalculados em background jobs, garantindo performance.
- **Privacidade por Design:** inclui geofuzzing dinâmico e **zonas de silêncio** configuráveis pelo usuário.

📌 Aplicações Práticas

- **Para Usuários:** entendem seu estado energético atual e como ele ressoa com o coletivo.
- **Para Locais Parceiros:** visualizam a vibração de seus estabelecimentos e podem atrair conexões compatíveis.
- **Para Eventos:** mostram antecipadamente a energia de um encontro, aumentando a expectativa e engajamento.
- **Para a IA Aurah Kosmos:** fornece insumos contínuos sobre a dinâmica vibracional global para ajustar recomendações, selos e conexões.

⚡ CAMADA 02 — ARQUITETURA DE BANCO E ESTRUTURA DE DADOS

🎯 Objetivo da Camada

Definir a **base tecnológica** que sustenta o Mapa de Frequências, garantindo **baixa latência, escalabilidade global e segurança de dados sensíveis**.

O mapa precisa manipular **três camadas simultâneas de informação**:

1. **Usuário Individual (tempo real)** → localização + frequência pessoal.
2. **Zonas Coletivas (pré-agregação)** → estados vibracionais de regiões.
3. **Locais/Eventos Parceiros (híbrido)** → pontos comerciais e sociais que interagem com os usuários.

Bancos de Dados Escolhidos

Banco	Função	Justificativa Técnica
PostgreSQL + PostGIS	Dados geográficos relacionais (cidades, regiões, polígonos, locais parceiros)	Robusto, queries espaciais rápidas, suporte nativo a geohashing.
Firestore (NoSQL)	Presença de usuários em tempo real (check-ins ativos, estados instantâneos)	Latência baixa, sincronização em tempo real, ótimo para updates dinâmicos.
Redis (Cache In-Memory)	Pré-agregação de células vibracionais (H3/S2)	Armazena cálculos recentes para consultas ultrarrápidas.
ElasticSearch (opcional)	Logs de busca, filtros avançados, histórico energético	Permite queries textuais e analíticas em paralelo.

Estratégia de Dados

1. Usuário Individual

- Persistido no Firestore.
- Dados: `id_user`, `timestamp`, `freq_atual`, `geohash`, `intencao`.
- TTL de 60 minutos (expira após inatividade).

2. Zonas Agregadas

- Pré-calculadas em background jobs a cada **5 minutos**.
- Estrutura H3 (hexágonos globais) → cada célula armazena `media_vibracional`, `densidade_usuarios`, `estado`.
- Armazenadas em Redis + PostgreSQL (persistência).

3. Locais/Eventos

- Relacionados via **IDs únicos no PostgreSQL**.
- Estados dinâmicos sincronizados com Firestore (check-ins ativos).

Estruturas de Tabelas (Simplificadas)

Tabela `users_presence` (Firestore → streaming)

```
{
  "id_user": "uuid",
  "timestamp": 1737412345,
  "freq_atual": 780,
  "geohash": "6gkz7p",
  "intencao": "social_evento"
}
```

Tabela `zones_state` (Redis + PostgreSQL)

Campo	Tipo	Descrição
<code>zone_id</code>	H3 string	Identificador único da célula vibracional
<code>freq_media</code>	float	Média ponderada da célula
<code>densidade</code>	int	Número de usuários ativos

Campo	Tipo	Descrição
estado	enum	{Expansão, Pico, Transição, Colapso}
last_update	timestamp	Última atualização

Tabela `places_state` (PostgreSQL)

Campo	Tipo	Descrição
id_place	uuid	Identificador do local parceiro
nome	varchar	Nome do local
freq_atual	float	Estado vibracional atual
checkins_ativos	int	Quantidade de usuários no momento
tier	enum	{Essencial, Premium, Visionário}

Segurança e Privacidade

- **Geofuzzing Dinâmico:** distorção maior em zonas residenciais, menor em áreas públicas.
- **Zonas de Silêncio:** Firestore ignora updates dentro de locais definidos pelo usuário (ex: casa).
- **Logs Criptografados:** toda consulta ou cálculo armazenado com hash duplo.

Fechamento

“Esta arquitetura híbrida garante que o Mapa de Frequências seja rápido como um radar, sólido como uma base relacional e fluido como o estado energético que representa.”

CAMADA 03 — FLUXO DE DADOS (TEMPO REAL + AGREGAÇÃO ASSÍNCRONA)

Objetivo

Descrever, com precisão executável, **como o Mapa de Frequências recebe, processa e expõe os dados** combinando:

- **Tempo real** para presença individual (Firestore/WebSocket).
- **Pré-agregação por geohash (H3/S2)** para estados coletivos (Redis/PostgreSQL).
- **Jobs em background** para cálculo robusto (trimmed mean + time-decay).
- **Mecanismos de privacidade e antifraude** (geofuzzing dinâmico, zonas de silêncio, rollback).

1) Topologia Lógica (alto nível)

[App móvel]

- API Gateway (JWT, rate limit)
- Privacy Filter (Zonas de Silêncio + Geofuzzing Dinâmico)
- Firestore (user_presence_rt)
- Event Bus (Kafka/PubSub): presence.v1, feedback.v1
- Stream Processor (dedupe/idempotency + windowing)
 - Redis (zones_state_cache) ← Background Aggregator (cada 5 min)
 - PostgreSQL (zones_state, places_state, transitions_log)
 - Outbound Topics: zone_state.v1, alerts.v1

- **Tempo real (pontos individuais):** Firestore sincroniza presença atual do usuário (TTL).
- **Estado coletivo (zonas):** calculado a cada 5 min por **Background Aggregator** e publicado em Redis + PostgreSQL.

2) Canais de Entrada (streams)

Fonte	Tópico/Event	Payload (resumo)	Observações
Presença do usuário	<code>presence.v1</code>	<code>{user_id, ts, lat, lon, h3_res7, freq_raw, intent}</code>	Produzido após Privacy Filter
Feedback emocional	<code>feedback.v1</code>	<code>{user_id, place_id?, h3, emotion, intensity, ts}</code>	Opcionalmente vinculado a <code>place_id</code>
Eventos/Locais	<code>place_event.v1</code>	<code>{place_id, type, start/end, expected_density}</code>	Bônus de estado (PICO/EXPANSÃO)
Antifraude	<code>presence.flag.v1</code>	<code>{user_id, reason, ts, impact_ids}</code>	Gera rollback (compensation)

Idempotência: chave de dedupe = `user_id + floor(ts/60s)` (1 update por minuto por usuário).

3) Tempo Real (individual)

Firestore collection: `user_presence_rt`

```
{
  "user_id": "uuid",
  "ts": 1737412345,
  "freq_current": 780,    // 0..1000 (normalizado)
  "h3_res7": "871fb4ac...",
  "intent": "social_evento",
  "expires_at": 1737415945 // TTL ~ 60min
}
```

- Atualização **via WebSocket/SDK** para o cliente.
- **Rate limit:** 1 update/60s por usuário (reduz churn; suficiente para mapas).
- **Expiração automática:** presença inativa some do mapa sem intervenção manual.

4) Pré-Agregação por Geohash (H3)

Background Aggregator (cron: a cada 5 min, janela deslizante de 10 min):

1. Carrega todas as presenças/feedbacks recentes por **célula H3 (res. 7/8)**.
2. Aplica **geofuzzing dinâmico** (já aplicado na ingestão, ver §8).
3. Calcula métricas por célula:
 - **densidade_usuarios**
 - **freq_media_robusta** (média aparada + time-decay)
 - **estado_coletivo** (PICO/EXPANSÃO/TRANSIÇÃO/COLAPSO)
4. Persiste:
 - **Redis** (`zones_state_cache`) → leitura rápida da UI.
 - **PostgreSQL** (`zones_state`) → histórico/BI/auditoria.
5. Publica em `zone_state.v1` para assinantes (IA, painéis).

Resolução H3 por zoom sugerido:

- **res=7** (cidade/large bairros)

- **res=8** (bairros/áreas urbanas densas)
- **res=9** (micro-zonas quando o zoom aproxima)

5) Fórmula de Agregação (robusta)

5.1 Time-decay (peso temporal)

Para cada ponto **i** na janela:

```
Δt_i = now - ts_i // em minutos
weight_time_i = exp(-Δt_i / τ)
τ (tau) default = 20 min // mais recente pesa MUITO mais
```

5.2 Trimmed mean (média aparada)

- Ordenar **freq_raw** dos pontos na célula.
- Remover **10% menores** e **10% maiores** (outliers).
- Aplicar média ponderada pelos **weight_time_i** restantes.

5.3 VIB_SCORE (célula)

```
VIB_SCORE_zone = clamp(
  trimmed_mean_weighted(freq_raw)
+ density_boost(densidade_usuarios)
+ event_boost(atividade_evento)
- toxicity_penalty(flags_negativos)
, 0, 1000)
```

- **density_boost** : função logarítmica suave (evita explosões artificiais).
- **event_boost** : +[5..25] conforme **place_event.v1**.
- **toxicity_penalty** : -[10..100] com base em denúncias/flags.

5.4 Mapeamento de Estado Coletivo

Estado	Regra Base (exemplo)
PICO	VIB_SCORE_zone ≥ 720 e densidade ≥ limiar_res ou evento ativo
EXPANSÃO	VIB_SCORE_zone ∈ [520..719] e tendência ↑ (últimas 2 janelas)
TRANSIÇÃO	VIB_SCORE_zone ∈ [320..519] ou tendência ~ estável
COLAPSO	VIB_SCORE_zone < 320 ou toxicity_penalty elevado

Limiar de densidade varia por resolução H3; calibrado por cidade.

6) Rollback Antifraude (compensação)

Quando um ponto é marcado como fraude (ex.: GPS spoofing):

1. Emite **presence.flag.v1** com **impact_ids** (células afetadas).
2. **Compensation Worker** recalcula as células imediatamente:
 - Remove contribuições do **user_id/ts** suspeito.

- Reaplica **time-decay + trimmed mean**.
- Atualiza Redis/PostgreSQL e `transitions_log`.

Garantia: eventual consistency em < 2 min para células impactadas.

7) Exposição para o Frontend (consulta)

7.1 Zonas (coletivo)

GET `/api/map/zones?bbox=<minLat,minLon,maxLat,maxLon>&res=7`

- **Headers:** `Cache-Control: max-age=60`, `ETag`
- **Resp:**

```
{
  "resolution": 7,
  "updated_at": "2025-08-25T18:12:00Z",
  "zones": [
    {
      "h3": "871fb4ac...",
      "vib_score": 674,
      "density": 42,
      "state": "EXPANSAO",
      "last_update": "2025-08-25T18:10:00Z"
    }
  ]
}
```

7.2 Presença Individual (tempo real)

GET `/api/map/user/me`

```
{
  "user_id": "uuid",
  "freq_current": 792,
  "state_hint": "ALTA",
  "last_update": "2025-08-25T18:11:07Z"
}
```

7.3 Locais/Hotspots

GET `/api/map/places?bbox=...`

```
{
  "places": [
    {
      "place_id": "loc_432",
      "name": "Café Aurora",
      "vib_score": 708,
      "checkins_active": 15,
      "state": "PICO",
      "tier": "Premium",
      "last_update": "2025-08-25T18:10:00Z"
    }
  ]
}
```

```
}  
]  
}
```

8) Privacidade por Design

8.1 Zonas de Silêncio (user opt-out)

- O app armazena **polígonos privados** (casa/trabalho).
- Se o GPS cair dentro de um polígono, **NÃO** enviamos presença (nem distorcida).
- Aplicado no **Privacy Filter** antes de publicar `presence.v1`.

8.2 Geofuzzing Dinâmico

- **Maior distorção** em áreas residenciais/baixa densidade (ex.: 150–300m).
- **Menor distorção** em áreas públicas/alta densidade (ex.: 30–70m).
- Aleatoriedade **coerente por sessão** (evita clusterização reversa).

9) Segurança, Limites e Idempotência

- **Autenticação:** JWT (Firebase/Auth0), escopos de acesso.
- **Webhooks:** HMAC SHA-256 (assinatura e replay-protection).
- **Rate limits:**
 - Presença: 1 req/min por usuário.
 - Feedback: 10 req/h por usuário.
 - Zones fetch: 10 req/10s por IP (burst control).
- **Idempotência:** `Idempotency-Key` por operação sensível; consumidores **“at-least-once”** com dedupe por chave.

10) Observabilidade & SLOs

Métricas (Prometheus/Grafana):

- `zones_agg_latency_seconds` (p95 < 90s por janela de 5 min)
- `zones_cache_hit_ratio` (alvo > 0.85)
- `presence_ingest_rate`
- `fraud_compensation_latency_seconds`
- `api_zones_p95_latency_ms` (alvo < 180ms)

Logs e Tracing: OpenTelemetry com `trace_id` propagado (API → stream → jobs).

SLOs:

- **Staleness máximo** das zonas: **10 min** (cache + última janela).
- **Disponibilidade API** `/zones`: 99.9%
- **Reprocessamento antifraude:** < 2 min (p95) nas células impactadas.

11) Pseudocódigo (core)

11.1 Aggregator (a cada 5 min)

```
def run_aggregator(window_minutes=10, h3_res=7, tau=20, trim=0.10):
    buckets = group_by_h3(fetch_presence(last=window_minutes), h3_res)
    outputs = []
    for h3_id, points in buckets.items():
        points = privacy_already_applied(points)      # sanity
        weights = [exp(-(now()-p.ts)/tau) for p in points]
        freq = [p.freq_raw for p in points]
        # trimmed mean:
        freq_trimmed, weights_trimmed = trim_symmetric(freq, weights, ratio=trim)
        vib = weighted_mean(freq_trimmed, weights_trimmed)
        vib += density_boost(len(points), h3_res)
        vib += event_boost(h3_id, now())
        vib -= toxicity_penalty(h3_id, last=window_minutes)
        vib = clamp(vib, 0, 1000)
        state = map_to_state(vib, len(points), h3_res)
        outputs.append({ "h3":h3_id, "vib":vib, "density":len(points), "state":state })
    write_to_redis(outputs)
    persist_to_postgres(outputs)
    publish("zone_state.v1", outputs)
```

11.2 Compensation (rollback)

```
def compensate_presence(flag):
    impacted = estimate_impacted_cells(flag.user_id, flag.ts_range)
    for h3_id in impacted:
        recalc_cell(h3_id)      # recomputa aplicando mesma janela e pesos
    log_transition(impacted)
```

12) Contratos de Erro (padrão)

Código	Mensagem	Contexto
INVALID_LOCATION	Fora da área permitida ou em Zona Silenciosa	presença/check-in
RATE_LIMITED	Limite de requisições excedido	ingestão/consulta
STALE_ZONE	Sem dados atuais; retornando última agregação válida	consulta zonas
PAYLOAD_INVALID	Campos ausentes/mal formatados	ingestão

13) Retenção & Custos

- **Kafka/PubSub (raw presence):** retenção 24h + compactação por chave.
- **Redis (zones_state_cache):** TTL 10 min; write-through para PostgreSQL.
- **PostgreSQL (zones_state histórico):** retenção 90 dias (particionamento mensal).
- **Arquivamento frio:** BigQuery/S3 para análises históricas.

✓ Fechamento

Esta camada consolida o **coração operacional** do Mapa de Frequências: **tempo real onde importa (pontos individuais)**, **pré-agregação robusta para o coletivo**, **privacidade forte por design** e **antifraude com compensação** — tudo com SLOs mensuráveis e custos sob controle.

◆ CAMADA 04 — APIs DO MAPA (CONTRATOS, PAYLOADS E VERSIONAMENTO)

“O mapa não é só visual. Ele é sustentado por contratos claros, dados consistentes e versionamento preciso. Sem APIs sólidas, o pulso global não existe.”

🎯 Objetivo da Camada

Definir com precisão **todas as APIs utilizadas pelo Mapa de Frequências**, seus **endpoints**, **métodos**, **payloads de entrada/saída e versionamento**, garantindo que não haja ambiguidade no desenvolvimento e integração entre os módulos.

📡 Estrutura de Endpoints

Endpoint	Método	Descrição	Autenticação	Frequência
<code>/api/v1/mapa/frequencia/cell</code>	GET	Retorna o estado vibracional agregado de uma célula geográfica (geohash H3/S2)	Token JWT + scope <code>mapa:read</code>	Chamado pelo app sempre que usuário navega no mapa
<code>/api/v1/mapa/frequencia/checkin</code>	POST	Registra um check-in vibracional do usuário	Token JWT + scope <code>mapa:write</code>	Disparado pelo app ao confirmar presença em um local
<code>/api/v1/mapa/frequencia/realtime</code>	GET (stream)	Retorna dados em tempo real de usuários em um raio específico (Firestore streaming)	Token JWT + scope <code>mapa:read</code>	Apenas para usuários Premium
<code>/api/v1/mapa/frequencia/historico</code>	GET	Retorna histórico vibracional de uma célula (últimos 7 dias ou 30 dias)	Token JWT + scope <code>mapa:read</code>	Consultas ocasionais
<code>/api/v1/mapa/frequencia/alerta</code>	POST	Envia alerta vibracional de colapso/expansão detectado	Token interno (Aurah Kosmos)	Evento automático
<code>/api/v1/mapa/frequencia/admin/recalcular</code>	POST	Força recalcular agregação de células (job manual/admin)	Chave interna + ACL admin	Uso administrativo

📄 Payloads Detalhados

◆ Requisição Check-in (`/mapa/frequencia/checkin`)

```
{
  "user_id": "uuid",
  "location": {
    "lat": -23.561,
    "lng": -46.655
  }
}
```

```
{,
  "intencao": "encontro",
  "vib_score": 720,
  "timestamp": "2025-08-25T12:45:00Z"
}
```

◆ Resposta de Estado de Célula (`/mapa/frequencia/cell`)

```
{
  "geohash": "87a6d9",
  "estado": "expansao",
  "usuarios_ativos": 154,
  "vib_score_medio": 695,
  "tendencia": "alta",
  "last_update": "2025-08-25T12:40:00Z"
}
```

◆ Resposta de Histórico (`/mapa/frequencia/historico`)

```
{
  "geohash": "87a6d9",
  "intervalo": "7d",
  "pontos": [
    { "timestamp": "2025-08-19T10:00:00Z", "vib_score": 680 },
    { "timestamp": "2025-08-20T10:00:00Z", "vib_score": 710 },
    { "timestamp": "2025-08-21T10:00:00Z", "vib_score": 690 }
  ]
}
```

Segurança e Rate Limits

- Todos os endpoints exigem **JWT assinado** com claims específicos.
- Limites:
 - **Usuários Free** → 30 requests/minuto.
 - **Usuários Premium** → 120 requests/minuto.
- Logs de todas as chamadas armazenados no **Observability Service** (Prometheus + Grafana).

Versionamento

- **Atual:** `/api/v1/...`
- Próximos releases: `/api/v2/...` devem garantir **retrocompatibilidade mínima de 6 meses**.
- Alterações críticas documentadas no **Changelog Público para Devs**.

Observabilidade e Monitoramento

- Cada endpoint gera logs em **3 camadas**:

1. **Acesso** (quem chamou, quando, com qual token).
 2. **Dados** (payload resumido sem PII).
 3. **Resultado** (status code, tempo de resposta, fallback usado).
- Alertas automáticos:
 - Se a latência média > 500ms em 5 minutos → alerta para DevOps.
 - Se mais de 5% das chamadas falham → fallback ativo.

← END Fechamento da Camada 04

“As APIs são o sangue que irriga o Mapa. Contratos claros, versionamento sólido e payloads limpos garantem que a visão do pulso coletivo nunca colapse em ruído técnico.”

