Manual Técnico Definitivo — Feed Sensorial

🔷 CAMADA 1 — PROPÓSITO TÉCNICO & VISÃO GERAL DO FEED SENSORIAL

📍 **Entrada**

“O Feed Sensorial do FriendApp não é uma timeline social. Ele é um **motor de ressonância**, projetado para organizar conteúdos com base em compatibilidade energética entre usuários e postagens.”

🎯 Objetivo da Camada

Fornecer a definição **técnica e funcional** do Feed Sensorial, estabelecendo:

- **Escopo**: substitui o feed cronológico e de engajamento por um sistema baseado em compatibilidade vibracional.
- **Finalidade**: conectar usuários a conteúdos que aumentam sua frequência energética.
- **Diferencial**: personalização em tempo real com IA, sem depender de popularidade ou curtidas.

⚙️ Estrutura Geral do Sistema

Componentes Principais

Componente	Função Técnica
Aurah Kosmos (IA)	Analisa cada post e usuário, gerando vetores energéticos comparáveis.
Módulo de Vetorização	Converte textos, imagens, áudios e vídeos em representações numéricas (post_vector).
Ranking Multi-Stage	Ordena postagens em três estágios: geração de candidatos, ranking leve e ranking pesado.
Firewall Vibracional	Rotula posts por intensidade emocional (leve, intenso, colapso) e permite filtragem pelo usuário.

Componente	Função Técnica
Sistema de Logs	Registra frequência inicial, impacto e expansão vibracional de cada post.

🔢 Definição de Feed Sensorial

- Cada **usuário** possui um `user_vector` (perfil energético em tempo real).
- Cada **postagem** possui um `post_vector` (conteúdo transformado em vetor semântico + emocional).
- O **feed** é gerado comparando `user_vector` e `post_vector` com base em **similaridade de cossenos**.
- Postagens são filtradas por regras de moderação, cacheadas por 3–5 minutos e exibidas em ordem de **score de ressonância**.

🧠 Fórmula Base do Score de Ressonância

Score(post,user)=(0.4∗cosinesimilarity(uservector,postvector))+(0.3∗intenc\ca~opost)+(0.2∗impactocoletivo)+(0.1∗freshness)Score(post, user) =
  (0.4 * cosine_similarity(user_vector, post_vector)) +
  (0.3 * intenção_post) +
  (0.2 * impacto_coletivo) +
  (0.1 * freshness)

Score(post,user)=(0.4∗cosinesimilarity(uservector,postvector))+(0.3∗intenc\ca~opost)+(0.2∗impactocoletivo)+(0.1∗freshness)

- `cosine_similarity` → mede compatibilidade entre usuário e conteúdo.
- `intenção_post` → peso extra para posts com tags positivas (curar, inspirar, alegrar).
- `impacto_coletivo` → valor derivado de reações vibracionais do coletivo.
- `freshness` → tempo de vida da postagem (evita congelamento).

📦 Banco de Dados — Estrutura Inicial (posts)

Campo	Tipo	Descrição
`post_id`	UUID	Identificador único

Campo	Tipo	Descrição
`user_id`	UUID	Autor
`post_vector`	Array[Float]	Vetor energético da postagem
`intencao`	String	Intenção declarada pelo usuário
`score_ressonancia`	Float	Valor calculado da fórmula
`firewall_label`	Enum	Leve, Intenso, Colapso
`created_at`	Timestamp	Data de criação

🔚 Fechamento

“A fundação está estabelecida: o Feed Sensorial é um motor de cálculo energético. Na próxima camada, descrevemos como conteúdos de diferentes formatos são **vetorizados** para alimentar esse motor.”

🔷 CAMADA 2 — VETORIZAÇÃO DE CONTEÚDO (TEXTO, IMAGEM, ÁUDIO, VÍDEO)

📍 Entrada

“O coração do Feed Sensorial é a capacidade de transformar qualquer postagem em dados matemáticos comparáveis. Essa camada define como cada tipo de conteúdo é convertido em vetores energéticos (post_vector), permitindo cálculos objetivos de compatibilidade.”