CAMADA 05 — ARQUITETURA DE BANCO & ESTRATÉGIA DE PRÉ-AGREGAÇÃO (H3)

Objetivo

Projetar a base de dados e o pipeline de **pré-agregação geográfica** que permitem ao Mapa de Frequências escalar com baixa latência e custo controlado, combinando:

- **PostgreSQL + PostGIS** para dados relacionais/geo e histórico agregável.
- **Firestore** para **tempo real individual** (presença do usuário).
- **Redis** para **cache** de zonas agregadas (leitura ultra-rápida).
- **H3** (grade hexagonal) para agregação espacial por células.

1) Topologia de Dados (alto nível)

```
[Mobile/Web]
└─(JWT)→ API Gateway
    └─ Privacy Filter (Zonas de Silêncio + Geofuzzing)
    └─ Firestore (user_presence_rt) ← tempo real individual
    └─ Kafka/PubSub (presence.v1, feedback.v1)
        └─ Query → Redis (zones_state_cache) → fallback → PostgreSQL (zones_state)

[Aggregators]
└─ Stream Proc (dedupe/window)
└─ Cron Aggregator (a cada 5 min, janela 10 min)
    → calcula métricas por H3
    → grava Redis (quente) + PostgreSQL (persistência/histórico)
```

2) Geohashing (H3) — parâmetros e uso

- **Resolução padrão:**
 - `res=7` (cidade / grandes bairros) — **default** para overview.
 - `res=8` (bairros) — quando o zoom se aproxima.
 - `res=9` (micro-zonas) — zoom alto em áreas densas.

- **Chave primária espacial:** `h3_index` (string).
- **Conversões:**
 - Ponto (lat/lon) → `h3_index` (resX).
 - Boundaries (hex para polígono) via função H3 → GeoJSON para PostGIS.

Regra de ajuste dinâmico: limitar retorno a ≤ 800 células por requisição de viewport (paginar se exceder).

3) Camadas de Persistência

3.1 Firestore (tempo real individual)

- **Coleção** `user_presence_rt`
 - `user_id (string)`, `ts (int)`, `freq_current (int 0..1000)`, `h3_res7 (string)`, `intent (string)`, `expires_at (int)`.
- **TTL** ~ 60 min; 1 update/min por usuário (rate limit).

3.2 Redis (cache quente de zonas)

- **Key:** `zone:{res}:{h3}` → JSON `{ vib_score, density, state, last_update }`
- **TTL:** 10 min; **write-through** a cada agregação.
- **Estratégia de partição:** hash por `h3` (uniformizar distribuição).

3.3 PostgreSQL + PostGIS (persistência e histórico)

Tabelas principais:

- `zones_state` (snapshot atual por célula e resolução)
- `zones_state_hist_yymm` (particionado por mês)
- `places` / `places_state` (parceiros/eventos)
- `transitions_log` (mudanças de estado por célula)
- `aggregator_runs` (telemetria dos jobs)
- `silence_zones` (polígonos privados do usuário, criptografados)

DDL (exemplo resumido):

```
-- Extensões
CREATE EXTENSION IF NOT EXISTS postgis;
CREATE EXTENSION IF NOT EXISTS btree_gin;

-- Snapshot atual
CREATE TABLE zones_state (
  res      smallint NOT NULL,      -- 7, 8, 9
  h3_index text      NOT NULL,
  vib_score integer  NOT NULL CHECK (vib_score BETWEEN 0 AND 1000),
  density  integer  NOT NULL DEFAULT 0,
  state    text     NOT NULL CHECK (state IN ('PICO','EXPANSAO','TRANSICAO','COLAPSO')),
  last_update timestamptz NOT NULL,
  geom      geometry(Polygon, 4326) NOT NULL, -- boundary do hex
  PRIMARY KEY (res, h3_index)
);

CREATE INDEX idx_zs_state ON zones_state(state);
CREATE INDEX idx_zs_gist ON zones_state USING GIST (geom);
```

```

-- Histórico particionado mensalmente
CREATE TABLE zones_state_hist (
  res      smallint,
  h3_index text,
  vib_score integer,
  density  integer,
  state    text,
  ts_bucket timestampz, -- ex: início de janela 5-min
  geom     geometry(Polygon, 4326)
) PARTITION BY RANGE (ts_bucket);

-- Ex.: criar partição mensal
-- CREATE TABLE zones_state_hist_2025_08 PARTITION OF zones_state_hist
-- FOR VALUES FROM ('2025-08-01') TO ('2025-09-01');

-- Log de transições de estado
CREATE TABLE transitions_log (
  res      smallint,
  h3_index text,
  state_from text,
  state_to  text,
  changed_at timestampz,
  vib_score integer,
  density  integer
);

-- Zonas de Silêncio (criptografadas)
CREATE TABLE silence_zones (
  user_id  uuid NOT NULL,
  label    text NOT NULL, -- casa, trabalho...
  geom     geometry(Polygon, 4326) NOT NULL, -- criptografado ou envelope cifrado
  created_at timestampz NOT NULL DEFAULT now(),
  PRIMARY KEY (user_id, label)
);
CREATE INDEX idx_silence_gist ON silence_zones USING GIST (geom);

```

4) Estratégia de Pré-Agregação (jobs)

- **Janela deslizante:** 10 min; **execução:** a cada 5 min.
- **Fórmula robusta:** média aparada (**trim 10%**) + **time-decay** ($\tau=20$ min) + ganhos (densidade/evento) – penalidades (toxicity).
- **Persistência dupla:** grava **Redis** (leitura rápida) e **PostgreSQL** (histórico + auditoria).
- **Publicação:** topic `zone_state.v1` para IA, painéis e notificações.

Operação dos jobs:

- **Stream Processor:** faz **dedupe** por `(user_id, floor(ts/60s))`, agrupa por `h3`, alimenta staging.
- **Cron Aggregator:** lê staging/FireStore/Kafka, calcula métricas, atualiza caches e histórico.
- **Compensation Worker:** reprocessa células afetadas por fraude (rollback) em até **< 2 min (p95)**.

5) Consultas — caminho de leitura

1. **Frontend** → `/api/map/zones?bbox=...&res=...`
2. API tenta **Redis** (hit desejado > 85%).
3. **Cache miss** → consulta **PostgreSQL** (última agregação válida) e **preenche** Redis.
4. **Garantia de frescor**: `updated_at` no payload + `Cache-Control: max-age=60`.

Filtro espacial (PostGIS):

```
SELECT res, h3_index, vib_score, density, state, last_update
FROM zones_state
WHERE res = 7
AND ST_Intersects(geom, ST_MakeEnvelope(:minLon, :minLat, :maxLon, :maxLat, 4326));
```

6) Índices, Particionamento & Performance

- `zones_state_hist`: particionado por **mês** (range) + índice GIST em `geom`.
- `zones_state`: PK (`res, h3_index`) + GIST em `geom` para filtros por viewport.
- **Manutenção**: `VACUUM (ANALYZE)` diário; `REINDEX` mensal; autovacuum tunado.
- **Rollups**: tarefas diárias para `zone_daily` (médias e picos) e mensais para `zone_monthly` (tendências).

7) Consistência & SLOs

- **Modelo**: *eventual consistency* para zonas agregadas; **tempo real** somente para presença individual.
- **SLOs principais**:
 - Staleness zonas: **≤ 10 min** (p95).
 - Latência `/zones`: **< 180 ms** (p95) com cache; **< 700 ms** (p95) fallback DB.
 - Reprocessamento antifraude (rollback): **< 2 min** (p95) por célula.

8) Privacidade & Segurança (dados)

- **Zonas de Silêncio**: filtro antes da ingestão — nada é gravado quando dentro do polígono.
- **Geofuzzing Dinâmico**: maior em áreas residenciais/baixa densidade (150–300 m), menor em áreas públicas (30–70 m).
- **Criptografia**:
 - Em repouso: TDE/Cloud KMS.
 - Em trânsito: TLS 1.2+.
- **Segregação PII**: `user_id` nunca aparece nas tabelas de zonas agregadas.
- **Logs**: sem PII; trace por `request_id` / `trace_id` (OpenTelemetry).

9) Custos & Governança

- **Redis**: TTL de 10 min + eviction LRU por **região**; sharding horizontal.
- **PostgreSQL**: retenção **90 dias** para `zones_state_hist` (arquivar em S3/BigQuery).
- **Downsampling**: reduzir resolução (`res=7`) quando viewport cobrir área muito ampla.

- **Cotas:** limitar chamadas de `/zones` por IP/usuário; pré-carregar células vizinhas (pequeno overfetch controlado).

10) Backfill, Migrações e Cold Start

- **Backfill histórico:** reprocessar `presence.v1` arquivado para compor `zones_state_hist`.
- **Migrações:** `CREATE ... IF NOT EXISTS` ; `ALTER TABLE ... ADD COLUMN` com *feature flag* no agregador.
- **Cold Start (mapa vazio):** usar `baseline_zones` (heurísticas por uso do solo/POIs públicos) para colorir **neutro/leve** até ter dados suficientes.

11) Pseudocódigo — agregação & leitura

Agregador (5/10 min):

```
def aggregate(window=10, res=7, tau=20, trim=0.10):
    buckets = group_by_h3(load_presence(window), res)
    out = []
    for h3_id, points in buckets.items():
        w = [exp(-(now()-p.ts)/tau) for p in points]
        freq = [p.freq for p in points]
        freq_t, w_t = trim_symmetric(freq, w, ratio=trim) # remove 10% extremos
        vib = weighted_mean(freq_t, w_t)
        vib += density_boost(len(points), res)
        vib += event_boost(h3_id, now())
        vib -= toxicity_penalty(h3_id, window)
        state = map_to_state(vib, len(points), res)
        out.append(record(h3_id, vib, len(points), state))
    write_redis(out); write_pg(out); publish('zone_state.v1', out)
```

Leitura (API `/zones`):

```
def get_zones(bbox, res):
    keys = h3_cover(bbox, res)[:800]
    data = redis_mget(keys)
    missing = [k for k,v in zip(keys, data) if v is None]
    if missing:
        db_rows = query_pg_by_bbox(bbox, res)
        for r in db_rows: redis_set(r.key, r, ttl=600)
        data = merge(data, db_rows)
    return data, updated_at=min(last_update(data))
```

12) Observabilidade (mínimo obrigatório)

- **Métricas (Prometheus):**
 - `zones_cache_hit_ratio` (alvo > 0.85)
 - `aggregator_job_duration_seconds` (p95 < 60s)
 - `api_zones_latency_ms` (p95 < 180ms)
 - `rollback_latency_seconds` (p95 < 120s)

- `zones_staleness_seconds` (p95 < 600s)
- **Logs:** structured JSON; sem PII; correlação com `trace_id`.
- **Alertas:** cache hit < 60% por 10 min; staleness > 15 min; latência p95 > alvo.

✓ Fechamento

Esta arquitetura garante **mapa fluido para o usuário, custos previsíveis, privacidade forte e dados auditáveis**. A pré-agregação via H3 é o pilar que transforma “tempo real impossível” em **tempo real percebido** — rápido, estável e escalável.

CAMADA 06 — MODELO DE CÁLCULO (VIB_SCORE) + CALIBRAÇÃO

Objetivo

Definir, de ponta a ponta, **como calcular o VIB_SCORE de uma célula H3** (estado coletivo) com **precisão, estabilidade e antifraude**, preservando desempenho e evitando oscilações visuais.

1) Entradas do Cálculo (por ponto “i”)

Cada ponto vem da presença/feedback do usuário dentro da janela (ex.: últimos 10 min).

Campo	Descrição	Faixa/Tipo
<code>freq_raw_i</code>	frequência individual normalizada (pipeline do usuário)	0..1000
<code>ts_i</code>	timestamp do evento	epoch
<code>trust_i</code>	confiança do ponto (GPS, device integrity, velocidade)	0.0..1.0
<code>emotion_i</code>	intensidade emocional (opcional)	-1..+1
<code>intent_i</code>	peso da intenção ativa (ex.: encontro, meditação)	0.5..1.2
<code>place_ctx_i</code>	afinidade com local/evento (in-loco, perímetro, remoto)	0.8..1.1

Obs.: `freq_raw_i` chega sem PII e já passou por Privacy Filter.

2) Pesos por ponto (time-decay + qualidade)

Para cada ponto, o **peso final** é a composição de tempo, confiança e contexto:

$$w_i = e^{-\Delta t_i \tau} \cdot \text{clamp}(\text{trust}_i, 0, 1) \cdot \text{clamp}(\text{intent}_i \cdot \text{place_ctx}_i, 0.5, 1.2) \cdot \text{contexto}$$

$$\text{w_i} = e^{-\tau \Delta t_i} \cdot \text{clamp}(\text{trust}_i, 0, 1) \cdot \text{clamp}(\text{intent}_i \cdot \text{place_ctx}_i, 0.5, 1.2) \cdot \text{contexto}$$

$w_i =$

$e^{-\tau \Delta t_i}$ time-decay

$\text{clamp}(\text{trust}_i, 0, 1)$ qualidade

$\text{clamp}(\text{intent}_i \cdot \text{place_ctx}_i, 0.5, 1.2)$ contexto

- $\Delta t_i = \text{now} - \text{ts}_i$ $\Delta t_i = \text{now} - \text{ts}_i$ (minutos)
- τ (default **20 min**) — decai rápido, privilegiando o **agora**.

3) Robustez estatística (trimmed mean)

Para evitar distorções por extremos:

1. Ordene os pares $(freq_raw_i, w_i)$ pela `freq_raw_i`.
2. **Remova simetricamente 10% dos menores e 10% dos maiores** ($trim = 0.10$).
3. Aplique **média ponderada** nos restantes:

$$\bar{f} = \frac{\sum w_i \cdot freq_raw_i}{\sum w_i} \quad \text{(após trimming)}$$

$$\bar{f} = \sum w_i \cdot freq_raw_i \quad \text{(após trimming)}$$

Fallback: se houver ≤ 10 pontos, use winsorization (aparar substituindo extremos pelos quantis 10%/90%) em vez de remover.

4) Boosts e Penalidades (célula H3)

A célula agrega mais sinais além da média robusta:

4.1 Density Boost (evita “tudo igual”)

$$density_boost = \alpha_{res} \cdot \log(1 + N)$$

$$density_boost = \alpha_{res} \cdot \log(1 + N)$$

- NNN = densidade de pontos válidos na janela
- α_{res} depende da resolução H3 (ex.: **res7=18, res8=12, res9=8**)
- **Cap:** máx +40 no score final

4.2 Event Boost (atividades/Locais Parceiros)

$$event_boost = \min(\beta_{tipo} + \beta_{tier}, 25)$$

$$event_boost = \min(\beta_{tipo} + \beta_{tier}, 25)$$

- Ex.: **evento oficial** $\beta_{tipo}=15$; **tier Premium** $\beta_{tier}=5$

4.3 Toxicity Penalty (moderação/denúncias)

$$toxicity_penalty = \min\left(\gamma \cdot \sum s_j \cdot e^{-\Delta t_j / \tau_{tox}}, 120\right)$$

$$toxicity_penalty = \min(\gamma \cdot \sum s_j \cdot e^{-\tau_{tox} \Delta t_j}, 120)$$

- s_j = severidade da ocorrência (ex.: 1..5)
- γ default **8**; τ_{tox} **60 min**

5) VIB_SCORE da célula (0..1000)

$$VIB_SCORE_zone = \text{clamp}(\bar{f} + density_boost + event_boost - toxicity_penalty, 0, 1000)$$

$$VIB_SCORE_zone = \text{clamp}(\bar{f} + density_boost + event_boost - toxicity_penalty, 0, 1000)$$

Suavização espacial opcional (anti-flicker):

Aplicar **suavização por vizinhança H3 (k-ring)** após o cálculo:

- Centro: **0.60**, anel 1: **0.25**, anel 2: **0.15** (somente se > 50% dos vizinhos válidos).

6) Mapeamento de Estado + Histerese

Para evitar “pisca” entre rótulos, usamos **limiares com histerese**:

Estado	Entrada (subir)	Saída (cair)
PICO	≥ 720 e $N \geq \text{limiar_dens}$	< 690
EXPANSÃO	≥ 520	< 490
TRANSIÇÃO	≥ 320	< 290
COLAPSO	< 320	—

- **Tendência:** ao decidir **EXPANSÃO**, se $f \sim \text{bar ff}$ crescente por 2 janelas → favorece promoção.
- **limiar_dens** calibrado por **resolução** e **cidade** (vide tabela).

7) Antifraude e Rollback

- Sinais suspeitos → `presence.flag.v1`.
- **Compensation Worker** remove contribuições do `user_id/intervalo` das células afetadas e **recalcula** (mesma janela).
- **SLO:** consistência eventual $< 2 \text{ min (p95)}$.

8) Tabelas de Calibração (PostgreSQL)

8.1 `vib_decay_params`

chave	valor	obs
<code>tau_minutes</code>	20	time-decay presença
<code>tau_tox_minutes</code>	60	decaimento toxidade
<code>trim_ratio</code>	0.10	corte simétrico

8.2 `vib_density_params`

res	alpha	dens_limiar
7	18	25
8	12	15
9	8	8

8.3 `vib_event_weights`

tipo	tier	beta_tipo	beta_tier
oficial	Premium	15	5
parceiro	Essencial	8	0
comunitário	—	5	0

8.4 `vib_toxicity_params`

gamma	cap
8	120

8.5 `vib_state_thresholds` (histerese)

estado	up	down
PICO	720	690
EXPANSAO	520	490
TRANSICAO	320	290

Feature flags: permitir alterar α , β , γ , τ , trims por cidade/país.

9) Pseudocódigo (agregação por célula)

```
def aggregate_cell(points, now, params):
    # Pesos
    weighted = []
    for p in points:
        w_time = exp(-(now - p.ts) / params.tau)
        w_ctx = clamp(p.trust * p.intent * p.place_ctx, 0.5, 1.2)
        weighted.append((p.freq_raw, w_time * w_ctx))

    # Trimmed mean (ou winsor se poucos pontos)
    freq_t, w_t = trim_or_winsor(weighted, ratio=params.trim_ratio)
    f_bar = weighted_mean(freq_t, w_t)

    # Boosts/penalidades
    dens = len(freq_t)
    vib = f_bar \
        + density_boost(dens, params.alpha_res, params.res) \
        + event_boost(params.event_ctx) \
        - toxicity_penalty(params.tox_ctx, params.tau_tox)

    vib = clamp(vib, 0, 1000)

    # Suavização espacial (opcional)
    vib = spatial_smooth(vib, neighbors=params.neighbors)

    # Estado com histerese
    state = hysteresis_map(vib, dens, params.thresholds, prev_state=params.prev_state)
    return vib, state
```

10) Exemplo Numérico (resumo)

- 60 pontos na janela; após trim 10% → **48 pontos**.
- $f^- = 640$ / $\bar{f} = 640$ / $f^+ = 640$ (ponderado por time-decay/qualidade).
- `density_boost(res7, N=48) ≈ 18 · log(49) ≈ 70` (cap aplica +40 → **+40**).
- Evento parceiro Premium: **+13**.
- Penalidade tox: **-16**.
- **VIB_SCORE = 640 + 40 + 13 - 16 = 677** → **EXPANSÃO** (tendência ↑ confirma).

11) Qualidade, Testes e Sinais de Alerta

Testes unitários recomendados:

- Trim/Winsor sob distribuições com outliers.
- Decaimento temporal (alterar τ /tau, validar responsividade).
- Histerese (não oscilar em bordo).
- Smoothing: não “vazar” demais para vizinhos.

- Rollback: recalcular célula e vizinhos impactados.

Métricas a monitorar:

- % de células estáveis vs. em transição (saúde do mapa).
- Oscilação de estado por célula (picos suspeitos).
- Impacto médio de rollback (detectar fraudes em cluster).
- Latência de agregação e staleness do cache.

✓ Fechamento

Este modelo equilibra **fidelidade ao agora** (time-decay), **robustez estatística** (trimmed mean), **contexto vivo** (density/eventos) e **segurança** (penalidade de toxicidade + rollback), entregando um **VIB_SCORE** estável, **audível e performático** — pronto para escalar sem surpresas.

📍 CAMADA 07 — VISUALIZAÇÃO (Individual x Coletivo)

🎯 Objetivo da Camada

Definir como o **Mapa de Frequências** deve ser **renderizado no frontend** (mobile/web), diferenciando a **visualização individual (do próprio usuário)** da **visualização coletiva (global ou regional)**. Garantir que os dados sejam apresentados de forma fluida, sem "flicker", e que a experiência seja intuitiva, clara e escalável.

◆ 1. Estrutura da Visualização

• Mapa Individual (Self-View)

- Mostra **apenas a frequência pessoal** do usuário.
- Inclui:
 - **Ícone do usuário** com aura dinâmica (cor = estado vibracional).
 - Histórico imediato de check-ins vibracionais (últimas 24h).
 - Botão "Expandir para Coletivo" (abre o modo global).
- Privacidade: apenas o próprio usuário vê esse mapa.

• Mapa Coletivo (Global/Regional)

- Renderiza **clusters geográficos** (via geohashing/H3).
- Exibe estados agregados de frequência por **zonas/células**.
- Ícones ou manchas coloridas indicam os estados:
 - **Pico** → vermelho intenso com pulso.
 - **Expansão** → verde-azulado em ondas.
 - **Transição** → amarelo-laranja intermitente.
 - **Colapso** → cinza/roxo opaco.
- Usuários individuais só aparecem quando há **check-in ativo + permissão**.

◆ 2. Anti-Flicker e Suavização





- Evitar que o mapa "pisque" a cada atualização.
- Implementar **buffer temporal (ex.: 5s)** antes de aplicar mudanças de cor.

- Animações de transição suaves (fade-in/out, pulsação progressiva).
- Usar **pré-agregação de dados (Camada 06)** para alimentar as cores das zonas.

◆ 3. Diferenciação Visual (Individual x Coletivo)

- **Individual** →
 - Ícone circular com aura pessoal.
 - Score numérico visível ao tocar no avatar.
 - Mensagem da IA (Aurah Kosmos) com feedback.
- **Coletivo** →
 - Zonas coloridas, opacas de longe e pulsantes ao dar zoom.
 - Legenda fixa no canto da tela com os estados vibracionais.
 - Possibilidade de filtrar: *Usuários* / *Locais Parceiros* / *Eventos*.

◆ 4. Legenda e Estados

Estado	Cor	Animação	Significado Técnico
Pico	 Vermelho	Pulso rápido	Alta concentração de usuários + alta frequência
Expansão	 Verde-azulado	Onda suave	Energia positiva crescente
Transição	 Amarelo/laranja	Intermitente	Estado instável
Colapso	 Cinza/roxo	Aura opaca	Baixa frequência coletiva

◆ 5. Regras Técnicas de Renderização

- **API usada:** `/api/map/frequency/:cell_id` (pré-agregado).
- **Tempo de atualização:**
 - Clusters coletivos → 5 min (jobs em background).
 - Presença individual → real-time (Firestore).
- **Fallback em cidades sem dados:**
 - Exibir **camada base neutra** (heurística).
 - Ex.: parques → neutro positivo, áreas residenciais → neutro baixo.

◆ 6. Segurança e Privacidade

- Sempre aplicar **geofuzzing dinâmico** em usuários individuais.
- Respeitar **Zonas de Silêncio** (casa/trabalho) → não exibir nunca.
- Permitir ao usuário ativar/desativar **visibilidade individual no mapa global**.

◆ 7. Fluxo de Integração

[Check-in Vibracional]

- ↓ Firestore (tempo real)
- ↓ API Aggregation (clusters 5 min)
- ↓ Frontend
 - ↳ Individual View → Avatar pessoal + aura

↳ Coletivo View → Clusters + estados coloridos

✨ Fechamento da Camada

"Aqui o invisível toma forma. O mapa não apenas mostra locais — ele traduz o estado vivo da comunidade. Cada ponto é um pulso, cada cluster uma batida coletiva. O usuário vê a si mesmo e ao todo, em perfeita harmonia visual."

🗺️ CAMADA 08 — MODELO DE DADOS PRINCIPAL (PostgreSQL + Firestore)

🎯 Objetivo da Camada

Definir de forma executável a arquitetura de dados híbrida do **Mapa de Frequências**, separando **dados persistentes e relacionais** (PostgreSQL + PostGIS) de **estados dinâmicos em tempo real** (Firestore). Esta camada garante **escalabilidade, velocidade e consistência** do ecossistema.

🗄️ Estrutura Relacional (PostgreSQL + PostGIS)

Tabela **users** (persistência de usuários no mapa)

Campo	Tipo	Descrição
id_user	UUID	Identificador único
username	VARCHAR(50)	Nome de exibição
verified	BOOLEAN	Status de verificação DUC/DCO
created_at	TIMESTAMP	Data de entrada no sistema

Tabela **checkins** (registros vibracionais)

Campo	Tipo	Descrição
id_checkin	UUID	Identificador único
id_user	UUID (FK)	Usuário que fez o check-in
geom_location	GEOGRAPHY(POINT)	Coordenada geográfica (PostGIS)
vib_score	FLOAT	Frequência vibracional no momento
intencao	VARCHAR(100)	Intenção declarada (evento, social, introspectivo, etc.)
created_at	TIMESTAMP	Data/hora do check-in

Tabela **locais** (pontos fixos do mapa)

Campo	Tipo	Descrição
id_local	UUID	Identificador único
nome	VARCHAR(100)	Nome do local
geom_area	GEOGRAPHY(POLYGON)	Área delimitada (PostGIS)
estado_vibracional	VARCHAR(20)	Classificação atual (expansão, transição, colapso)
updated_at	TIMESTAMP	Última atualização

⚡ Estrutura em Tempo Real (Firestore)

Coleção: **active_users**

Campo	Tipo	Descrição
id_user	String	Identificador do usuário

Campo	Tipo	Descrição
latitude	Float	Latitude atual
longitude	Float	Longitude atual
vib_score	Number	Frequência vibracional instantânea
status	String	online / offline / invisível
last_seen	Timestamp	Última atualização

Coleção: `heatmap_cells` (pré-agregação por geohashing)

Campo	Tipo	Descrição
cell_id	String	ID da célula H3
avg_score	Number	Média vibracional da célula
density	Integer	Número de usuários ativos
estado	String	Estado vibracional agregado
updated_at	Timestamp	Última agregação

Sincronização SQL ↔ Firestore

- **Persistência:** todo check-in validado é gravado em `checkins` (SQL).
- **Tempo real:** Firestore mantém apenas **usuários ativos e células agregadas**.
- **Jobs de agregação:** a cada 5 min, um **worker** recalcula médias no SQL e atualiza Firestore → `heatmap_cells`.
- **Fallback:** se Firestore cair, o app consulta o SQL com cache de 10 min.

Pseudocódigo de Inserção + Sincronização

```
# Inserção de check-in no PostgreSQL
def registrar_checkin(id_user, lat, lon, vib_score, intencao):
    insert into checkins (id_user, geom_location, vib_score, intencao, created_at)
    values (id_user, ST_Point(lat, lon), vib_score, intencao, NOW())

# Atualiza Firestore (tempo real)
firestore.collection("active_users").doc(id_user).set({
    "latitude": lat,
    "longitude": lon,
    "vib_score": vib_score,
    "status": "online",
    "last_seen": now()
})
```

Regras Técnicas

- **GDPR/LGPD compliance:** check-ins **não podem ser alterados** (imutáveis), apenas marcados como inválidos.
- **Geofuzzing dinâmico** aplicado antes de gravar coordenadas.
- **Zonas de silêncio:** bloqueiam envio para Firestore (mas mantêm histórico no SQL, com flag `private_zone`).

✨ Fechamento da Camada

“Com a fusão entre **PostgreSQL/PostGIS** e **Firestore**, o Mapa de Frequências respira em dois ritmos:

⚡ rápido como o instante presente, 🗄 sólido como o registro histórico.

Esse design híbrido garante que cada pulso do mundo seja visível, mas também seguro, privado e imutável."

⚙ CAMADA 09 — ALGORITMO DE AGREGAÇÃO & PRÉ-CÁLCULO (H3 + TIME-DECAY)

🎯 Objetivo

Projetar o **motor de agregação** que transforma sinais individuais (presença/feedback) em **estados coletivos por células H3** com **baixa latência**, **robustez estatística** e **custos previsíveis**, usando:

- Geohashing (H3/S2)
- Janela deslizante + **time-decay** exponencial
- **Média aparada** (trimmed mean) com antifraude
- **Micro-batch/stream** com deduplicação e idempotência
- Rollup hierárquico (res9 → res8 → res7)
- Publicação em **Redis** (cache quente) + **PostgreSQL** (persistência)

1) Escopo de Entrada (events)

- `presence.v1` : {user_id, ts, lat, lon, freq_raw, intent, trust}
- `feedback.v1` : {user_id, ts, h3, emotion, intensity}
- `place_event.v1` : {place_id, type, start, end, tier}
- `presence.flag.v1` (antifraude): {user_id, ts_range, reason, impact_h3[]}

┃ Chave de dedupe: (user_id, floor(ts/60s)) — 1 atualização/min por usuário.

2) Estratégia de Janela e Watermarks

- **Janela deslizante**: 10 min | **Agregação**: a cada 5 min (micro-batch)
- **Watermark**: atraso tolerado de **2 min** (eventos fora desse atraso entram no próximo ciclo com peso temporal reduzido)
- **Late data policy**: reprocessar célula se gap ≤ 10 min; caso contrário, lançar para **backfill tópico** (assíncrono)

3) Particionamento & Distribuição

- **Kafka/PubSub**: `presence.v1` particionado por **prefixo H3 (res7)** → garante locality
- **Workers**: `N` pods consumindo partições (autoscaling por lag)
- **State store** (streaming): RocksDB (Flink) ou Bigtable/Dynamo (Beam/Spark) por **key = (res, h3)**
- **Idempotência**: compactação por chave de dedupe + `Idempotency-Key` nos writes

4) Pipeline Operacional (alto nível)

(1) Ingestão → (2) Privacy Filter (Zonas de Silêncio + Geofuzzing)
→ (3) H3 Mapper (res9, res8, res7) → (4) Dedupe/Windowing

- (5) Reduce por célula (trimmed mean + time-decay + boosts/penalidades)
- (6) Histerese + Estado → (7) Rollup (res9→res8→res7)
- (8) Persistência: Redis (quente) + PostgreSQL (hist/snapshot)
- (9) Publish: zone_state.v1 → assinantes (IA/PAINÉIS/Notificações)

5) Núcleo do Cálculo (resumo)

As fórmulas detalhadas estão na Camada 06. Aqui definimos como aplicá-las em produção.

5.1 Pesos por ponto (tempo + qualidade + contexto)

- $w_i = \exp(-(now - ts_i)/\tau) * \text{clamp}(\text{trust}_i, 0, 1) * \text{clamp}(\text{intent}_i * \text{place_ctx}_i, 0.5, 1.2)$
- τ (**tau**) padrão: 20 min (calibrável por cidade)

5.2 Robustez (trimmed mean)

- Ordenar por `freq_raw` e **aparar 10%** menor e maior (winsor se ≤ 10 pts)
- Média ponderada pelos `w_i` restantes

5.3 Boosts/Penalidades por célula

- $\text{density_boost} = \alpha_{\text{res}} * \log(1 + N)$ (cap +40)
- $\text{event_boost} = \min(\beta_{\text{tipo}} + \beta_{\text{tier}}, 25)$
- $\text{toxicity_penalty} = \min(\gamma * \sum s_j * \exp(-\Delta t / \tau_{\text{tox}}), 120)$

5.4 VIB_SCORE final (0..1000)

- $\text{VIB} = \text{clamp}(\hat{f} + \text{density_boost} + \text{event_boost} - \text{toxicity_penalty}, 0, 1000)$

5.5 Estado com histerese

- PICO / EXPANSÃO / TRANSIÇÃO / COLAPSO (limiares \uparrow/\downarrow) + tendência

6) Rollup Hierárquico (res9 → res8 → res7)

- Cálculo “fino” em **res9** (micro-áreas) quando **zoom alto** ou densidade alta
- Rollup dos **filhos**:
 - $\text{VIB_res8} = \text{trimmed_mean_weighted}(\text{children VIB_res9}, \text{weights}=\text{children density})$
 - $\text{VIB_res7} = \text{trimmed_mean_weighted}(\text{children VIB_res8}, \dots)$
- Estado** do pai é recalculado com **mesma histerese** (não herdar label “por voto”)
- Atalho**: se densidade baixa ($<$ limiar), agregue direto em res8/res7

7) Suavização Espacial (opcional, pós-cálculo)

- $\text{VIB}' = 0.60 * \text{VIB_center} + 0.25 * \text{avg}(k\text{-ring1}) + 0.15 * \text{avg}(k\text{-ring2})$
- Aplicar **apenas** se $\geq 50\%$ dos vizinhos válidos; nunca “inventar” dado inexistente
- Executar **depois** do estado com histerese para não mascarar oscilações

8) Antifraude & Rollback

- Detectado spoof/fraude → `presence.flag.v1`

2. Compensation Worker:

- Localiza contribuições (`user_id, ts_range`) nas células impactadas
- **Remove** pontos, **recalcula** com mesma janela (time-decay + trim)
- Atualiza Redis e PostgreSQL; registra em `transitions_log`

3. SLO: consistência eventual < 2 min (p95) após flag

9) Publicação & Persistência

- **Redis** (`zone:{res}:{h3}`): `{ vib, density, state, last_update }` TTL 10 min
- **PostgreSQL**:
 - `zones_state` (snapshot atual por res/h3)
 - `zones_state_hist_YYYY_MM` (particionado 5-min buckets)
 - `transitions_log` (mudanças de estado)
- **Tópico** `zone_state.v1`: payload "slim" para consumidores downstream

10) Pseudocódigo — Micro-batch Aggregator

```
def microbatch_aggregate(window=10, step=5, res_list=(9,8,7)):  
    # 1) Carrega eventos recentes (presence + feedback) com watermark  
    events = load_stream(window_minutes=window, watermark=2)  
  
    # 2) Mapeia pontos para H3 resoluções necessárias  
    by_res = {r: defaultdict(list) for r in res_list}  
    for ev in events:  
        if is_silence_zone(ev) or !privacy_ok(ev):  
            continue  
        for r in res_list:  
            h3 = h3_index(ev.lat, ev.lon, r)  
            by_res[r][h3].append(ev)  
  
    # 3) Agrega do fino para o amplo  
    results = {r: {} for r in res_list}  
    # res9 (ou res8 se escolhido como fino)  
    for h3, pts in by_res[9].items():  
        vib, dens, state = compute_cell(pts) # trimmed mean + time-decay + boosts/pen  
        results[9][h3] = (vib, dens, state)  
  
    # Rollup res8 ← filhos res9  
    for h3_parent, children in group_children(results[9], parent_res=8).items():  
        vib, dens, state = rollup(children) # média robusta ponderada por dens  
        results[8][h3_parent] = (vib, dens, state)  
  
    # Rollup res7 ← filhos res8  
    for h3_parent, children in group_children(results[8], parent_res=7).items():  
        vib, dens, state = rollup(children)  
        results[7][h3_parent] = (vib, dens, state)  
  
    # 4) Suavização espacial (opcional) e histerese final  
    for r in res_list:
```

```

for h3 in results[r]:
    vib, dens, state = results[r][h3]
    vib = spatial_smooth_if_applicable(r, h3, vib)
    state = hysteresis(prev_state(r,h3), vib, dens)
    results[r][h3] = (vib, dens, state)

```

```

# 5) Persistência e publicação
write_redis(results)
write_postgres_snap_and_hist(results)
publish_zone_state(results)

```

compute_cell(pts) implementa Camada 06 (pesos, trim, boosts/penalidades, histerese).

11) Mecanismos de Robustez

- **Backpressure:** autoscaling por **lag** no tópico e por **latência** do job
- **Load-shedding:** reduzir resolução (usar res7) em **zoom muito amplo** ou quando `zones_cache_hit < alvo`
- **Retry & DLQ:** eventos inválidos vão para **Dead Letter Queue** com amostras para diagnóstico
- **Exactly-once (prático):** at-least-once + idempotência de writes (ETag/Upsert por PK `(res,h3)`)

12) Configuração (feature flags)

Flag	Default	Descrição
<code>AGG_WINDOW_MIN</code>	10	Tamanho da janela deslizando
<code>AGG_STEP_MIN</code>	5	Frequência de execução
<code>TAU_MIN</code>	20	Decaimento temporal
<code>TRIM_RATIO</code>	0.10	Corte simétrico para trimmed mean
<code>SMOOTH_NEIGHBORS</code>	on	Suavização k-ring
<code>ROLLUP_MODE</code>	<code>res9→8→7</code>	Hierarquia ativa
<code>DENSITY_CAP</code>	40	Máximo de density boost
<code>EVENT_CAP</code>	25	Máximo de event boost

Flags por região/cidade via tabela de calibração (Camada 06).

13) Observabilidade & SLOs (voltado ao job)

- **Métricas:**
 - `agg_job_duration_seconds{res}` (p95 < 60s)
 - `agg_staleness_seconds` (p95 < 600s)
 - `zones_cache_hit_ratio` (alvo > 0.85)
 - `rollback_latency_seconds` (p95 < 120s)
 - `late_events_ratio` (alerta se > 5%)
- **Alertas:**
 - Staleness > 15 min (crit)
 - Lag de partição > 5 min (crit)

- Hit ratio < 60% por 10 min (warn)

14) Testes & Qualidade

- **Unitários:** trim/winsor, time-decay, histerese, rollup, smoothing
- **Property-based:** estabilidade sob outliers e bursts
- **Carga:** 1M eventos/5 min; validar latência e custos
- **E2E:** fraude → flag → rollback < 2 min; confirmar `transitions_log`
- **Canário:** nova calibração por cidade entra em **10% das células** antes do rollout total

15) Custos & Controles

- **Redis:** TTL 10 min, **eviction LRU por região**
- **PostgreSQL:** partição mensal; retenção hist 90 dias (arquivar frio)
- **Streaming:** autoscaling horizontal; compressão nos tópicos; compactação por chave
- **Fallback:** se agregador falhar, API devolve **último snapshot** com `stale=true`

✓ Fechamento

Este algoritmo converte um “tempo real inviável” em **tempo real percebido**, com micro-batch inteligente, estatística robusta, privacidade ativa e antifraude com compensação. O resultado é um **mapa fluido, escalável e auditável**, pronto para milhões de usuários.



CAMADA 10 — PRIVACIDADE & SEGURANÇA

Geofuzzing Dinâmico • Zonas de Silêncio • Consentimentos • K-Safety • Auditoria

🎯 Objetivo

Garantir que o Mapa de Frequências opere com **privacidade por design** e **segurança por padrão**, preservando utilidade do produto e mitigando reidentificação. Esta camada formaliza políticas, algoritmos e contratos técnicos.