🎯 Objetivo da Camada

Detalhar o processo de **vetorização multimodal** de conteúdos, explicando como a IA Aurah Kosmos traduz texto, imagens, áudios e vídeos em **representações numéricas de alta dimensão**. Esses vetores são a base do cálculo de ressonância.

🧠 Estrutura de Vetorização

Tipo de Conteúdo	Técnicas Utilizadas	Saída (post_vector)
Texto	NLP + embeddings semânticos (ex: transformer encoder)	Vetor 512D
Imagem	CNN pré-treinada + análise de cor/expressão facial	Vetor 1024D
Áudio	FFT + espectrograma + embeddings de emoção sonora	Vetor 256D
Vídeo	Extração de frames + áudio + narrativa	Vetor 1536D (concatenação multimodal)

📊 Pipeline Técnico de Vetorização

```
function vetorizar_post(post):
    if post.tipo == "texto":
        vector = NLP_encoder(post.texto)
    elif post.tipo == "imagem":
        vector = CNN_extractor(post.imagem)
    elif post.tipo == "audio":
        vector = Audio_embedding(post.audio)
    elif post.tipo == "video":
        frame_vec = CNN_extractor(frames)
        audio_vec = Audio_embedding(audio)
        vector = concat(frame_vec, audio_vec)

    normalizado = normalize(vector)
    return normalizado
```

🔢 Fórmula da Compatibilidade

Com `user_vector` e `post_vector` definidos, a compatibilidade é calculada como:

Compatibilidade=cosinesimilarity(uservector,postvector)Compatibilidade = cosine_similarity(user_vector, post_vector)

Compatibilidade=cosinesimilarity(uservector,postvector)

- Valor entre `1` e `1`, onde:

- **1.0** → máxima ressonância
- **0.0** → neutro (sem relação)
- **1.0** → ressonância oposta (contraste energético)

📦 Banco de Dados — Campos Técnicos

Campo	Tipo	Descrição
`post_vector`	Array[Float]	Representação numérica do post
`user_vector`	Array[Float]	Vetor vibracional do usuário
`similaridade`	Float	Resultado do cálculo de cosine similarity
`intencao`	String	Intenção declarada pelo autor
`rotulo_IA`	String	Classificação automática (leve, intenso, colapso)

🔬 Exemplo de Saída Técnica

```
{
  "post_id": "p123",
  "user_id": "u456",
  "tipo": "texto",
  "post_vector": [0.12, 0.88, -0.44, ...],
  "user_vector": [0.15, 0.80, -0.47, ...],
  "similaridade": 0.94,
  "rotulo_IA": "expansao"
}
```

📌 Notas Técnicas Importantes

- Vetores são recalculados em cada interação do usuário (mudança de aura/estado).
- Para performance, vetores são armazenados em **FAISS ou Milvus** (banco vetorial) para consultas rápidas em escala.
- Similaridade de cossenos é o cálculo principal, mas pode ser complementado por métricas como **dot product** ou **distância euclidiana normalizada** em experimentos.

🔚 Fechamento

“A partir da vetorização multimodal, todo conteúdo se torna um vetor mensurável. Na próxima camada, detalhamos como esses vetores são usados no **Ranking Multi-Stage**, garantindo performance e escalabilidade.”

🔷 CAMADA 3 — RANKING MULTI-STAGE (CANDIDATE, LEVE, PESADO + CACHING)

📍 Entrada

“Gerar um feed personalizado em tempo real para milhões de usuários exige mais do que fórmulas — é necessária uma arquitetura escalável. O ranking multi-stage garante performance, precisão e baixo custo computacional.”

🎯 Objetivo da Camada

Definir a arquitetura de ranqueamento em múltiplos estágios (multi-stage ranking), usada em sistemas de escala (Instagram, TikTok, YouTube) e adaptada ao FriendApp para cálculo de **Score de Ressonância**.