1) Modelos de Risco & Princípios

- **Minimização de dados:** coletar apenas o necessário para cálculo/agregação.
- **Separação de papéis:** presença individual (tempo real) ≠ agregados de zonas (pré-cálculo).
- **Desidentificação forte:** nada de PII em agregados; `user_id` nunca vai para `zones_state`.
- **Defesa em profundidade:** criptografia, controle de acesso, limitação de consulta, rate-limit, auditoria.
- **K-safety de renderização:** nenhuma célula é exibida se `k` usuários ativos < limiar.

2) Políticas de Consentimento (LGPD/GDPR)

Consentimento	Escopo	Default	Efeito Técnico
Mapa Individual	Ver sua própria presença/frequência	ON	Leitura do doc <code>user_presence_rt/{user}</code>
Mapa Global	Contribuir anonimamente para agregados	ON	Envia eventos <code>presence.v1</code> após Privacy Filter

Consentimento	Escopo	Default	Efeito Técnico
Compartilhar presença pública	Aparecer como ponto individual em mapas coletivos	OFF	Só exibe avatar/marker se ON
Histórico	Guardar check-ins para IA/relatórios	ON	Persiste em <code>checkins</code> (SQL) com flags
Zonas de Silêncio	Nunca transmitir em locais sensíveis	USER-DEFINED	Bloqueio no Privacy Filter (on-device/server)

Ledger de consentimento:

`consent_ledger` (PostgreSQL) com `user_id`, `purpose`, `state`, `updated_at`. Usado em todos os pipelines (enforced em gateway).

3) Zonas de Silêncio (Silence Zones)

- Definidas pelo usuário (casa, trabalho, outros polígonos).
- Enforcement:**
 - On-device:** checagem local (preferencial).
 - Server:** `Privacy Filter` antes de publicar `presence.v1`.
- Armazenamento:** tabela `silence_zones` com **geom cifrado** (KMS), índice GIST para interseção.

Contra-exfiltração: nenhum evento é emitido quando `point ∈ silence_zone`.

4) Geofuzzing Dinâmico (anti-reidentificação)

Objetivo: distorcer coordenadas **proporcionalmente à densidade/população do solo** e contexto.

4.1 Raio de Distorção

$$r = \text{clamp}(r_{\min}, r_{\max}, \alpha \cdot \text{density_cell} + 1 + \beta \cdot \text{risk_residential})$$

$$r = \text{clamp}(r_{\min}, r_{\max}, \alpha \cdot \frac{1}{\sqrt{\text{density_cell} + 1}} + \beta \cdot \text{risk_residential})$$

$$r = \text{clamp}(r_{\min}, r_{\max}, \alpha \cdot \text{density_cell} + 1$$

$$+ \beta \cdot \text{risk_residential})$$

- `density_cell` = densidade estimada (H3 res8) da região.
- `risk_residential` $\in \{0,1\}$ (mapa de uso do solo).
- Sugerido: `r_min=30m` (áreas públicas densas), `r_max=300m` (residenciais).

4.2 Amostragem & Estabilidade

- Amostra **ruído 2D isotrópico** (ex.: Laplace/gaussiano truncado) com **seed por sessão**:
 - `seed = HMAC_SHA256(device_id || session_id, K_priv)`
 - Mantém **coerência na sessão** (evita média inversa), muda entre sessões.

4.3 Magnetização a Locais Públicos

- “Snap” para **polígonos públicos** próximos (praças/ruas/POIs), evitando queda dentro de residências.
- Regras: raio de snap $\leq r/2$, prioridade vias/praças.

Pseudocódigo (simplificado):

```
def fuzz(point, density, is_residential, seed):
    r = clamp(R_MIN, R_MAX, A / sqrt(density+1) + B*is_residential)
    noise = sample_2d_noise(seed, radius=r)    # estável por sessão
    p1 = point + noise
```

```

if is_residential_area(p1):
    p2 = snap_to_public_polygon(p1, max_dist=r/2) or p1
    return p2
return p1

```

5) K-Safety & Regras de Renderização

- **k mínimo por célula para exibir agregados:**
 - `k_res7 = 15` , `k_res8 = 10` , `k_res9 = 6` (calibrável por cidade).
- **Suppress pequeno-n:** se `N < k` , retornar **estado neutro** com `suppressed=true` e **sem valores numéricos**.
- **Bucketização:** quando `N` próximo de `k` , retornar faixas (ex.: 320–360) em vez de valor exato.
- **Janelas temporais:** janela ≥ 10 min (Camada 09) para diluir reidentificação temporal.

Contrato API (exemplo supressão):

```

{
  "h3": "871fb4ac...",
  "suppressed": true,
  "state": "NEUTRO",
  "reason": "LOW_K",
  "last_update": "2025-08-25T18:10:00Z"
}

```

6) Privacy Filter (gateway)

Executado **antes** de Kafka/Firestore.

Checagens:

1. Consent ledger → permissões do usuário.
2. Silence zones → aborta evento.
3. Trust score do device (SafetyNet/Integrity/Attestation).
4. Geofuzzing Dinâmico (se necessário).
5. Redução de precisão em áreas sensíveis (5–6 casas decimais → 4).

Pseudocódigo:

```

def privacy_filter(event, user_ctx):
    if not consent_allows(event.purpose, user_ctx): return DROP
    if inside_silence_zone(event.lat, event.lon, user_ctx): return DROP
    if !device_integrity_ok(event.device): return QUARANTINE
    lat, lon = maybe_fuzz(event.lat, event.lon, user_ctx)
    return EMIT(event.with_coords(lat, lon))

```

7) Segurança de Plataforma

7.1 Criptografia

- **Em trânsito:** TLS 1.2+ com PFS (ECDHE).
- **Em repouso:** TDE/KMS (AES-256) em Postgres; **Field-level encryption** para `silence_zones.geom`.
- **KMS/HSM:** chaves com rotação **90 dias**; acesso via **service accounts**.

7.2 Acesso & Autorização

- **Auth:** JWT (Auth0/Firebase Auth) com *scopes* (ex.: `map:read`, `map:write`).
- **RBAC/ABAC:** OPA (Open Policy Agent) com políticas por **papel + propósito**.
- **Segregação de dados:** os serviços que calculam zonas **nunca** recebem PII; apenas IDs efêmeros.

7.3 Hardening & Proteções

- **Rate-limit/WAF:** por IP e por usuário.
- **Anomalia de consulta:** bloqueio se viewport pedir > 800 células repetidamente.
- **DLP:** regras que impedem exportação de coordenadas brutas de Firestore/SQL.
- **DDOS:** CDN + autoscaling + circuit breakers.

8) Auditoria, Logs & Observabilidade

- **Logs estruturados** (sem PII) com `trace_id`, `request_id`, `user_scope`, `decision` (permit/suppress/drop).
- **Auditoria imutável** (WORM/S3 + carimbo de tempo) para decisões de privacidade.
- **Métricas chave:**
 - `privacy_suppression_rate`
 - `low_k_requests_rate`
 - `silence_zone_hits`
 - `fuzz_radius_avg`
 - `reidentification_risk_score` (heurístico)
- **Alertas:**
 - `suppression_rate` < 1% em áreas residenciais (sinal de fuga de política).
 - exportações anômalas / picos de leituras por célula.

9) Retenção & Direitos do Titular (LGPD/GDPR)

Dado	Retenção	Ação do Titular
<code>user_presence_rt</code> (tempo real)	TTL 60 min	Não aplicável (ephemeral)
<code>checkins</code> (SQL)	12 meses (default)	Exclusão/portabilidade via <code>/dpo/export</code>
<code>zones_state_hist</code>	90 dias (agregado)	Anônimo (não sujeita a exclusão individual)
<code>consent_ledger</code>	Enquanto ativo + 24m	Histórico de consentimentos
<code>silence_zones</code>	Até remoção do usuário	CRUD pelo app

Exclusão: *hard delete* de `checkins` por `user_id` com **compensation** (recalcular agregados onde aplicável).

10) Regras de UI/UX (transparência)

- Painel de privacidade com:
 - **Toggles** de consentimento.
 - **Mapa de Zonas de Silêncio** (CRUD simples).

- **Explicação do geofuzzing** com raio aproximado.
- **Exportar meus dados e Apagar meus dados** (com SLA).
- Banner de primeira execução explicando Mapa Individual x Global.

11) Testes, Conformidade & Red Team

- **Testes de privacidade:**
 - Reidentificação por agregados (ataque de diferenciação entre k e $k-1$).
 - Ataque de triangulação temporal (múltiplas consultas).
 - Simulação de "home inference" em baixa densidade.
- **Pentests periódicos; DPIA/RIPD** formal para LGPD.
- **Chaos privacy drills:** desligar fuzzing e garantir que **suppression** entra automaticamente.

12) Tabelas & Schemas (novos/ajustes)

`consent_ledger`

```
CREATE TABLE consent_ledger (
  user_id    uuid NOT NULL,
  purpose    text NOT NULL, -- MAP_GLOBAL, SHARE_PUBLIC_POINT, HISTORY
  state      text NOT NULL CHECK (state IN ('GRANTED','DENIED')),
  updated_at timestamptz NOT NULL,
  PRIMARY KEY (user_id, purpose)
);
```

`silence_zones` (já definido na Camada 05; aqui com nota de cifragem de campo):

- `geom` armazenado como **ciphertext**; descriptografado apenas em memória no Privacy Filter.

Flag em `checkins` :

```
ALTER TABLE checkins ADD COLUMN is_private_zone boolean NOT NULL DEFAULT false;
ALTER TABLE checkins ADD COLUMN deleted_at timestamptz;
```

13) Contratos de Erro (APIs relacionadas)

Código	Mensagem	Quando
<code>CONSENT_REQUIRED</code>	consentimento ausente para esta operação	Tentou contribuir no global sem consent
<code>INSIDE_SILENCE_ZONE</code>	transmissão bloqueada em zona privada	Evento em casa/trabalho
<code>LOW_K_SUPPRESSED</code>	célula suprimida por privacidade	$N < k$
<code>PRIVACY_RATE_LIMIT</code>	muitas consultas na mesma célula	Mitigar triangulação
<code>DEVICE_INTEGRITY_FAIL</code>	dispositivo não confiável	Falha em verificação de integridade

14) SLOs de Privacidade & Segurança

- **Zero** vazamento de coordenadas brutas em payloads públicos.
- **≥ 99.9%** de enforcement de Zonas de Silêncio (telemetria).

- **Rollback < 2 min (p95)** após detecção de fraude/exclusão.
- **K-safety** aplicado em **100%** das renderizações de zonas.

✓ Fechamento

Com **geofuzzing dinâmico**, **k-safety**, **zonas de silêncio**, **consent ledger**, **auditoria imutável** e **criptografia ponta a ponta do pipeline**, esta camada fecha o ciclo de **privacidade por design** sem sacrificar performance, utilidade ou escalabilidade do Mapa de Frequências.

CAMADA 11 — OBSERVABILIDADE & AUDITORIA VIBRACIONAL (Métricas, Logs, Traces, SLOs Operacionais)

“Não basta vibrar. É preciso medir, auditar e garantir que o pulso do mapa seja confiável, seguro e monitorado de ponta a ponta.”

🎯 Objetivo da Camada

Garantir que cada atualização do **Mapa de Frequências** seja rastreável, auditável e mensurável em tempo real. Esta camada estabelece a infraestrutura de **observabilidade completa**: métricas, logs, traces, auditoria imutável e alertas proativos.

Ela une **infraestrutura clássica de DevOps (Prometheus, Grafana, Datadog)** com **métricas vibracionais únicas do FriendApp**, garantindo compliance, performance e confiança.

📊 Métricas Obrigatórias

Métrica	Tipo	Descrição	SLO/Alvo
latencia_atualizacao	Latência	Tempo médio entre check-in e refletir no mapa	≤ 3s Firestore / ≤ 5 min agregados
taxa_privacidade_suprimida	Privacidade	% de dados suprimidos por geofuzzing ou zonas silenciosas	≥ 98% dos casos
taxa_fraude_detectada	Segurança	Percentual de check-ins fraudulentos revertidos	≥ 99% revertidos em < 2 min
consistencia_agregados	Consistência	Diferença entre dados brutos e pré-agregados por célula	≤ 1% de desvio
uptime_servico_mapa	Confiabilidade	Disponibilidade do serviço de mapa e APIs	≥ 99,95%
tempo_rollback	Resposta a fraudes	Tempo médio para rollback após fraude detectada	≤ 60s

🔍 Logs & Auditoria

Todos os eventos relacionados ao mapa devem ser registrados em **banco imutável** com criptografia dupla.

Tipos de Logs:

- **Log de Check-in:** usuário, timestamp, geofuzzing aplicado.
- **Log de Agregação:** cálculos por célula (H3/S2), estado vibracional resultante.
- **Log de Anomalia:** detecção de clusters suspeitos ou manipulação.
- **Log de Rollback:** eventos revertidos com detalhes técnicos.

- **Log de Acesso Administrativo:** consultas ao painel de auditoria.

Auditoria & Compliance

- Dados críticos armazenados em **WORM (Write Once, Read Many)** para LGPD/GDPR.
- Consultas só podem ser feitas por **IA de Auditoria** ou **solicitação judicial**.
- Logs vibracionais vinculados ao **Painel Administrativo** para rastreabilidade.







Traces & Observabilidade em Tempo Real

- **OpenTelemetry + Grafana Tempo** para rastrear a jornada de cada atualização:
 - Usuário → Check-in → Firestore → Job de agregação → PostgreSQL → Renderização no app.
- Cada trace vinculado a **IDs únicos de sessão vibracional**.



Alertas Automáticos

Definições de alertas configuradas no **Prometheus + AlertManager**:

-  **Queda de Consistência:** desvio > 1% entre agregados e brutos.
-  **Latência Anormal:** atualização > 10s no Firestore.
-  **Fraude em Massa:** > 20 check-ins suspeitos na mesma célula em < 1 min.
-  **Colapso de Zona:** queda vibracional extrema em cidade inteira (Aurah Kosmos notificada).



SLOs Operacionais

- **Latência Firestore (individuais):** 3 segundos.
- **Latência Agregados (zonas):** 5 minutos.
- **Uptime APIs Mapa:** 99,95%.
- **Rollback Fraude:** 60 segundos.
- **Retenção de Logs:**
 - Check-ins → 12 meses.
 - Fraudes → 36 meses.
 - Auditoria → Indefinido (criptografado).



Fluxo Resumido

```
[Usuário Faz Check-in]
  ↓
[Firestore Real-time]
  ↓
[Job de Agregação Background]
  ↓
[PostgreSQL com H3/S2]
  ↓
[Observabilidade: Logs + Métricas + Traces]
  ↓
[Alertas Prometheus + Painel Grafana]
  ↓
```

Compliance & Segurança

- **LGPD/GDPR:** Consentimento explícito para visibilidade no mapa.
- **Zonas Silenciosas:** Dados de casa/trabalho nunca transmitidos.
- **Geofuzzing Dinâmico:** Proteção adaptativa por densidade.
- **Auditoria Externa:** Logs prontos para inspeções de segurança e ética.

Fechamento da Camada

“Aqui o invisível se torna mensurável. O Mapa de Frequências não só pulsa, ele prova, audita e garante sua própria integridade. Esta camada é o escudo técnico que sustenta a confiança de milhões.”



CAMADA 12 — MOTOR DE GEOHASHING & PRÉ-AGREGAÇÃO VIBRACIONAL

Objetivo da Camada

Resolver o gargalo técnico de **tempo real global** por meio de um **motor de pré-agregação vibracional**, garantindo **escalabilidade, precisão e baixo custo de infraestrutura**.

Conceito

Em vez de recalcular o estado energético de todos os usuários em tempo real, o sistema adota um **modelo híbrido**:

-  **Pontos individuais** (usuários ativos) → atualizados em **Firestore** em tempo real.
-  **Zonas agregadas (células vibracionais)** → pré-calculadas a cada X minutos, usando **geohashing**.

Essa combinação garante **fluidez na experiência do usuário** sem sobrecarregar os bancos.

Estrutura do Geohashing

Algoritmo	Função	Benefício
H3 (Uber)	Grade hexagonal global	Melhor equilíbrio entre precisão e performance
S2 (Google)	Alternativa com suporte a queries geográficas nativas	Integração direta com APIs e PostGIS

- Cada célula representa uma área (~300m² até 5km², dependendo do zoom).
- Usuários em check-in vibracional são agregados na célula correspondente.

Lógica de Processamento

1. **Coleta bruta** → check-ins e frequências enviados pelos usuários (Firestore).
2. **Job em background** (Cloud Functions / Workers):
 - Executado a cada **5 minutos**.
 - Agrega dados de todas as células ativas.

- Calcula o **VIB_SCORE médio** da célula com:

$$\text{VIB_SCORE} = \Sigma (\text{freq_usuário} * \text{peso_tempo}) / \Sigma \text{pesos}$$

3. **Aplicação de decaimento temporal** → check-ins recentes têm mais peso.
4. **Filtragem de outliers** → média aparada (trimmed mean) para evitar distorções.
5. **Persistência em banco:**
 - PostgreSQL + PostGIS → histórico e queries geográficas.
 - Firestore → estados mais recentes para consumo direto no app.

Consumo no App

- O usuário abre o mapa → o app consulta apenas as **células agregadas visíveis**.
- Atualização no frontend → **quase instantânea**, pois o cálculo já está pronto.
- Usuários individuais → continuam sendo exibidos via **streaming Firestore**.





Privacidade

- **Geofuzzing dinâmico** → distorção maior em áreas de baixa densidade.
- **Zonas de silêncio** → casa/trabalho nunca aparecem no mapa, nem distorcidos.

Estrutura Técnica

Componente	Tecnologia	Função
Motor de Agregação	Cloud Functions + Python/Go	Calcula médias por célula
Banco Geográfico	PostgreSQL + PostGIS	Histórico, queries avançadas
Banco em Tempo Real	Firestore	Estados instantâneos
Jobs Scheduler	Pub/Sub ou Cron Jobs	Dispara processamentos periódicos
API de Consumo	<code>/api/map/frequencies</code>	Retorna dados já agregados por célula

Benefícios da Pré-Agregação





-  **Escalabilidade Global** → suporta milhões de usuários sem gargalo.
-  **Baixa Latência** → consultas instantâneas no mapa.
-  **Privacidade reforçada** → dados sempre agregados em grupos, não expostos individualmente.
-  **Visão Coletiva** → o mapa pulsa de forma realista sem depender de cada check-in em tempo real.

✨ "Com o motor de geohashing, o Mapa de Frequência deixa de ser um cálculo inviável e se torna um radar global escalável, seguro e pulsante."

CAMADA 13 — FÓRMULA DE VIB_SCORE (Cálculo + Decaimento Temporal + Outliers)

Objetivo da Camada

Definir uma **fórmula matemática robusta e executável** para calcular o **estado vibracional** de usuários, locais e zonas, considerando **tempo, intensidade e consistência**. A meta é que o **VIB_SCORE** seja:

-  **Matematicamente sólido**
-  **Resiliente a outliers e fraudes**
-  **Representativo da energia atual, não do passado**
-  **Escalável para agregações coletivas**

Estrutura Geral do VIB_SCORE

Fórmula Base

$$\text{VIB_SCORE}(\text{zona/local}) = \frac{\sum (\text{freq_user_i} * \text{peso_tempo_i} * \text{peso_confiança_i})}{\sum (\text{pesos})}$$

- `freq_user_i` → frequência vibracional individual (0 a 1000).
- `peso_tempo_i` → fator de decaimento temporal (check-ins recentes valem mais).
- `peso_confiança_i` → score de confiança (verificação DUC/DCO, histórico do usuário, consistência).
- `Σ(pesos)` → normalização para manter o score entre **0 e 1000**.