⚙️ Estrutura em 3 Estágios

Estágio	Descrição Técnica	Tamanho Médio
Candidate Generation	Seleção inicial de milhares de posts relevantes (conexões, região, afinidade básica)	~5.000 posts
Ranking Leve	Algoritmo rápido ordena candidatos usando sinais simples (tempo, engajamento leve)	~500 posts
Ranking Pesado	Aplicação do Score de Ressonância completo (cosine similarity + intenção + impacto)	~50 posts

🔄 Pipeline Multi-Stage

```
function gerar_feed(usuario):
    candidatos = buscar_candidatos(usuario) # Estágio 1
```

```
    candidatos_rankeados = ranking_leve(candidatos) # Estágio 2
    feed_final = ranking_pesado(candidatos_rankeados) # Estágio 3
    cache.store(usuario.id, feed_final, ttl=300s)
    return feed_final
```

🧠 Ranking Leve (Sinais Simples)

Sinal	Peso (%)	Observação
Recência	40%	Posts mais recentes
Afinidade (tags comuns)	30%	Interesses compatíveis
Popularidade local	20%	Visualizações e ressonância leve
Tipo de conteúdo	10%	Preferências de mídia (texto/vídeo)

🧮 Ranking Pesado (Score Completo)

Usa a fórmula definida na **Camada 1**:

Score=(0.4∗cosinesimilarity)+(0.3∗intenc\ca~opost)+(0.2∗impactocoletivo)+(0.1∗freshness)Score = (0.4 * cosine_similarity) +
    (0.3 * intenção_post) +
    (0.2 * impacto_coletivo) +
    (0.1 * freshness)

Score=(0.4∗cosinesimilarity)+(0.3∗intenc\ca~opost)+(0.2∗impactocoletivo)+(0.1∗freshness)

⚡ Estratégia de Cache

- Cada feed personalizado é salvo no **Redis Cache** por **3–5 minutos (TTL)**.
- Isso evita recomputar a cada abertura do app.
- Atualizações em tempo real (ex: novo post de conexão próxima) disparam **invalidação seletiva**.

📦 Banco de Dados — Estrutura do Ranking

Campo	Tipo	Descrição
`user_id`	UUID	Usuário que recebe o feed
`post_id`	UUID	Post candidato
`score_leve`	Float	Score do ranking leve
`score_final`	Float	Score do ranking pesado
`posicao_feed`	Integer	Ordem final no feed
`cache_expira_em`	Timestamp	Data/hora de expiração do snapshot em cache

🔍 Exemplo de Log do Ranking

```
{
  "user_id": "u123",
  "post_id": "p987",
  "score_leve": 0.62,
  "score_final": 0.87,
  "posicao_feed": 3,
  "cache_expira_em": "2025-09-08T18:05:00Z"
}
```

📌 Notas Técnicas Importantes

- Candidate Generation é **altamente paralelizável** com consultas em bancos vetoriais (FAISS/Milvus).
- Ranking Leve roda em **modelo simplificado** (<10ms por consulta).
- Ranking Pesado só roda no **top 500 candidatos**, reduzindo custo em 90%.
- Cache garante feed fluido mesmo em redes móveis.

🔚 Fechamento

“Com o ranking multi-stage, o FriendApp garante escala global sem abrir mão da personalização profunda. A seguir, explicamos como o **Firewall Vibracional Configurável** protege o usuário sem censurar conteúdos sensíveis.”

🔷 CAMADA 4 — FIREWALL VIBRACIONAL CONFIGURÁVEL + CONTENT WARNINGS

📍 **Entrada**

“Proteger o usuário sem censurar o outro. O Firewall Vibracional é um sistema técnico de filtragem configurável que oferece camadas de proteção e escolha consciente.”

🎯 Objetivo da Camada

- Definir como a IA rotula postagens por intensidade emocional.
- Transferir o **poder de filtragem** para o usuário (não para o sistema).
- Implementar **avisos de conteúdo** (content warnings) para postagens intensas.