Decaimento Temporal

Check-ins recentes impactam mais que antigos.

Fórmula

$$\text{peso_tempo} = e^{(-\Delta t / \lambda)}$$

- `Δt` → tempo desde o check-in (em minutos).
- `λ` → constante de decaimento (ex: 60 → após 1h o peso cai para ~37%).

📌 Isso garante que **um check-in de 5 min atrás vale muito mais** do que um de 2h atrás.

Tratamento de Outliers

Problema

Um usuário com frequência extremamente alta/baixa pode distorcer a média.

Solução

- Usar **Média Aparada (Trimmed Mean)** → remover o **5% mais alto e o 5% mais baixo** dos valores antes de calcular a média.
- Alternativa: **Winsorizar** os extremos (substituir outliers pelo valor máximo permitido no intervalo).

📌 Assim, o mapa reflete **a energia coletiva real**, e não distorções pontuais.

Peso de Confiança (Anti-Fraude)


Nem todos os check-ins têm o mesmo valor. O sistema atribui **peso_confiança** com base em:

Critério	Peso	Observação
Usuário Verificado (DUC/DCO)	1.2	Confiabilidade máxima
Usuário Não Verificado	1.0	Peso neutro
Check-in suspeito (detectado por IA)	0.3	Impacto reduzido
Check-in reincidente fraudulento	0	Ignorado totalmente

Estrutura de Banco

Campo	Tipo	Descrição
user_id	UUID	Identificação do usuário
zona_hash	H3 index	Célula geográfica
freq_user	INT (0–1000)	Frequência vibracional
timestamp	TIMESTAMP	Hora do check-in
confidence_score	FLOAT (0–1.2)	Peso de confiança
processed	BOOLEAN	Se já foi agregado em VIB_SCORE da zona

Exemplo de Cálculo

 Usuários ativos em uma célula (5 min atrás):

- Usuário A: 900 (verificado)
- Usuário B: 500 (não verificado)
- Usuário C: 1000 (fraudulento detectado)





Após aplicação dos pesos:

$$\begin{aligned} \text{VIB_SCORE} = & \\ & (900 \times 1.0 \times 1.2) + (500 \times 1.0 \times 1.0) + (1000 \times 0.3) \\ & \text{-----} \\ & (1.2 + 1.0 + 0.3) \end{aligned}$$

$$\text{VIB_SCORE} = 1460 / 2.5 = 584$$

O score da célula = **584** → estado de **Expansão Moderada**.

Benefícios

-  Representa **a energia atual**, não acumulada do passado.
-  Evita distorções de outliers e fraudes.
-  Mantém coerência estatística para zonas pequenas e grandes.
-  Flexível: parâmetros (λ , trimmed mean, pesos) podem ser calibrados com machine learning.

✨ "Com esta fórmula, o VIB_SCORE se torna mais do que um número: é um reflexo matemático justo, limpo e vivo da energia coletiva."

CAMADA 14 — CLASSIFICAÇÃO VIBRACIONAL

Objetivo da Camada

Traduzir o **VIB_SCORE** numérico (calculado na camada anterior) em **estados visuais e semânticos** que o usuário enxerga no mapa. O objetivo é criar uma lógica clara e imersiva para classificar zonas, locais e usuários em **estados vibracionais intuitivos**, representados por cores, ícones e animações.

Intervalos de Classificação

Faixa VIB_SCORE	Estado Vibracional	Cor Base	Ícone / Estilo	Significado Técnico
800 – 1000	Pico Energético	Vermelho-Dourado pulsante	🔥	Alta densidade vibracional positiva; local em ebulição energética.
600 – 799	Expansão	Roxo-Lilás com brilho suave	✨	Crescimento coletivo estável; boas interações fluindo.
400 – 599	Transição	Azul-Ciano dinâmico	🌊	Zona em oscilação, pode evoluir para expansão ou cair em colapso.
200 – 399	Colapso Parcial	Cinza-Azul opaco	⚠️	Queda energética, interações negativas detectadas.
0 – 199	Colapso Total	Preto com pulsação vermelha	🚫	Risco vibracional crítico, local pode exigir intervenção da IA.

Regras Visuais

- **Animações Dinâmicas**
 - Estados **altos** (Expansão, Pico) → animações em **espiral/brilho crescente**.
 - Estados **baixos** (Colapso) → pulsos intermitentes e redução de luminosidade.
- **Densidade de Usuários**
 - Se > 50 usuários ativos numa mesma célula: camada extra de partículas douradas.
 - Se < 5 usuários ativos: efeito “respiração lenta” para indicar baixa presença.
- **Eventos**
 - Eventos em zonas de Pico recebem selo especial: “🔥 Hotspot Vibracional”.

Algoritmo de Classificação

Pseudocódigo simplificado:





```
def classificar_estado(vib_score):
    if vib_score >= 800:
        return "Pico Energético"
    elif vib_score >= 600:
        return "Expansão"
    elif vib_score >= 400:
        return "Transição"
    elif vib_score >= 200:
        return "Colapso Parcial"
    else:
        return "Colapso Total"
```

Integração com Outros Sistemas

- **Feed Sensorial** → posts feitos em zonas de Pico recebem boost automático.

- **Eventos** → IA recomenda mais fortemente eventos em Expansão ou Pico.
- **Segurança Vibracional** → colapsos totais acionam firewall vibracional automático.
- **Gamificação / FriendCoins** → check-ins em zonas de Pico podem gerar bônus.

Benefícios

-  Usuário entende rapidamente o **estado energético de um local**.
-  Cores e símbolos criam uma experiência **imersiva e intuitiva**.
-  Técnicos e devs têm **limiares numéricos claros** para classificar zonas.
-  Mantém consistência com a fórmula de VIB_SCORE (Camada 13).

✨ “Aqui, o número se transforma em cor, a lógica em experiência. O Mapa de Frequência pulsa como um radar vivo da energia humana.”

CAMADA 15 — CHECK-IN VIBRACIONAL

Objetivo da Camada

Validar **presença real + intenção vibracional** do usuário em um local, garantindo que o **Mapa de Frequência** reflita somente dados legítimos, ao mesmo tempo em que protege contra **check-ins falsos** ou **fraudes de geolocalização**.

Lógica de Validação

O Check-in Vibracional não é apenas um “GPS ping”, mas sim um **processo multi-etapas**:

Etapa	Validação	Descrição Técnica
1. Geolocalização	GPS + Rede Wi-Fi + Cell Tower	O sistema cruza múltiplas fontes de localização para reduzir spoofing.
2. Intenção Declarada	UI no App	O usuário deve confirmar a intenção do check-in (ex: “estou participando de um evento”, “visitando local parceiro”).
3. Frequência Pessoal	IA Aurah Kosmos	A IA avalia coerência vibracional (se o estado energético do usuário é compatível com o local).
4. Tempo de Permanência	Monitoramento mínimo (5 min)	Check-ins só são validados após um tempo de presença mínima.
5. Anti-Spoofing	Algoritmo de Consistência	Detecta saltos geográficos impossíveis (teletransporte digital) e bloqueia.

Regras Anti-Fraude

Cenário de Fraude	Deteção	Ação Automática
GPS Fake (Mock Location)	API Android/iOS detecta uso de mock	Check-in bloqueado + alerta no log
Check-in em Massa (Bots)	Múltiplos check-ins idênticos no mesmo segundo	Bloqueio automático + auditoria IA
Teletransporte Digital	Movimento > 500 km em 1 min	Marca como inválido
Repetição Excessiva	10+ check-ins em 1h no mesmo local	Apenas o primeiro válido, demais descartados
Check-in Fraudulento em Evento	Divergência entre presença física e ticket validado	Evento notificado + check-in revertido

Persistência de Dados

Os dados de check-in são armazenados em **camadas distintas**:

Banco / Serviço	Função	Tipo de Dado
Firestore (tempo real)	Atualização instantânea no mapa	Status de presença, aura, timestamp
PostgreSQL (relacional)	Histórico e auditoria	Logs detalhados (usuário, local, duração)
Aurah Kosmos Core	Ajuste vibracional	Interpretação da coerência entre usuário e local
Sistema de Segurança	Anti-fraude	Logs de detecção e bloqueio

Fluxo Operacional

flowchart TD

```
A[Usuário inicia Check-in] → B[Validação GPS + Wi-Fi]
B → C[Confirmação Intenção no App]
C → D[IA Aurah Kosmos valida frequência]
D → E{Permanência > 5min?}
E -- Sim → F[Check-in Validado]
E -- Não → G[Check-in Pendente]
F → H[Mapa de Frequência atualizado]
F → I[Logs gravados em PostgreSQL]
G → J[Aguardando tempo mínimo]
```

Integrações Ativas

- **Mapa de Frequência** → Atualização imediata das células vibracionais.
- **Eventos e Locais Parceiros** → Check-in libera benefícios e registro de presença.
- **Aurah Kosmos** → IA ajusta coerência vibracional entre usuário e ambiente.
- **Gamificação** → Check-ins válidos geram XP, conquistas e FriendCoins.
- **Segurança Vibracional** → Ativa bloqueios automáticos em fraudes detectadas.

Benefícios

- Evita **spam de localização** ou manipulação de métricas.
- Garante **qualidade dos dados vibracionais** no mapa.
- Conecta **presença física + intenção + energia pessoal**, criando uma métrica única.
- Dá aos devs **contratos claros de persistência e logs**.

✨ "Aqui, o simples ato de estar em um lugar se transforma em dado vibracional puro. O check-in não é só presença: é coerência entre corpo, intenção e campo."

CAMADA 16 — SISTEMA DE GEOHASHING E PRÉ-AGREGAÇÃO

Objetivo da Camada

Garantir que o **Mapa de Frequência** possa ser usado por **milhões de usuários simultaneamente**, sem travamentos ou custos proibitivos, substituindo cálculos "em tempo real" por um modelo de **pré-agregação eficiente com geohashing**.

Problema Original

- Cada abertura do mapa exigiria **recalcular a média vibracional** dos usuários em tempo real.
- Consultas pesadas (ex: *SELECT all users WHERE location in X*) seriam lentas e custosas.
- O sistema não escalaria em cidades com **milhares de usuários ativos**.

Solução: Geohashing + Pré-Agregação

1. Estrutura de Geohash

- Dividir o mapa do mundo em **células hexagonais** (H3 da Uber ou S2 do Google).
- Cada célula tem um **ID único** (`hash_id`) que representa uma área geográfica.
- Nível de precisão ajustável:
 - **Nível alto (zoom bairro/rua)** → células pequenas (~50m).
 - **Nível baixo (zoom cidade/estado)** → células grandes (~km).

2. Processo de Pré-Agregação





Etapa	Descrição	Ferramenta
1. Coleta	Check-ins vibracionais + frequência atual de usuários	Firestore
2. Job de Background	A cada 5 minutos, processa dados por célula geohash	Worker em Kubernetes
3. Cálculo do VIB_SCORE	Média aparada + decaimento temporal	Postgres + Redis
4. Persistência	Grava resultados pré-agregados em cache	Redis (hot data)
5. Consulta Mobile	App requisita apenas células visíveis na tela	API <code>/mapa/cell/:hash_id</code>

3. Fórmula de Agregação Refinada

$$\text{VIB_SCORE(célula)} = \frac{(\sum (\text{frequência_usuario} * \text{peso_tempo}) - \text{outliers})}{N}$$

- **peso_tempo**: decai exponencialmente (check-in recente = peso maior).
- **outliers**: descartados pelo algoritmo *trimmed mean*.
- **N**: número de usuários válidos na célula.

Regras de Escalabilidade

-  **Atualizações cíclicas**: cada célula é recalculada em *batch jobs*.
-  **Dados em cache (Redis)**: consultas do mobile retornam em milissegundos.
-  **Lazy Loading**: apenas células visíveis no mapa são carregadas.
-  **Fallback**: se não houver dados suficientes → aplica camada heurística (ex: vibração neutra).

Integrações Ativas

- **Mapa de Frequência Mobile** → consome células pré-agrupadas.
- **Aurah Kosmos** → ajusta recomendações com base nas células vibracionais.
- **Eventos e Locais Parceiros** → vinculam-se a células específicas (hash_id obrigatório).
- **Gamificação** → conquistas baseadas em check-ins por células energéticas.
- **Observabilidade** → métricas de performance monitoradas em Prometheus + Grafana.

Fluxo Operacional

flowchart TD

```

A[Check-in Usuário] → B[Firestore]
B → C[Job Background 5min]
C → D[Geohash Aggregation]
D → E[Calcular VIB_SCORE]
E → F[Redis Cache]
F → G[API /mapa/cell/:id]
G → H[App Mobile renderiza células vibracionais]
  
```

Benefícios

- Escala para **milhões de usuários** sem queda de performance.
- Consulta rápida: **<200ms por requisição**.
- Redução de custos de infraestrutura.
- Permite mapas vivos, dinâmicos e sustentáveis.

✨ “O segredo da magia está nos bastidores. O usuário vê um mapa vivo em tempo real, mas na essência, são ciclos inteligentes de pré-agregação que mantêm a orquestra vibrando sem parar.”

CAMADA 17 — PRIVACIDADE E ZONAS DE SILÊNCIO

Objetivo da Camada

Elevar a **segurança e confiança do usuário** no Mapa de Frequência, garantindo que **nenhuma localização sensível seja revelada** e que a experiência continue fluida, mas com proteção robusta e invisível.

Problema a Resolver

- Usuários podem ser **reidentificados** mesmo com geofuzzing fixo ($\pm 50m$).
- Residências e locais privados (ex: trabalho, escolas) não podem ficar vulneráveis.
- O usuário precisa sentir que **tem o controle** da sua presença no mapa.

Soluções Implementadas

1. Geofuzzing Dinâmico (Adaptive Fuzzing)

- **Maior distorção** em áreas de baixa densidade (bairros residenciais).
- **Menor distorção** em áreas movimentadas (parques, shoppings).

- Algoritmo ajusta automaticamente a precisão conforme contexto:

```
if densidade_usuarios < threshold:
    deslocamento = random(100m, 300m)
else:
    deslocamento = random(20m, 50m)
```

2. Zonas de Silêncio (Silent Zones)

- Usuário pode marcar locais privados no app: 🏠 casa, 🏢 trabalho, 🧑 locais pessoais.
- Dentro dessas zonas → **nenhum dado é transmitido** (nem mesmo distorcido).
- Zonas são **criptografadas localmente** e **não ficam visíveis nem para admins**.

3. Consentimento Granular

- Usuário escolhe se quer:
 - ☒ Mostrar-se no mapa público (visível para outros).
 - ☒ Apenas no mapa coletivo (energia sem identificação).
 - ☒ Não aparecer (somente dados agregados).

Regras Técnicas de Privacidade

Cenário	Ação do Sistema
Usuário dentro de Zona de Silêncio	Nenhum dado coletado/transmitido
Usuário fora da zona, mas em baixa densidade	Geofuzzing dinâmico aplicado (±100m–300m)
Usuário em evento/área pública	Localização precisa (apenas se check-in ativo)
Usuário optou por anonimato	Energia agregada aparece no coletivo, sem avatar individual

Banco e Estrutura

Tabela	Campos	Observação
user_zones	id_user, hash_location, zone_type, radius	Criptografado com AES-256
location_logs	id_user, hash_location, timestamp, fuzzed_coord	Não registra pontos dentro de zonas de silêncio
privacy_preferences	id_user, share_mode (public, collective, hidden)	Definido pelo usuário no app

Fluxo Operacional

flowchart TD

```
A[GPS User] → B[Verifica Zona de Silêncio]
B → |Dentro| C[Não coleta localização]
B → |Fora| D[Aplica Geofuzzing Dinâmico]
D → E[Armazena em Firestore]
E → F[Atualiza Mapa de Frequência]
F → G[Exibe no App (público, coletivo ou oculto)]
```

Integrações Ativas

- **Aurah Kosmos** → respeita zonas de silêncio, não sugere conexões dentro delas.
- **Feed Sensorial** → só mostra check-ins voluntários, nunca automáticos em zonas privadas.
- **Sistema de Segurança Vibracional** → botão de pânico funciona mesmo em zonas privadas (localização só é transmitida em emergência).
- **Gamificação** → check-ins em zonas privadas não contam para XP ou FriendCoins.

Benefícios

- Usuários têm **confiança total** de que locais sensíveis nunca serão revelados.
- Privacidade é **configurável e adaptativa**, sem atrapalhar a experiência.
- Sistema se torna **à prova de rastreamento invasivo**.

✨ "No FriendApp, o mapa revela o coletivo, mas protege o individual. A energia se compartilha, mas a intimidade permanece inviolável."

CAMADA 18 — FÓRMULA AVANÇADA DO VIB_SCORE

Objetivo da Camada

Definir uma **fórmula robusta, escalável e justa** para calcular o estado vibracional de usuários, locais e células no mapa, evitando distorções por outliers, respeitando o tempo real com eficiência e mantendo coerência com a experiência do usuário.

Problemas Resolvidos nesta Camada

1. **Média distorcida por outliers**
 - Check-ins muito extremos (muito altos ou baixos) poderiam distorcer o estado coletivo.
2. **Peso igual para check-ins antigos e recentes**
 - Energia antiga influenciava demais o presente.
3. **Cálculo em tempo real inviável**
 - Recalcular tudo por usuário geraria custo proibitivo.

Solução Matemática Definida

Fórmula Geral do VIB_SCORE

```
VIB_SCORE(célula, t) = TrimmedMean(  
  Σ (freq_user * weight_time * weight_confiança) / N  
)
```

Componentes

1. **Trimmed Mean (Média Aparada)**
 - Remove automaticamente os 5% maiores e menores valores antes de calcular a média.

- Garante que extremos não distorçam o coletivo.

2. Weight_time (Decaimento Temporal)

- Fórmula: $\exp(-\lambda * \Delta t)$
- Quanto maior o Δt (tempo desde o check-in), menor o peso.
- Exemplo:
 - Check-in há 5 min → peso ~1.0
 - Check-in há 2h → peso ~0.35
 - Check-in há 6h → peso ~0.05

3. Weight_confiança

- Baseado no score de confiabilidade do usuário (cadastro verificado, histórico sem fraudes, engajamento positivo).
- Usuários confiáveis pesam mais que contas recém-criadas.

Pseudocódigo do Algoritmo

```
def calcular_vib_score(celula, t_atual):
    freq_ponderada = []
    for checkin in celula.checkins:
        delta_t = t_atual - checkin.timestamp
        weight_time = exp(-0.2 * delta_t_horas)
        weight_conf = checkin.user.conf_trust
        valor = checkin.freq * weight_time * weight_conf
        freq_ponderada.append(valor)

    # Remove outliers (5% menores e maiores)
    freq_filtrada = remove_outliers(freq_ponderada, p=0.05)

    # Média final
    vib_score = mean(freq_filtrada)
    return vib_score
```

Integrações Ativas

- **Aurah Kosmos** → pode ajustar os parâmetros λ (decay) e limites de outliers conforme padrões coletivos.
- **Mapa de Frequência** → consome o VIB_SCORE já processado para cada célula (pré-agregação via H3).
- **Gamificação** → XP e recompensas atreladas à participação em zonas de alta frequência.
- **Sistema de Segurança** → zonas com VIB_SCORE baixo demais podem disparar alertas.

Benefícios

- Escalabilidade: cálculo feito em background jobs, não em tempo real.
- Justiça: evita manipulação por outliers ou fraudes.
- Precisão: reflete o "agora" com decaimento temporal.
- Flexibilidade: IA Aurah pode recalibrar os pesos dinamicamente.

✨ "O VIB_SCORE é a métrica que traduz o invisível em número. Ele não é só cálculo: é justiça vibracional aplicada em escala global."

◆ CAMADA 19 — ARQUITETURA DE BACKGROUND JOBS E GEOHASHING

🎯 Objetivo da Camada

Garantir que o **Mapa de Frequência** funcione em escala global, com milhares/milhões de usuários simultâneos, **sem sobrecarregar a infraestrutura**.

A estratégia é usar **background jobs + geohashing** para manter a sensação de "tempo real" no app, sem depender de cálculos pesados a cada segundo.

📌 Problema Original

- Consultar e recalcular a energia de todos os usuários em tempo real geraria **latência alta e custos proibitivos**.
- O manual antigo previa atualizações a cada 1 minuto → inviável em escala global.

✅ Solução Arquitetural

1. Geohashing (Divisão Inteligente do Mapa)

- O planeta é dividido em células hexagonais (biblioteca **H3 da Uber** ou **S2 do Google**).
- Cada célula representa uma "zona vibracional" → bairro, praça, quarteirão etc.
- Usuários que fazem check-in são atribuídos à célula correspondente.

📌 Exemplo:

- Usuário A em São Paulo → célula H3 nível 9 (~1 km²).
- Usuário B no mesmo bairro → cai na mesma célula, agregando energia.

2. Background Jobs (Agregação Periódica)

- Jobs em **fila de processamento** (RabbitMQ, Kafka ou Cloud Pub/Sub).
- A cada **5 minutos** (configurável):
 - Recalculam o **VIB_SCORE médio** de cada célula.
 - Aplicam **time-decay** (peso menor para check-ins antigos).
 - Atualizam banco de dados com o estado vibracional agregado.

🌀 Benefício:

- Para o usuário, a experiência continua sendo "tempo real".
- Mas o backend só processa agregados a cada 5 min, economizando **90% da carga computacional**.

3. Dualidade "Tempo Real" vs. "Quase Tempo Real"

- **Pontos individuais (Firestore):**
 - Check-in do usuário é salvo em tempo real → mostra no mapa instantaneamente.
- **Agregado da célula (Postgres + Jobs):**

- Estado da zona é atualizado via jobs → exibido em blocos de 5 minutos.

🔑 Assim, o usuário vê o próprio ponto em tempo real, mas o **clima coletivo da região** é atualizado de forma escalável.

🧩 Fluxo Técnico Simplificado

flowchart TD

```
U[Usuário faz Check-in Vibracional]
F[Firestore] → J[Job Queue]
J → P[Processador de Background]
P → H[Hashing H3/S2]
H → DB[(PostgreSQL + Redis Cache)]
DB → A[App Mobile Consulta Células]
```

📊 Tecnologias Sugeridas

Camada	Ferramenta Recomendada	Função
Job Queue	Kafka / PubSub / RabbitMQ	Processamento assíncrono e distribuído
Geohashing	H3 (Uber)	Divisão do planeta em hexágonos precisos
Banco Principal	PostgreSQL + PostGIS	Consulta geográfica e dados históricos
Cache	Redis	Retorno instantâneo de dados agregados
Real-time Users	Firestore	Posição individual e presença em tempo real

✅ Benefícios da Arquitetura

- **Escalabilidade Global:** suporta milhões de check-ins por hora.
- **Custo Controlado:** processamento assíncrono evita sobrecarga.
- **UX Fluida:** usuários percebem dinamismo, mesmo com agregação de 5 min.
- **Flexibilidade:** granularidade ajustável (células maiores ou menores).

✨ "Com geohashing e jobs inteligentes, o mapa deixa de ser pesado e passa a ser vivo, fluido e sustentável em escala planetária."

📦 CAMADA 20 — ARQUITETURA DE BANCO DE DADOS HÍBRIDA

🎯 Objetivo da Camada

Definir como os **dados do Mapa de Frequência** são armazenados, consultados e distribuídos em tempo real, garantindo **equilíbrio entre performance, escalabilidade e persistência histórica**.

📦 Modelos de Armazenamento

1. PostgreSQL + PostGIS (Histórico & Consultas Geográficas)

- Função: **armazenar dados históricos e agregados**.
- Inclui cálculos espaciais via **PostGIS** (consultas por raio, regiões, interseções).

- Exemplo de dados:
 - `id_user`, `célula_H3`, `score_vibracional`, `timestamp`.
- Benefício: consultas complexas sobre evolução de regiões ao longo do tempo.

2. Firestore (Tempo Real - Posição Individual)

- Função: manter a presença **instantânea** dos usuários ativos.
- Cada check-in gera:
 - `user_id`, `lat`, `lon`, `estado_atual`, `intenção`.
- Atualização contínua: **< 1 segundo de latência** para refletir o usuário no mapa.
- Dados efêmeros → expiram em 2h para não sobrecarregar.

3. Redis (Cache e Resposta Instantânea)

- Função: reduzir tempo de resposta no frontend.
- Armazena:
 - Estados **pré-calculados das células H3** (via jobs).
 - Resultados de queries comuns.
- TTL: 5–10 min → mantém frescor sem sobrecarregar PostgreSQL.

Fluxo Operacional

flowchart TD
 U[Usuário faz Check-in] → FS[Firestore (tempo real)]
 FS → J[Job de Agregação]
 J → PSQL[(PostgreSQL + PostGIS)]
 J → R[Redis Cache]
 PSQL → R
 R → APP[Aplicativo Consulta Células]

Estratégia de Sincronização

Camada	Frequência	Tipo de Dados
Firestore	Instantâneo (<1s)	Check-ins individuais, presença ativa
Redis	Renovação 5–10 min	Estados agregados de células
PostgreSQL	Permanente	Histórico e análise avançada

Segurança & Privacidade

- Dados sensíveis do usuário → **hash + geofuzzing dinâmico**.
- Zonas de Silêncio → dados não são salvos nem no histórico.
- Logs e auditoria → armazenados em camada separada (compliance).

Benefícios do Modelo Híbrido

- **Velocidade (Firestore + Redis):** experiência fluida para usuários ativos.

- **Histórico robusto (PostgreSQL):** análises globais, auditoria e previsões.
- **Escalabilidade:** cada camada assume apenas o que faz melhor.

✨ “Com o banco híbrido, o Mapa de Frequência respira como o mundo: rápido no agora, sólido no passado e visionário no futuro.”

◆ CAMADA 21 — LÓGICA DO VIB_SCORE

🎯 Objetivo da Camada

Definir a fórmula e os algoritmos para calcular o **estado vibracional coletivo** de cada célula/área do mapa, garantindo precisão, resiliência contra fraudes e escalabilidade.

📊 Fórmula Base do VIB_SCORE

O **VIB_SCORE** de uma célula (zona geográfica) é calculado considerando:

1. **Energia dos usuários ativos** (check-ins + presença contínua).
2. **Tempo de emissão** (quanto mais recente, mais peso).
3. **Compatibilidade de intenções** (peso maior para intenções coletivas similares).
4. **Correção de outliers** (remoção de valores extremos).

📐 Fórmula Matemática

$$\text{VIB_SCORE(célula)} = \frac{(\sum [\text{freq_usuario} * \text{peso_intencao} * \text{decaimento_tempo}] - \text{outliers})}{N_ajustado}$$

Onde:

- **freq_usuario**: frequência vibracional registrada pelo usuário (0–1000).
- **peso_intencao**: fator entre 0.8–1.2 dependendo da intenção declarada (ex: meditar = mais coerente que zoar no local).
- **decaimento_tempo**: $e^{(-\lambda * \Delta t)}$ → eventos recentes têm mais peso.
- **outliers**: valores removidos via **média aparada (trimmed mean)**.
- **N_ajustado**: total de usuários após remoção de outliers.

🕒 Time Decay (Decaimento Temporal)

- **Check-ins recentes (últimos 15 min)** → peso = 1.0
- **30 min atrás** → peso ~ 0.7
- **60 min atrás** → peso ~ 0.4
- **>2h atrás** → peso próximo de 0 (sem relevância).

💡 Isso garante que o mapa represente **o agora**, não o passado.

🚫 Outliers e Ruídos

- Outliers positivos (ex: usuário marcando 1000 em local vazio).

- Outliers negativos (usuário tentando sabotar jogando 0).

🔧 Solução:

- Aplicar **Trimmed Mean (10%)** → remove 10% dos extremos superior e inferior.
- Validar consistência via IA → detectar padrões suspeitos e aplicar rollback automático.

Estados Energéticos Definidos

Cada célula H3 recebe um estado qualitativo baseado no VIB_SCORE:

VIB_SCORE Médio	Estado	Cor no Mapa
0 – 200	Colapso	Vermelho Escuro
201 – 400	Transição Negativa	Laranja
401 – 600	Neutro/Estável	Amarelo
601 – 800	Expansão Positiva	Verde
801 – 1000	Pico de Alta Frequência	Azul/Lilás

Lógica de Reversão de Fraudes

1. IA detecta check-in suspeito (ex: repetição em alta frequência em local privado).
2. Marca log para auditoria.
3. Remove influência do usuário da célula.
4. Recalcula VIB_SCORE sem esse input.

Exemplo de Pseudocódigo

```
def calcular_vib_score(celula):
    usuarios = get_checkins(celula)
    usuarios = remover_outliers(usuarios)

    soma = 0
    for u in usuarios:
        peso_tempo = math.exp(-0.05 * u.tempo_minutos)
        soma += u.freq * u.peso_intencao * peso_tempo

    vib_score = soma / len(usuarios) if usuarios else baseline(celula)
    return vib_score
```

Benefícios da Lógica

- **Confiável** → resistente a fraudes.
- **Dinâmico** → muda com o tempo em minutos.
- **Escalável** → pré-agregação + simplificação estatística.
- **Clareza Visual** → fácil interpretação no mapa.

✨ “O VIB_SCORE é o coração matemático do mapa — traduz caos de sinais individuais em uma batida coletiva coerente.”

◆ CAMADA 22 — JOBS DE AGREGAÇÃO COM GEOHASHING (H3/S2)

🎯 Objetivo da Camada

Garantir que o **Mapa de Frequência** consiga processar milhões de usuários em escala global sem travar. Para isso, adotamos **pré-agregação assíncrona com Geohashing (H3/S2)**, reduzindo cálculos ao vivo e otimizando a experiência do usuário.

🌐 Divisão do Mundo em Células (H3 / S2)

- O mapa é dividido em células hexagonais (H3) ou esféricas (S2).
- Cada célula tem uma **resolução** definida:
 - **Cidade / Bairro** → Resolução 6–7.
 - **Quadra / Rua** → Resolução 9–10.
- Cada usuário é associado automaticamente a uma célula com base no GPS + fuzzing.

🕒 Atualização Assíncrona (Batch Jobs)

- Jobs rodam a cada **5 minutos** (ajustável).
- Cada job processa:
 - Número de usuários ativos por célula.
 - Média de VIB_SCORE da célula (com outliers removidos).
 - Densidade vibracional (alta, média, baixa).
- Os resultados são salvos em cache (Redis + Firestore) para consultas rápidas.

🔄 Fluxo Técnico

1. **Entrada:** Check-ins dos usuários → Firestore em tempo real.
2. **Pré-processamento:** Normalização + aplicação de time-decay.
3. **Agregação:** Função assíncrona calcula médias por célula H3/S2.
4. **Cache & Banco:** Redis (hot data) + PostgreSQL (histórico).
5. **Saída:** O app lê apenas o valor agregado → rápido e escalável.

⚡ Benefícios da Estratégia

- **◆ Escalável:** milhões de usuários podem ser processados.
- **◆ Custo Reduzido:** evita cálculos pesados em tempo real.
- **◆ Experiência Fluida:** atualização parece em tempo real para o usuário.
- **◆ Granularidade Flexível:** resolução ajustada conforme zoom do mapa.

🧩 Exemplo de Pseudocódigo

```
def job_agregacao():  
    celulas = dividir_mapa_h3(resolucao=7)  
    for celula in celulas:
```

```
usuarios = get_checkins(celula)
usuarios = aplicar_time_decay(remover_outliers(usuarios))

vib_score = calcular_media(usuarios)
salvar_cache(celula.id, vib_score)
```

Estrutura de Dados Agregados

Campo	Tipo	Descrição
cell_id	String	ID da célula H3/S2
vib_score	Float	Frequência média ajustada
density	Enum (low, mid, high)	Densidade vibracional
last_update	Timestamp	Última execução do job
anomalies_flag	Boolean	Sinaliza fraude/outlier detectado

✨ “Os jobs de agregação transformam o impossível em escalável — o mundo inteiro pulsando, sem travar nem por um segundo.”

CAMADA 23 — PRIVACIDADE AVANÇADA

Objetivo da Camada

Garantir que o **Mapa de Frequência** respeite **100% a privacidade do usuário**, impedindo que sua localização real seja rastreada, mesmo em cenários de baixa densidade de dados ou ataques de reidentificação.

Geofuzzing Dinâmico

- **Problema com fuzzing fixo:** deslocamentos estáticos (ex: $\pm 50\text{m}$) ainda permitem identificar residência de um usuário se houver repetição.
- **Solução implementada:**
 - Distorção adaptativa:
 - Áreas densas \rightarrow deslocamento pequeno (30–70m).
 - Áreas residenciais \rightarrow deslocamento maior (200–500m).
 - Variação aleatória a cada envio (não repetitiva).
 - Reamostragem inteligente: nunca repetir o mesmo deslocamento em áreas de baixa densidade.

Zonas de Silêncio (Silent Zones)

- Usuário pode registrar locais privados (ex: casa, trabalho).
- Dentro dessas zonas:
 - Nenhum dado de GPS é enviado para o backend.
 - O app apenas mantém o usuário “ativo” sem emitir ponto no mapa.
- Armazenamento: todas as zonas são salvas de forma **local e criptografada** no dispositivo do usuário.

Criptografia de Localização

- Dados de check-in e localização vibracional são criptografados com:
 - **AES-256** no app antes de sair do dispositivo.
 - **TLS 1.3** em trânsito.
 - **Chaves rotacionadas** a cada 30 dias.
- Logs de auditoria armazenam apenas **hashes não reversíveis**.

Exemplo de Estrutura Técnica


Campo	Tipo	Proteção Aplicada
user_id	UUID	Hash SHA-256 + salt
geo_point	Lat/Lon	Geofuzzing dinâmico + AES-256
zone_silence	Boolean	Encriptação local
timestamp	UTC	Anonimizado por intervalos de 5 min
vib_score	Float	Dados sem localização exata

Fluxo Operacional de Privacidade

1. Usuário abre o app → GPS detecta localização.
2. Antes de enviar:
 - Verifica se está em Zona de Silêncio → se sim, bloqueia envio.
 - Se não, aplica geofuzzing dinâmico.
3. Dados criptografados → enviados via TLS.
4. Backend armazena dados agregados → nunca brutos.

Benefícios

- Usuário tem **controle total** sobre onde aparecer.
- Dados são **quebrados e inutilizáveis** mesmo em caso de vazamento.
- O sistema previne **ataques de inferência** e mantém a confiança.

 “Privacidade não é uma camada extra. É a fundação invisível que permite ao usuário vibrar no coletivo sem medo.”

CAMADA 24 — LÓGICA DE CÁLCULO DO VIB_SCORE

Objetivo da Camada

Definir com clareza matemática e técnica como o **VIB_SCORE** (indicador vibracional de zonas, usuários e locais) é calculado, garantindo **precisão, escalabilidade, antifraude e atualização em tempo aceitável para o usuário**.

Fórmula Base do VIB_SCORE

$$\text{VIB_SCORE}(\text{zona}) = \text{MédiaAparada}(\sum (f_i * w_t * w_c) / N)$$

- **f_i** → frequência vibracional do usuário *i*
- **w_t** → peso temporal (decaimento)
- **w_c** → peso de confiança (usuário validado, sem histórico de fraude)
- **N** → total de contribuições válidas na célula geográfica

Componentes Técnicos

1. Média Aparada (Trimmed Mean)

- Remove **outliers extremos** (5–10% superiores e inferiores).
- Impede que um usuário mal-intencionado distorça a média com valores irreais.

2. Decaimento Temporal (Time Decay)

- Fórmula de peso:

$$w_t = e^{(-\Delta t / \lambda)}$$

- Δt = tempo desde o check-in
- λ = constante de decaimento (ex: 30 min)
- Check-ins recentes têm muito mais impacto.

3. Peso de Confiança (Trust Weight)

- Usuários verificados (DUC/DCO) têm maior peso.
- Usuários com histórico suspeito têm peso reduzido.
- Variável dinâmica ajustada pela IA Aurah Kosmos.

Mecanismos de Antifraude

- **Detecção de Check-ins Suspeitos**
 - GPS incompatível com histórico de deslocamento.
 - Mudança brusca de cidade/país em segundos → invalidação automática.
- **Rollback Inteligente**
 - Quando um check-in é marcado como fraudulento, seu impacto é **revertido** retroativamente no VIB_SCORE da zona.
- **Cross-check com Eventos**
 - Presença validada em **eventos ou locais parceiros** tem peso maior (reduz fraude).

Exemplos de Cenários

1. Usuário A faz check-in vibracional **agora** → peso total (100%).
2. Usuário B fez check-in há 2h → peso reduzido (25%).
3. Usuário C (não verificado) com padrão suspeito → peso reduzido a 10%.
4. Usuário D em evento parceiro validado → peso dobrado (200%).

Atualização e Escalabilidade

- **Agregação por Geohash (H3/S2):**
 - Cada célula recebe sua média aparada a cada **5 minutos** em background.
- **Firestore em tempo real:**
 - Apenas pontos individuais (check-ins recentes) atualizados sem delay.
- **PostgreSQL/PostGIS:**
 - Armazena séries históricas de VIB_SCORE para análises retroativas.

Exemplo de Pseudocódigo

```
def calcular_vib_score(celula):
    dados = coletar_checkins(celula, ultimas_24h)
    dados_filtrados = remover_outliers(dados, p=0.05)
    scores = []
    for d in dados_filtrados:
        w_t = exp(-d.tempo_decorrido / lambda)
        w_c = calcular_peso_confianca(d.usuario_id)
        scores.append(d.frequencia * w_t * w_c)
    return media(scores)
```

✨ Benefícios

- **Precisão** → elimina distorções por fraude ou valores extremos.
- **Tempo Real Percebido** → atualização a cada 5 min parece instantânea.
- **Escalável** → otimizado para milhões de usuários.
- **Confiável** → rollback e pesos dinâmicos mantêm a integridade do mapa.

💡 "O VIB_SCORE não é apenas um número. É a tradução matemática da vibração coletiva em cada espaço do mundo."

◆ CAMADA 25 — INTERFACE DO MAPA (Camadas Visuais, Legendas e UX de Frequência Coletiva)

🎯 Objetivo da Camada






Definir como o **Mapa de Frequência** será exibido ao usuário, traduzindo cálculos técnicos (VIB_SCORE, estados, check-ins, clusters) em uma **experiência visual, clara, educativa e impactante**, sem comprometer privacidade.

Estrutura Visual por Camadas









1. Camada Base (Mapa Geográfico)

- Renderização com **Mapbox/Leaflet**.
- Exibição simplificada de ruas, bairros e marcos principais.
- Privacidade: sem coordenadas exatas de usuários, apenas zonas.