⚙️ Rotulagem Automática pela IA Aurah Kosmos

Rótulo	Critério Técnico	Exemplo de Conteúdo
Leve	Frequência > 6.0 e intenção positiva	Reflexão motivacional
Intenso	Frequência 4.0–6.0 ou emoção forte	Desabafo emocional
Colapso	Frequência < 4.0 e negatividade persistente	Post de luto, raiva

🔍 **Campos adicionados em banco:** `firewall_label: Enum [leve, intenso, colapso]`

🔄 Configuração pelo Usuário (lado cliente)

Configuração do Usuário	Resultado no Feed
Mostrar tudo	Nenhum filtro, posts aparecem com rótulo simples
Esconder posts de colapso	Posts marcados como “colapso” não aparecem automaticamente
Aviso em conteúdos intensos	Overlay aplicado: “Conteúdo sensível. Toque para visualizar”

🧪 Exemplo de Content Warning

```
{
  "post_id": "p123",
  "firewall_label": "colapso",
  "overlay": true,
  "mensagem": "Este post expressa uma dor profunda. Toque para visualiz
ar."
}
```

📱 **UI no app**: post fica borrado até o usuário tocar para desbloquear.

🧠 Pseudocódigo — Aplicação de Firewall

```
def aplicar_firewall(post, user_config):
    if post.firewall_label == "colapso" and not user_config["mostrar_colaps
o"]:
        return "ocultar"
    elif post.firewall_label == "intenso" and user_config["avisar_intenso"]:
        return "mostrar_overlay"
    else:
        return "mostrar_normal"
```

🔐 Segurança e Ética

- Firewall nunca apaga conteúdo → apenas **rotula e entrega com controle ao usuário**.
- Usuário sempre pode mudar sua configuração nas preferências.
- Logs de posts rotulados ficam disponíveis para auditoria interna.

📦 Banco de Dados — Tabela `firewall_config`

Campo	Tipo	Descrição
`user_id`	UUID	Usuário que configurou o firewall
`mostrar_colapso`	Boolean	True/False

Campo	Tipo	Descrição
`avisar_intenso`	Boolean	Se true → overlay aplicado
`ultima_atualizacao`	Timestamp	Data da última configuração

🔚 Fechamento

“O Firewall Vibracional transforma moderação em escolha. Cada usuário decide quanto deseja se expor, enquanto a IA garante rótulos claros e filtros configuráveis.”

🔷 CAMADA 5 — MODOS DE FEED (PARA VOCÊ, CONEXÕES, EXPLORAR + “POR QUE ESTOU VENDO ISSO?”)

📍 Entrada

“O FriendApp não é uma caixa-preta algorítmica. O usuário tem poder para escolher como deseja navegar: pela curadoria da IA, pelas conexões ou pela exploração de novos campos.”

🎯 Objetivo da Camada

Fornecer **múltiplos modos de visualização do feed**, aumentando transparência, controle do usuário e prevenindo sensação de bolha.

🧭 Modos Disponíveis

Modo	Descrição Técnica	Público-Alvo
Para Você	Curadoria completa da IA Aurah Kosmos (Score de Ressonância + filtros)	Default
Conexões	Postagens **cronológicas** apenas de conexões autênticas	Premium
Explorar	Conteúdos sugeridos fora da bolha, com contraste vibracional seguro	Free/Premium

⚙️ Lógica de Implementação

```
function gerar_feed(usuario, modo):
    if modo == "para_voce":
        return feed_curado(usuario)
    elif modo == "conexoes":
        return buscar_posts_conexoes(usuario, order_by="created_at DESC")
    elif modo == "explorar":
        return buscar_posts_contraste(usuario)
```

🧠 “Por que estou vendo isso?” (Explicabilidade)

- Cada post recebe metadados de justificativa.
- O usuário pode tocar em um botão para entender por que foi mostrado.

📊 Exemplo de Payload

```
{
  "post_id": "p789",
  "explicacao": "Este post ressoa com sua fase de 'Transmutação' e sua int
enção marcada como 'Cura'.",
  "criterios": {
    "similaridade": 0.92,
    "intencao": "cura",
    "afinidade_usuario": true}
}
```

📦 Banco de Dados — Tabela `feed_explicabilidade`

Campo	Tipo	Descrição
`post_id`	UUID	Post exibido
`user_id`	UUID	Usuário visualizando
`criterios`	JSON	Lista de critérios que levaram à exibição
`explicacao`	String	Texto legível para o usuário
`modo_visual`	Enum	para_voce, conexoes, explorar

🎨 UI/UX dos Modos

- **Alternância no topo do feed** (tabs ou botões).
- Feedback visual leve para indicar o modo ativo.
- Posts exibem etiqueta de origem:
  - "🔮 Para Você"
  - "👥 Conexões"
  - "🌍 Explorar"

📌 Notas Técnicas

- No modo **Explorar**, a IA injeta 10–15% de posts de contraste vibracional para evitar bolhas.
- No modo **Conexões**, o sistema não aplica Score de Ressonância → apenas ordem cronológica.
- Logs são armazenados separadamente para cada modo, permitindo auditoria.

🔚 Fechamento

"Com múltiplos modos e explicabilidade, o FriendApp equilibra curadoria inteligente e liberdade de escolha. O próximo passo é detalhar o **algoritmo de cálculo matemático do Score de Ressonância**."

🔷 CAMADA 6 — ALGORITMO MATEMÁTICO DO SCORE DE RESSONÂNCIA

📍 Entrada

"O Score de Ressonância é o núcleo do feed sensorial. Nesta camada, o conceito é transformado em um cálculo matemático executável, que pode ser implementado diretamente pelos devs."

🎯 Objetivo da Camada

Definir de forma **matemática e objetiva** o algoritmo que ordena postagens no feed, explicando variáveis, fórmulas e fluxos de cálculo.

🧬 Variáveis Principais

Variável	Tipo	Descrição
`user_vector`	Array[Float]	Vetor energético do usuário (gerado por perfil + estado atual)
`post_vector`	Array[Float]	Vetor multimodal da postagem (texto, imagem, áudio, vídeo)
`cosine_similarity`	Float	Medida de compatibilidade entre `user_vector` e `post_vector`
`intencao_post`	Float (0–1)	Peso da intenção declarada pelo autor (cura, alegria, inspiração)
`impacto_coletivo`	Float (0–1)	Nível de ressonância da postagem em toda a rede (derivado de logs)
`freshness`	Float (0–1)	Peso de recência (posts mais novos têm maior valor)

🧮 Fórmula Matemática do Score

Score(post,user)=(0.4∗cosinesimilarity(uservector,postvector))+(0.3∗intencaopost)+(0.2∗impactocoletivo)+(0.1∗freshness)Score(post, user) =
  (0.4 * cosine_similarity(user_vector, post_vector)) +
  (0.3 * intencao_post) +
  (0.2 * impacto_coletivo) +
  (0.1 * freshness)

Score(post,user)=(0.4∗cosinesimilarity(uservector,postvector))+(0.3∗intencaopost)+(0.2∗impactocoletivo)+(0.1∗freshness)

🧠 Cálculo da Similaridade (Cosine Similarity)

cosinesimilarity(A,B)=(A⋅B)/(∣∣A∣∣∗∣∣B∣∣)cosine_similarity(A, B) =
  (A · B) / (||A|| * ||B||)

cosinesimilarity(A,B)=(A⋅B)/(∣∣A∣∣∗∣∣B∣∣)

- `A · B` → produto escalar dos vetores.

- `||A||` e `||B||` → norma euclidiana dos vetores.
- Resultado entre `1` e `1`.

📊 Exemplo de Implementação em Código

```
import numpy as np

def calcular_score(user_vector, post_vector, intencao, impacto, freshness):
    cosine_sim = np.dot(user_vector, post_vector) / (
        np.linalg.norm(user_vector) * np.linalg.norm(post_vector)
    )
    score = (0.4 * cosine_sim) + (0.3 * intencao) + (0.2 * impacto) + (0.1 * fre
shness)
    return round(score, 3)
```

🔍 Exemplo de Uso

```
user_vector = [0.1, 0.5, 0.3]
post_vector = [0.2, 0.6, 0.1]
intencao = 0.9
impacto = 0.7
freshness = 0.8

print(calcular_score(user_vector, post_vector, intencao, impacto, freshnes
s))
# Saída: 0.812
```

📦 Banco de Dados — Registro do Score

Campo	Tipo	Descrição
`post_id`	UUID	Identificação do post
`user_id`	UUID	Usuário para o qual o score foi calculado

Campo	Tipo	Descrição
`score_final`	Float	Resultado do cálculo
`cosine_similarity`	Float	Similaridade entre usuário e post
`intencao_post`	Float	Peso da intenção
`impacto_coletivo`	Float	Impacto médio do post na rede
`freshness`	Float	Peso de recência
`gerado_em`	Timestamp	Data/hora do cálculo

📌 Notas Técnicas

- Score é recalculado dinamicamente quando o `user_vector` muda (ex.: alteração de humor/estado).
- Para performance, cálculos de cosine similarity são feitos via banco vetorial (FAISS/Milvus).
- O score é usado apenas no **ranking pesado** (ver Camada 3).