2. Camada de Aura Coletiva (Zonas Vibracionais)

- Cada célula **H3/S2** colorida de acordo com VIB_SCORE agregado.
 - Paleta dinâmica:
 -  Colapso (0–200)
 -  Transição Baixa (200–400)
 -  Estável Neutro (400–600)
 -  Expansão (600–800)
 -  Pico Elevado (800–1000)
 - Opacidade proporcional à densidade de usuários ativos.
3. **Camada de Usuários Individuais (Check-ins)**
- Ícones discretos (pontos pulsantes).
 - Atualizados em **tempo quase real** via Firestore.
 - Nome do usuário não exibido no mapa público → apenas no **painel privado**.
4. **Camada de Locais Parceiros**
- Ícones especiais (hexágonos com brilho).
 - Ao tocar: exibe **nome, selo vibracional, eventos ativos**.
5. **Camada de Eventos e Experiências**
- Marcadores circulares dinâmicos.
 - Cor associada ao estado vibracional médio dos participantes.
 - Eventos oficiais aparecem com **aura holográfica expandida**.

Legenda Técnica (Interface)

Elemento	Visual	Significado
 Zona Vermelha	Aura intensa vermelha	Colapso energético
 Zona Laranja	Aura média	Transição baixa
 Zona Amarela	Aura neutra	Estabilidade
 Zona Verde	Aura forte	Expansão positiva
 Zona Azul	Aura pulsante azul	Pico vibracional
 Hexágono Luminoso	Local Parceiro	Espaço validado
 Círculo pulsante	Evento	Atividade coletiva
 Ponto discreto	Check-in	Presença individual

Interação UX

- **Zoom adaptativo**
 - Zoom out → visão de clusters regionais.
 - Zoom in → detalhes de locais, eventos e check-ins próximos.
- **Tooltip sensorial**
 - Ao tocar em uma zona: mostra VIB_SCORE médio, número estimado de usuários ativos e insights da IA Aurah Kosmos.
- **Filtro de Camadas**
 - Usuário pode ativar/desativar:

- Usuários
 - Eventos
 - Locais Parceiros
 - Trilhas vibracionais (Aurah).
- **Feedback da IA Aurah Kosmos**
 - Mensagens adaptativas:

"Sua cidade está em Expansão Vibracional. Aproveite para criar conexões reais."

"Zona próxima em Colapso. Sugerimos atividades regenerativas."

Privacidade e Segurança na UI


- Nunca mostrar coordenadas exatas de usuários.
- Check-ins exibidos como **pontos flutuantes com distorção (geofuzzing)**.
- **Zonas de Silêncio** não aparecem em nenhuma camada.

Integração Técnica

- **Frontend (Flutter/React Native)** renderiza o mapa com **Mapbox SDK**.
- **Backend** entrega dados via APIs:
 - `/api/map/zones` → VIB_SCORE por célula geográfica.
 - `/api/map/checkins` → check-ins em tempo real.
 - `/api/map/events` → eventos ativos e seus estados vibracionais.
 - `/api/map/partners` → locais parceiros com status energético.

Benefício Estratégico

- O mapa traduz dados técnicos e complexos em uma **visualização clara, imersiva e inspiradora**.
- Gera **engajamento emocional** (o usuário "vê" a energia do mundo).
- É a ponte entre **ciência de dados vibracional** e **experiência mágica no app**.

 "O Mapa de Frequência é mais do que um recurso. É o espelho energético do planeta, vivo na palma da mão de cada usuário."

CAMADA 26 — SEGURANÇA E PRIVACIDADE AVANÇADA

Objetivo da Camada

Garantir que toda coleta e exibição de dados de localização no **Mapa de Frequência** seja feita com **máxima proteção de privacidade**, respeitando a vontade do usuário e assegurando conformidade com LGPD/GDPR.

Componentes de Segurança

1. Geofuzzing Dinâmico (Proteção da Localização Pessoal)

- **O que é:** deslocamento artificial da posição GPS do usuário.
- **Como funciona:**
 - Áreas de **baixa densidade populacional** → maior deslocamento (até 500m).
 - Áreas de **alta densidade** (ex: centros comerciais) → menor deslocamento (20–50m).
- **Benefício:** torna a reidentificação extremamente difícil, mesmo em clusters repetidos.

2. Zonas de Silêncio (Locais Privados do Usuário)

- **Definição:** endereços definidos manualmente (ex: casa, trabalho, escola).
- **Regras:**
 - Quando o usuário estiver dentro da zona → nenhum dado de localização é transmitido.
 - Não aparece no mapa, nem mesmo distorcido.
- **Configuração:**
 - Usuário define até 3 zonas gratuitas.
 - Premium → até 10 zonas personalizadas.

3. Criptografia e Armazenamento

- **GPS + dados vibracionais** → criptografados via **AES-256 + Hash duplo**.
- **Banco híbrido:**
 - PostgreSQL (dados históricos, logs).
 - Firestore (eventos em tempo real).
- **Política de retenção:**
 - Check-ins ativos → 24h.
 - Logs agregados → 12 meses.
 - Dados brutos → descartados após processamento.

4. Controle pelo Usuário

- Botão "🛡️ **Modo Invisível**" no mapa: usuário some instantaneamente de todas as camadas.
- Painel de Privacidade → histórico de check-ins, com opção de apagar manualmente.
- Transparência → tela mostra sempre quando e como a localização está sendo usada.

Fluxo Técnico (Resumo)

flowchart TD

```

GPS["📍 Captura de GPS"] → GF["🌀 Geofuzzing Dinâmico"]
GF → ZS["🏠 Está em Zona de Silêncio?"]
ZS -- Sim → BLOCO["🚫 Não envia dados"]
ZS -- Não → CRIPTO["🔒 Criptografia AES-256"]
CRIPTO → FIRESTORE["⚡ Firestore (tempo real)"]
CRIPTO → POSTGRES["🗄️ PostgreSQL (logs históricos)"]
FIRESTORE → MAPA["🌐 Renderização no Mapa"]
POSTGRES → ANALYTICS["📊 Agregações / VIB_SCORE"]
  
```

🧠 Compliance e Auditoria

- **LGPD/GDPR:** direito de esquecimento → usuário pode excluir todo o histórico.
- **Auditoria IA:** logs acessados apenas por **IA de Segurança** e equipe de compliance.
- **Zero Knowledge:** dados privados nunca são transmitidos em formato legível.

✨ Benefício Estratégico

- Usuário tem **confiança absoluta** para usar o mapa.
- Privacidade é garantida por design (**privacy by design**).
- Transparência e controle empoderam o usuário, sem comprometer a experiência coletiva.

💡 “O mapa é coletivo, mas a privacidade é individual. Cada batida vibracional respeita o silêncio sagrado do usuário.”

⚡ CAMADA 27 — PERFORMANCE E ESCALABILIDADE

🎯 Objetivo da Camada

Garantir que o **Mapa de Frequência** seja **rápido, fluido e escalável**, suportando milhões de usuários simultâneos sem perda de performance.

🏠 1. Estratégia de Pré-Agregação

- **Problema Atual:** cálculo em tempo real para cada ponto/local não escala.
- **Solução:** usar **geohashing** (bibliotecas H3 da Uber ou S2 do Google) para dividir o planeta em células hexagonais.

Como funciona:

1. Cada célula recebe agregações periódicas: `vib_score`, `densidade_usuarios`, `estado_vibracional`.
2. Jobs em **background** recalculam os estados a cada **5 minutos**.
3. O aplicativo consulta apenas os dados pré-agrupados → **resposta instantânea**.

🏗️ 2. Estrutura Técnica de Cálculo

Pipeline:

1. **Firestore (tempo real)** → coleta check-ins brutos.
2. **ETL em Background (Kafka + Workers Go/Python)** → processa e agrega.
3. **PostgreSQL + PostGIS** → armazena agregações históricas e zonas.
4. **Redis** → cache dos últimos cálculos para consulta imediata.

🔄 3. Redefinição de “Tempo Real”

- Para **zonas agregadas:** atualização a cada 5 minutos → percepção de fluidez.
- Para **usuários individuais no mapa:** Firestore mantém streaming em tempo real (latência <1s).
- **Experiência híbrida:** fluidez para zonas + precisão para usuários.

4. Fórmula de Cálculo Refinada

Antigo:

Média ponderada simples de check-ins.

Novo (corrigido):

$$\text{VIB_SCORE} = (\sum (\text{frequencia_usuario} * \text{peso_temporal})) / N_{\text{outliers}}$$

- **Peso temporal (time decay):** check-ins antigos perdem impacto exponencialmente.
- **Outliers:** descartados via **trimmed mean** (5% maiores e menores valores).

5. Garantias de Estabilidade

- **Escalabilidade horizontal:** workers independentes processando zonas diferentes.
- **Failover automático:** se um job falhar, outro reassume a célula.
- **Monitoramento:** métricas no Prometheus + alertas no Grafana.
- **Tolerância a latência:** consultas frontend priorizam Redis → garantem fluidez mesmo em pico.

Fluxo Técnico (Agregação de Dados)

flowchart TD

```
FIRESTORE["⚡ Firestore - Check-ins Brutos"] → KAFKA["🐦 Kafka Streams"]
KAFKA → WORKER["⚙️ Workers de Agregação (Go/Python)"]
WORKER → REDIS["⚡ Redis (Cache)"]
WORKER → POSTGRES["🗄️ PostgreSQL + PostGIS"]
REDIS → FRONTEND["📱 App Consulta em Tempo Real"]
POSTGRES → ANALYTICS["📊 Relatórios / IA Aurah Kosmos"]
```

✨ Benefício Estratégico

- **Performance garantida** → milhões de usuários simultâneos sem travar.
- **Escalabilidade global** → cada zona do planeta processada de forma independente.
- **Experiência fluida** → usuário sente dinamismo mesmo em grande escala.

💡 "Não é apenas performance. É transformar complexidade global em fluidez na palma da mão."

CAMADA 28 — LÓGICA DE CÁLCULO DO VIB_SCORE

Objetivo da Camada

Definir com **precisão matemática e escalável** como o VIB_SCORE é calculado, garantindo fidelidade energética, resiliência contra distorções e performance em produção.

1. Fórmula Base do VIB_SCORE

```
VIB_SCORE(local) =  
(  $\sum$  (freq_usuario_i * w_tempo_i) ) / N_filtrado
```

Onde:

- `freq_usuario_i` → frequência vibracional registrada pelo usuário i.
- `w_tempo_i` → peso temporal aplicado (time decay).
- `N_filtrado` → total de entradas após remoção de outliers.

2. Decaimento Temporal (Time Decay)

- Check-ins mais recentes têm peso maior.
- Modelo: **Exponential Decay Function**.

Fórmula:

```
w_tempo = e-(t_atual - t_checkin) / λ
```

- λ = fator de meia-vida (ex: 30 minutos).
- Quanto mais antigo o check-in, menor o impacto no cálculo.

3. Filtro de Outliers

- Problema: usuários extremos poderiam distorcer o mapa.
- Solução: aplicar **Trimmed Mean (5%)** → remove os 5% mais altos e mais baixos.

Exemplo:

- Frequências válidas: `[200, 210, 800, 220, 215, 50, 205]`
- Outliers removidos: `800` e `50`
- Média calculada apenas com valores consistentes.

4. Processo Operacional

1. **Entrada:** check-ins validados pelo `checkin-service`.
2. **Pré-processamento:** aplicar geohash para identificar a célula geográfica.
3. **Filtro:** eliminar outliers extremos.
4. **Aplicar time decay:** reponderar cada valor conforme tempo.
5. **Cálculo:** gerar média ajustada da célula.
6. **Armazenamento:** salvar resultado em Redis (cache) e PostgreSQL (histórico).

5. Pseudocódigo Técnico

```
def calcular_vib_score(checkins, tempo_atual, lambda_decay=30):  
    # Remove outliers (5% mais altos e mais baixos)  
    checkins_filtrados = trimmed_mean_filter(checkins, percent=5)
```

```
soma_pesada = 0
peso_total = 0

for c in checkins_filtrados:
    w_tempo = exp(-(tempo_atual - c.timestamp) / lambda_decay)
    soma_pesada += c.frequencia * w_tempo
    peso_total += w_tempo

vib_score = soma_pesada / peso_total if peso_total > 0 else 0
return vib_score
```

6. Garantias Técnicas

- **Resiliência:** outliers não distorcem.
- **Dinamismo:** zonas reagem rápido a mudanças recentes.
- **Escalabilidade:** cálculo aplicado em batches agregados (não ponto a ponto).
- **Transparência:** logs armazenam todos os valores usados → auditoria possível.

✨ Benefício Estratégico

- **Mapa mais realista:** reflete o agora, não o passado.
- **Confiança:** usuários não veem distorções causadas por valores extremos.
- **Segurança de Dados:** todos os cálculos podem ser auditados em caso de fraude.

💡 “O VIB_SCORE não é apenas um número. É a métrica pulsante que traduz a energia coletiva em linguagem matemática confiável.”

CAMADA 29 — PRIVACIDADE E ZONAS DE SILÊNCIO

Objetivo da Camada

Garantir que o **Mapa de Frequência respeite a privacidade absoluta do usuário**, evitando exposição indevida de dados sensíveis e permitindo que cada pessoa defina **limites de visibilidade** sobre sua energia, localização e histórico vibracional.

1. Mecanismos de Privacidade Ativos

Mecanismo	Descrição Técnica	Benefício para o Usuário
Geofuzzing Dinâmico	Distorção adaptativa da localização: ±50m em áreas urbanas densas, até ±500m em regiões residenciais.	Impede a inferência de endereços privados.
Anonimização em Massa	Em zonas de baixa densidade (poucos usuários), o mapa só mostra estado vibracional agregado.	Protege usuários em cidades pequenas ou bairros vazios.
Consentimento Granular	O usuário escolhe: visível no mapa, visível apenas para conexões, ou invisível.	Liberdade total de controle.
Check-ins Invisíveis	Possibilidade de registrar presença energética sem aparecer publicamente.	Mantém impacto no mapa sem exposição social.

2. Zonas de Silêncio

- O usuário pode cadastrar **endereços privados** (casa, trabalho, escola etc.).
- Quando dentro dessas zonas:
 - Nenhum dado de localização é enviado, nem mesmo com geofuzzing.
 - O sistema aplica **silenciamento total** (zero logs, zero sinais).
- Exemplo prático:
 - Usuário marca sua casa como zona de silêncio → sempre que estiver lá, o app entra em modo privado, sem envio de dados.

3. Fluxo Técnico das Zonas de Silêncio

flowchart TD

```
A[GPS Detecta Localização] → B{Zona definida como Silêncio?}
B -- Sim → C[Não envia dados de localização]
C → D[Status no mapa = "Silenciado"]
B -- Não → E[Aplica Geofuzzing Dinâmico]
E → F[Envia dados agregados para célula geográfica]
```

4. Pseudocódigo Operacional

```
def processar_localizacao(user, localizacao):
    if localizacao in user.zonas_silencio:
        return {"status": "silenciado", "enviar": False}


    localizacao_fuzz = aplicar_geofuzzing(localizacao, densidade_area(localizacao))
    return {"status": "ativo", "localizacao": localizacao_fuzz}
```

5. Logs e Segurança

- Nenhum dado de zonas de silêncio é persistido em banco.
- Todas as regras são processadas **localmente no device** antes do envio.
- Auditoria só registra que o **usuário estava em silêncio**, nunca o endereço real.

Benefício Estratégico

- **Proteção Total:** o usuário sente confiança em usar o mapa sem medo de exposição.
- **Escalabilidade Ética:** evita problemas legais (LGPD, GDPR) e constrói confiança global.
- **Experiência Consciente:** o usuário escolhe quando e como aparecer no radar coletivo.

 “O Mapa de Frequência não é uma vigilância, é um espelho. E todo espelho deve respeitar os momentos em que o reflexo não quer ser visto.”

CAMADA 30 — EXPERIÊNCIA DO USUÁRIO (UX) E PREVENÇÃO DE MAPA VAZIO

Objetivo da Camada

Garantir que o **Mapa de Frequência nunca apareça “morto” ou vazio**, mesmo em cidades com poucos usuários ativos. O objetivo é **preservar a sensação de um radar vivo** e manter a experiência mágica do FriendApp, sem comprometer a veracidade técnica.

1. Problema do Mapa Vazio (Cold Start)

- Em regiões com poucos usuários ou check-ins, o mapa pode aparecer **sem cores, pontos ou pulsos energéticos**.
- Esse “silêncio visual” causa frustração e transmite a impressão de que **o app não funciona ali**.

2. Solução: Camada Base Heurística

- Criar uma **camada de vibração base** alimentada por heurísticas e dados públicos.
- Mesmo sem usuários ativos, a cidade exibirá cores e estados energéticos plausíveis.

Exemplos de heurísticas:

Local	Vibração Base	Justificativa
Parques	Neutro-Positivo 🌱	Associados a lazer e bem-estar.
Áreas Comerciais	Expansão ☀️	Alto movimento de pessoas e trocas.
Áreas Residenciais	Neutro 🌙	Ambiente estável, sem grandes fluxos.
Espaços Culturais	Elevação 🎵	Locais de encontro, arte e cultura.
Zonas Críticas (histórico de problemas)	Transição / Colapso ⚠️	Dados públicos de segurança e densidade.

3. Camadas Técnicas de Implementação

- Fallback de Dados:** quando não houver dados vibracionais reais, o sistema exibe a camada heurística.
- IA Aurah Kosmos:** ajusta a cor base conforme contexto cultural, horário ou eventos públicos locais.
- Pré-agregação por Células (H3/S2):** mesma arquitetura usada para dados reais, garantindo consistência visual.

4. Fluxo Técnico

flowchart TD

```
A[Usuário abre o mapa] → B{Dados reais disponíveis?}
B -- Sim → C[Renderizar pulsos reais]
B -- Não → D[Aplicar camada heurística]
D → E[Aurah Kosmos ajusta cor/contexto]
C & E → F[Mapa vivo exibido ao usuário]
```

5. Pseudocódigo

```
def obter_estado_mapa(zona):
    dados_reais = consultar_cache(zona)

    if not dados_reais:
        estado = heuristica_base(zona.tipo_local)
```

```
return {"status": "heurístico", "estado": estado}


return {"status": "real", "estado": calcular_vib_score(dados_reais)}
```

6. Experiência do Usuário

- O usuário **nunca verá um mapa vazio**.
- Em cidades novas, terá a sensação de que já existe um **campo vivo em andamento**.
- Assim que novos usuários chegarem, os **dados reais começam a se sobrepor gradualmente** à camada heurística.

7. Benefício Estratégico

- **Elimina frustração inicial (cold start)**.
- **Passa sensação de radar vivo e global**.
- **Expansão mais rápida em novos mercados**, pois o app já parece ativo mesmo em locais com baixa densidade inicial.

 “O Mapa nunca dorme. Mesmo quando não há ninguém olhando, ele pulsa com o potencial do que pode vir.”

CAMADA 31 — MECANISMOS CONTRA FRAUDES E REVERSÃO DE CHECK-INS SUSPEITOS


Objetivo da Camada


Garantir que **check-ins fraudulentos ou maliciosos** não distorçam a energia do Mapa de Frequência. Esta camada define mecanismos de **detecção automática, reversão de impacto e penalização de usuários**, preservando a integridade vibracional coletiva.