🔚 Fechamento

“O Score de Ressonância traduz energia em matemática. Na próxima camada, detalhamos como o sistema trata **conteúdos sensíveis** com filtros configuráveis pelo usuário e avisos de conteúdo.”

🔷 CAMADA 7 — CONTENT WARNINGS E TRATAMENTO DE CONTEÚDOS SENSÍVEIS

📍 Entrada

“No FriendApp, vulnerabilidade não é censurada — é protegida. Esta camada descreve como a plataforma lida com conteúdos intensos ou de colapso sem apagar sua autenticidade, mas oferecendo controle ao usuário.”

🎯 Objetivo da Camada

Implementar um sistema técnico de **avisos de conteúdo** (content warnings) e overlays que protegem os receptores sem silenciar o emissor.

⚙️ Processo Técnico de Content Warning

1. **Análise da IA Aurah Kosmos**
   - Classifica o post em: **Leve**, **Intenso**, **Colapso** (ver Camada 4).
2. **Aplicação de Overlay (se configurado)**
   - Post marcado como *Intenso* ou *Colapso* → recebe aviso antes da exibição.
3. **Ação do Usuário**
   - Usuário decide se deseja revelar o conteúdo.

🧠 Pseudocódigo de Content Warning

```
def aplicar_content_warning(post, user_config):
    if post.firewall_label == "colapso":
        return {
            "overlay": True,
            "mensagem": "Este post expressa dor profunda. Toque para revela
r."
        }
    elif post.firewall_label == "intenso" and user_config["avisar_intenso"]:
        return {
            "overlay": True,
            "mensagem": "Conteúdo emocional intenso. Deseja visualizar?"
        }
    else:
        return {"overlay": False}
```

📊 Exemplo de Payload com Content Warning

```
{
  "post_id": "p555",
  "firewall_label": "colapso",
  "overlay": true,
  "mensagem": "Este post contém uma expressão de dor profunda. Toque para visualizar."
}
```

📦 Banco de Dados — Tabela `content_warnings`

Campo	Tipo	Descrição
`post_id`	UUID	Post analisado
`firewall_label`	Enum	leve, intenso, colapso
`overlay_ativo`	Boolean	True se aviso foi aplicado
`mensagem_overlay`	String	Texto do aviso mostrado ao usuário
`visualizado`	Boolean	True se usuário decidiu abrir o conteúdo
`data_aplicacao`	Timestamp	Momento em que o aviso foi inserido

🖼️ UX de Aviso no App

- Post aparece **borrado** ou com uma **camada translúcida**.
- Botão: “👁️ Ver conteúdo”
- Se o usuário clicar:
  - Post é exibido normalmente
  - Log registra que o conteúdo foi revelado

🔐 Notas Técnicas e Éticas

- **Não existe censura:** todo conteúdo é armazenado e pode ser visto.
- O usuário tem o **direito de escolha**: revelar ou não.
- Logs registram se o aviso foi ignorado → alimentam a IA com dados sobre tolerância dos usuários.

- Posts com alta frequência de content warnings podem ser sugeridos para **moderação manual**.

🔚 Fechamento

“O sistema de Content Warnings garante equilíbrio: proteger sem censurar, acolher sem expor à força. Na próxima camada, entramos nos **Logs Sensoriais e Observabilidade** para monitorar impacto e histórico de cada postagem.”

🔷 CAMADA 8 — LOGS SENSORIAIS E OBSERVABILIDADE

📍 Entrada

“Não basta publicar — é preciso medir. O sistema de logs sensoriais garante que cada postagem seja registrada, acompanhada e analisada para retroalimentar a IA e manter o feed transparente, auditável e evolutivo.”