1. Tipos de Fraudes Detectadas

Tipo de Fraude	Descrição	Exemplo Real
GPS Spoofing	Usuário simula estar em local diferente	Apps de falsificação de GPS
Check-in Automático	Bots ou scripts enviam múltiplos check-ins	Requisições automatizadas via API
Check-in Massivo	Grupo coordenado distorce energia de uma zona	Ataque em área específica
Check-in Incompatível	Vibração declarada não corresponde ao padrão histórico do usuário	Usuário “força” score positivo sem coerência

2. Mecanismos de Detecção

- **Validação Técnica:**
 - GPS + Wi-Fi triangulação + tempo de deslocamento plausível.
 - Se tempo entre check-ins for inferior ao tempo físico mínimo →  fraude.
- **Validação Energética (Aurah Kosmos):**
 - IA compara o padrão vibracional histórico do usuário com o check-in atual.

- Incompatibilidades gritantes →  suspeita.
- **Análise Comportamental:**
 - Detecção de padrões repetitivos (ex: check-ins de 1 em 1 minuto).
 - Comparação com clusters normais de comportamento.
- **Cross-Validation Comunitária:**
 - Múltiplos usuários presentes no mesmo local contradizem um check-in suspeito.

3. Processo de Reversão de Impacto

Fluxo Operacional:

```

flowchart TD
    A[Check-in recebido] --> B{Suspeito?}
    B -- Não --> C[Aplica no Mapa]
    B -- Sim --> D[Marcar como inválido]
    D --> E[Reverter impacto no VIB_SCORE]
    E --> F[Atualizar estado do Mapa]
    D --> G[Registrar no Log de Fraudes]
    G --> H[Aplicar penalidade ao usuário]
  
```

Técnicas:

- **Rollback Automático:**
 - Se detectado como fraudulento → recalcula VIB_SCORE sem aquele dado.
- **Reversão Retroativa:**
 - Impacto já propagado é revertido em background job.
 - IA Aurah Kosmos recalibra estados de zona.

4. Penalidades e Proteções

Tipo de Infração	Penalidade
GPS Spoofing (1ª vez)	Alerta educativo + bloqueio de check-in por 24h
GPS Spoofing (reincidência)	Redução de alcance no feed + suspensão de 7 dias
Check-ins automáticos	Bloqueio definitivo de API Key
Fraudes em massa	Banimento coletivo e alerta à equipe de segurança

5. Pseudocódigo Simplificado

```

def validar_checkin(usuario, localizacao, vib_score):
    if not deslocamento_plausivel(usuario, localizacao):
        marcar_fraude(usuario, "gps_spoofing")
        return False

    if comportamento_suspeito(usuario):
        marcar_fraude(usuario, "pattern_anomalo")
        return False
  
```

```

if not coerencia_vibracional(usuario, vib_score):
    marcar_fraude(usuario, "incompatibilidade_energetica")
    return False

return True

```

6. Logs e Observabilidade

- Todo check-in suspeito gera entrada no **FraudLog**.
- Logs armazenados com:
 - `id_user`, `localizacao`, `tipo_suspeita`, `acao_tomada`, `timestamp`.
- Integração com **Painel de Segurança Vibracional** para auditoria em tempo real.

✨ 7. Benefício Estratégico

- **Protege a integridade do Mapa.**
- **Impede manipulações externas.**
- **Aumenta a confiança dos usuários**, mostrando que o FriendApp é seguro e justo.

💡 "No radar vibracional, apenas frequências autênticas sobrevivem. Toda fraude se dissolve na malha protetora do campo."



CAMADA 32 — PRIVACIDADE AVANÇADA

Objetivo da Camada

Assegurar que toda coleta e exibição de dados de localização no **Mapa de Frequência** respeite a **privacidade absoluta do usuário**, combinando **algoritmos adaptativos**, **zonas de silêncio** e **criptografia forte**, em conformidade com LGPD/GDPR.

1. Geofuzzing Dinâmico

- **Definição:** deslocamento intencional e adaptativo da posição GPS antes do envio.
- **Regras Dinâmicas:**
 - Áreas densas (centros, eventos): deslocamento pequeno (20–70m).
 - Áreas residenciais / baixa densidade: deslocamento maior (100–500m).
- **Seed de Sessão:** deslocamentos pseudo-aleatórios estáveis por sessão → impede rastreamento por repetição.
- **Snap Público:** quando possível, o ponto final é "magnetizado" para locais públicos (praças, ruas, POIs).

Pseudocódigo simplificado:

```

def aplicar_geofuzzing(lat, lon, densidade, contexto, seed):
    if contexto == "zona_residencial":
        raio = random(100, 500)
    else:
        raio = random(20, 70)

```

```
return deslocar(lat, lon, raio, seed)
```

2. Zonas de Silêncio

- **Configuração pelo usuário:** até 3 zonas gratuitas (ex.: casa, trabalho); até 10 para Premium.
- **Funcionamento:**
 - Se dentro de uma zona → **nenhum dado de localização é enviado**.
 - Nem mesmo fuzzed → é silenciamento completo.
- **Persistência:**
 - Armazenadas criptografadas no device e backend (**AES-256** + hash).
 - Nunca exibidas no mapa, nem para admins.

3. Criptografia de Localização

- **Em trânsito:** TLS 1.3 obrigatório.
- **Em repouso:** AES-256 + rotação de chaves (90 dias).
- **Logs de auditoria:** apenas armazenam hash irreversível (**SHA-256**) do check-in → impossível reconstruir coordenadas reais.

4. Estrutura Técnica de Dados

Campo	Tipo	Proteção
user_id	UUID	Hash irreversível
lat / lon	Float	Geofuzzing dinâmico + AES
zone_silence	Boolean	Nunca transmitido
timestamp	UTC	Arredondado a intervalos de 5 min
vib_score	Int (0-1000)	Dado vibracional (não sensível)

5. Fluxo Operacional


flowchart TD

```
GPS["📍 Captura de GPS"] --> ZS{"🏠 Está em Zona de Silêncio?"}
ZS -- Sim --> BLOCO["🚫 Bloqueia envio"]
ZS -- Não --> FZ["🌀 Aplica Geofuzzing Dinâmico"]
FZ --> CRIPTO["🔒 Criptografia AES-256"]
CRIPTO --> FIRESTORE["⚡ Firestore (tempo real)"]
CRIPTO --> POSTGRES["🗄️ PostgreSQL (histórico agregado)"]
```

6. Compliance & Auditoria

- **LGPD/GDPR:** usuário pode solicitar exclusão ("direito de ser esquecido").
- **Consentimento:** registrado no **consent_ledger** (**GRANTED** ou **DENIED**).
- **Auditoria:** logs imutáveis (WORM storage) armazenam apenas ações de consentimento, nunca coordenadas brutas.

7. UX de Privacidade

- Botão “ **Modo Invisível**” → usuário some do mapa em tempo real.
- Painel de privacidade → visualizar e apagar histórico.
- Transparência → app mostra sempre se está “visível, coletivo ou silenciado”.

✨ Benefício Estratégico

- Garante **segurança jurídica global** (LGPD/GDPR).
- Cria **confiança plena no usuário** para participar do mapa.
- Dá controle: cada um decide como aparece, quando aparece, e se aparece.

💡 “O Mapa de Frequência vibra coletivamente, mas respeita o silêncio individual. Privacidade não é extra: é o alicerce invisível da confiança.”

CAMADA 33 — COLD START (Mapa Base Heurístico)

Objetivo da Camada

Impedir que o usuário veja um **mapa “vazio” ou inativo** em cidades/regiões com pouca ou nenhuma atividade inicial. Essa camada garante que o Mapa de Frequência **sempre mostre vida**, mesmo sem usuários ativos no momento, preservando a experiência de radar global.

1. Problema do Cold Start

- Em áreas novas ou com baixa densidade de usuários, o mapa poderia aparecer sem cor ou pulsos energéticos.
- Essa ausência transmite a sensação de que o **app não funciona** naquela região.

💡 2. Solução: Camada Heurística de Frequência Base

- Quando não houver dados vibracionais suficientes, o sistema usa **heurísticas baseadas em tipo de local** para gerar **um estado inicial neutro ou plausível**.
- Isso cria a sensação de **radar vivo**, mesmo sem dados reais.

3. Heurísticas Aplicadas

Tipo de Local	Estado Base	Justificativa
Parques / Áreas Verdes	Expansão Leve (600–650)	Associado a lazer, natureza e bem-estar
Zonas Comerciais	Estável/Transição Positiva (500–600)	Movimento constante de pessoas
Áreas Residenciais	Neutro (400–500)	Ambientes estáveis, baixa variação
Centros Culturais	Expansão Vibracional (650–700)	Locais de troca, criatividade e fluxo social
Zonas Críticas (alta criminalidade)	Colapso (200–300)	Dados públicos de segurança urbana
Áreas Sem Dados	Neutro (450)	Valor de fallback padrão

4. Fluxo Operacional

flowchart TD

U[Usuário abre o mapa] → Q{Há dados vibracionais reais?}

Q -- Sim → R[Renderizar dados reais]

Q -- Não → H[Aplicar heurística por tipo de local]

H → AURAH[Aurah Kosmos ajusta nuances]

R & AURAH → M[Mapa exibido ao usuário]

5. Papel da IA Aurah Kosmos


- Ajusta a heurística em tempo real:
 - Se for final de semana → parques ganham energia extra.
 - Se houver evento cultural público → locais culturais passam para Expansão.
 - De madrugada → áreas comerciais regridem para Neutro/Transição.

6. Estrutura Técnica

- **Tabela** `baseline_zones` em PostgreSQL:
 - `zone_id` (H3/S2)
 - `tipo_local`
 - `vib_score_base`
 - `ajustavel_por_IA` (boolean)
- Dados carregados automaticamente em Redis para resposta rápida.

7. Benefícios

- Usuário **nunca vê o mapa morto** → sempre há vida visível.
- Mantém **imersão e credibilidade** do app desde o primeiro uso.
- Apoiar a **expansão internacional**, garantindo experiência uniforme em qualquer cidade.

 *"O Mapa nunca está vazio. Mesmo onde ninguém vibra ainda, ele mostra o potencial do espaço — como se já respirasse em expectativa do que virá."*

CAMADA 34 — ROLLBACK & REVERSÃO DE FRAUDES

Objetivo da Camada

Definir como o sistema **identifica, isola e reverte** check-ins fraudulentos ou maliciosos que já impactaram o Mapa de Frequência, preservando a integridade do **VIB_SCORE** coletivo e garantindo transparência nos logs.

1. Cenários de Fraude

Tipo de Fraude	Exemplo	Impacto no Mapa
GPS Spoofing	Usuário simula estar em outro país via app fake	Distorce zonas em áreas críticas
Check-in Automático (Bot)	Requisições contínuas por script	Infla artificialmente VIB_SCORE
Fraude em Massa	Grupo coordenado faz check-in falso em célula	Cria "picos" irreais de energia

Tipo de Fraude	Exemplo	Impacto no Mapa
Score Incompatível	Usuário envia vibração 1000 sem coerência	Polui média e manipula clusters

2. Detecção

- **Validação de Localização:** GPS + Wi-Fi + torre de celular → divergência = ►
- **Validação Temporal:** distância incompatível em tempo curto = ►
- **IA Aurah Kosmos:** análise de padrões históricos do usuário (energia incoerente = ►)
- **Detecção de Anomalias:** média da célula sobe >50% em 1 min → ►

3. Fluxo de Rollback

flowchart TD

```

A[Check-in recebido] → B{Suspeito?}
B -- Não → C[Validação concluída → Mantém impacto]
B -- Sim → D[Marca check-in como inválido]
D → E[Remove contribuição do VIB_SCORE]
E → F[Recalcula célula e vizinhos]
D → G[Loga no sistema de fraudes]
G → H[Aplica penalidade ao usuário]

```

4. Técnicas de Reversão

1. Rollback Imediato (Tempo Real)

- Remove check-in suspeito antes de propagar.

2. Rollback Retroativo (Background Job)

- Caso o impacto já tenha sido refletido, recalcula zona + vizinhos.
- Atualiza Redis (cache) e PostgreSQL (histórico).

3. Rollback Coletivo

- Em fraude em massa → recalcular célula inteira ignorando usuários maliciosos.

5. Pseudocódigo Técnico

```

def aplicar_rollback(checkin):
    if checkin.flag_fraude:
        zona = identificar_celula(checkin.localizacao)
        remover_contribuicao(zona, checkin.user_id)
        vib_score_atualizado = recalcular_vib_score(zona)
        atualizar_cache(zona, vib_score_atualizado)
        registrar_log_fraude(checkin.user_id, zona, vib_score_atualizado)

```

6. Logs e Auditoria

- Logs de fraudes armazenados em **FraudLog**:
 - `user_id` (hash)

- `zona` (H3)
 - `tipo_fraude`
 - `timestamp`
 - `ação_tomada`
- Auditoria só acessa dados anonimizados.

7. Penalidades Automáticas

Infração	Penalidade
GPS Spoofing (1x)	Alerta + bloqueio de check-in por 24h
Reincidência	Suspensão 7 dias
Fraude em Massa	Banimento + bloqueio da API Key
Score falso recorrente	Shadowban (usuário vê mapa, mas não impacta)

✨ Benefício Estratégico

- **Mantém integridade** → evita manipulações no radar coletivo.
- **Transparência** → logs imutáveis permitem auditoria.
- **Confiabilidade** → garante que energia no mapa sempre reflete o real.

💡 "Toda fraude é dissolvida no campo. O mapa só preserva vibrações autênticas."

CAMADA 35 — INTEGRAÇÕES E DEPENDÊNCIAS CÍCLICAS DO ECOSISTEMA

Objetivo da Camada

Mapear **todas as integrações do Mapa de Frequência** com os outros módulos do FriendApp, mostrando **de onde vêm os dados, para onde vão, e como eles são processados**. Essa visão garante que os devs compreendam as dependências e os ciclos de retroalimentação do ecossistema.

Integrações de Entrada (Dados que alimentam o Mapa)

Origem	Tipo de Dado	Método
Cadastro Consciente	Perfil inicial do usuário + status de verificação	API interna <code>/usuario/core</code>
Teste de Personalidade Energética	Vetor vibracional base do usuário	API <code>/perfil/vibracional</code>
Aurah Kosmos	Análises preditivas + curadoria vibracional	Streaming + triggers
Sistema de Eventos	Check-ins coletivos em eventos ativos	Trigger <code>/eventos/checkin</code>
Locais Parceiros	Estado energético de estabelecimentos	API <code>/locais/estado</code>
Feed Sensorial	Interações e emoções compartilhadas	Trigger <code>/feed/reacao</code>
Gamificação & Jogo da Transmutação	Progressos energéticos de usuários	API <code>/gamificacao/logs</code>

Integrações de Saída (Dados gerados pelo Mapa)

Destino	Dado Enviado	Uso Posterior
Aurah Kosmos	Estados vibracionais agregados	IA usa para recomendações e alertas

Destino	Dado Enviado	Uso Posterior
Feed Sensorial	Destaques de zonas em Pico ou Colapso	Aparece como insight no feed
Eventos & Experiências	Estado vibracional do local do evento	Ajuda usuários a decidir participação
Locais Parceiros	Frequência energética real-time	B2B: usado para métricas e impulsionamento
Painel de Reputação Vibracional	Histórico de presença em zonas	Aumenta ou reduz score do usuário
Gamificação & FriendCoins	Recompensas por check-ins vibracionais	Estimula engajamento
Sistema Premium	Camadas adicionais de visualização	Exclusivo para usuários pagos
Logs & Observabilidade	Métricas de vibração + fraudes	Painel admin + auditoria LGPD

Triggers e Eventos do Mapa

- `onCheckinVibracional` → dispara atualização em célula.
- `onZoneUpdate` → publica novo VIB_SCORE agregado.
- `onFraudeDetectada` → rollback + alerta IA Aurah Kosmos.
- `onZonaPico` → notificação no Feed Sensorial.
- `onZonaColapso` → trigger de segurança vibracional.

Dependências Estruturais

Sistema	Tipo de Dependência
Aurah Kosmos (IA Central)	Curadoria e interpretação preditiva
Firestore	Presença individual em tempo real
PostgreSQL + PostGIS	Histórico de zonas e agregados
Redis	Cache das células agregadas
Gamificação	Vincula ações a XP e FriendCoins
Segurança Vibracional	Ativa bloqueios e proteções em zonas críticas

Fluxo Cíclico do Ecossistema

flowchart TD

```

Cadastro → PerfilVib
PerfilVib → Mapa
Feed → Mapa
Eventos → Mapa
Locais → Mapa
Mapa → Aurah
Mapa → Feed
Mapa → Eventos
Mapa → Locais
Mapa → Reputacao
Mapa → Gamificacao
Mapa → Premium

```

Benefício Estratégico

- O Mapa não é isolado → **é o coração pulsante** do FriendApp.

- Cada ação reverbera em múltiplos sistemas, criando **ciclos de feedback vivo**.
- Devs têm clareza sobre **todas as entradas, saídas e dependências críticas**.

💡 "O Mapa de Frequência não recebe nem envia dados. Ele respira com todo o ecossistema, como um coração que pulsa no ritmo da comunidade."

📦 CAMADA 36 — FECHAMENTO TÉCNICO

🎯 Objetivo da Camada

Consolidar toda a arquitetura, cálculos, integrações e mecanismos de segurança do **Mapa de Frequência**, mostrando aos devs, gestores e investidores que este sistema é **executável, escalável e auditável**.

🧩 1. Síntese Arquitetural

- **Entrada de Dados:** check-ins individuais → Firestore (tempo real).
- **Pré-Agregação:** jobs em background → Redis (cache) + PostgreSQL (histórico).
- **Cálculo Vibracional:** VIB_SCORE robusto → média aparada + time-decay + antifraude.
- **Segurança:** geofuzzing dinâmico + zonas de silêncio + criptografia.
- **UX:** mapa nunca vazio → camada heurística (Cold Start).
- **Rollback:** fraudes revertidas em até 2 min (p95).
- **Observabilidade:** logs imutáveis, métricas (Prometheus), alertas (Grafana).

⚙️ 2. Garantias Técnicas

Área	Garantia	Métrica/SLO
Performance	Latência consultas de zonas	< 200ms (Redis)
Escalabilidade	Suporte a milhões de usuários	Arquitetura horizontal (workers + geohash)
Privacidade	LGPD/GDPR compliance	Geofuzzing, zonas de silêncio, direito de exclusão
Antifraude	Reversão rápida de manipulações	Rollback < 2 min
Confiabilidade	Disponibilidade do serviço	99,95% uptime
Experiência	Nunca mapa vazio	Camada heurística global

🔗 3. Dependências Críticas

- **Aurah Kosmos (IA Central)** → ajustes dinâmicos e curadoria vibracional.
- **Eventos/Locais Parceiros** → alimentam e recebem dados energéticos.
- **Feed Sensorial** → reflete zonas em Pico ou Colapso.
- **Gamificação & FriendCoins** → recompensa ações vibracionais.
- **Segurança Vibracional** → protege zonas críticas e aplica bloqueios.

📊 4. Valor Estratégico

- **Para usuários:** experiência mágica, confiável e respeitosa com privacidade.
- **Para devs:** manual executável, com fórmulas, APIs, pseudocódigos e contratos claros.
- **Para investidores:** escalabilidade global, compliance legal e diferencial competitivo único.

✨ Fechamento Final

“O Mapa de Frequência é o coração vivo do FriendApp.

Ele respira os check-ins individuais, pulsa em clusters coletivos, se protege contra fraudes e preserva o silêncio onde é preciso.

Com ele, o invisível se torna visível, o dado vira energia, e a energia se transforma em confiança.”