🎯 Objetivo da Camada

Implementar um **sistema robusto de logs e observabilidade** que capture dados de impacto vibracional de cada post, permitindo:

- Monitoramento em tempo real.
- Auditoria e transparência para devs e moderadores.
- Treinamento contínuo da IA Aurah Kosmos.

📡 Tipos de Logs Registrados

Tipo de Log	O que Registra
Frequência Base	Valor inicial do post (antes de interações)
Interações	Curtidas vibracionais, comentários sensoriais, denúncias
Expansão Vibracional	Quantidade de conexões impactadas positivamente
Colapso Vibracional	Casos de impacto negativo detectado pela IA ou usuários
Uso de Content Warning	Se o post foi exibido com overlay e quantos revelaram

---

🧠 Pseudocódigo de Registro de Logs

```
def registrar_log(post_id, user_id, evento, valor):
    log = {
        "post_id": post_id,
        "user_id": user_id,
        "evento": evento,
        "valor": valor,
        "timestamp": agora()
    }
    banco.logs.insert(log)
```

📊 Estrutura no Banco — `logs_sensorial`

Campo	Tipo	Descrição
`log_id`	UUID	Identificação única do log
`post_id`	UUID	Postagem associada
`user_id`	UUID	Usuário que gerou a interação
`evento`	String	Tipo de evento (curtida, denúncia, warning)
`valor`	Float/JSON	Dados do evento
`timestamp`	Timestamp	Data/hora da ocorrência

🗺️ Dashboard de Observabilidade (para devs e admins)

- **Mapa de calor vibracional**: impacto por horário/região.
- **Tendências coletivas**: posts que mais elevaram ou colapsaram o campo.
- **Logs de auditoria**: posts ocultados, denunciados ou autoapagados.
- **KPIs**:
  - Tempo médio de interação.
  - % de posts com expansão vs colapso.
  - Frequência média da rede.

📦 Exemplo de Log Gerado

```
{
  "log_id": "l001",
  "post_id": "p555",
  "user_id": "u789",
  "evento": "curtida_vibracional",
  "valor": 1,
  "timestamp": "2025-09-08T12:45:00Z"
}
```

📌 Notas Técnicas

- Logs são armazenados em **Postgres (estruturado)** + **Elasticsearch (consultas rápidas)**.
- Dados críticos (impacto vibracional coletivo) são replicados em tempo real para o painel interno.
- Logs alimentam relatórios semanais para moderação e aprendizado da IA.

🔚 Fechamento

“Com observabilidade completa, o FriendApp transforma interações em inteligência. O próximo passo é detalhar o **Painel de Insights Vibracionais (exclusivo Premium)**, que mostra ao usuário como seu conteúdo impacta a rede.”

🔷 CAMADA 9 — PAINEL DE INSIGHTS VIBRACIONAIS (EXCLUSIVO PREMIUM)

📍 Entrada

“O feed mostra, mas o painel explica. Esta camada detalha como o FriendApp oferece aos usuários Premium uma visão clara do impacto energético de suas postagens.”

🎯 Objetivo da Camada

Entregar ao usuário Premium **feedback mensurável** sobre como suas postagens afetam:

- Sua própria frequência.
- As conexões autênticas.
- O campo coletivo da rede.

📊 Métricas do Painel

Métrica	Descrição Técnica
Índice de Ressonância	Score médio de compatibilidade entre o post e os usuários
Frequência Média dos Receptores	Valor em Hz calculado pelas interações no post
Pico de Expansão	Momento de maior impacto (timestamp)
Eco Coletivo	Número de interações vibracionais positivas geradas em cadeia
Colapsos Detectados	Casos em que usuários reagiram negativamente (com logs para auditoria)

🧠 Pipeline Técnico

1. Post é publicado → IA gera `post_vector`.
2. Interações são registradas em `logs_sensorial`.
3. A cada 6h, Aurah Kosmos processa os logs → gera métricas agregadas.
4. Dados são exibidos no painel do usuário Premium em tempo quase real.

🔢 Pseudocódigo Simplificado

```
def calcular_insights(post_id):
    interacoes = buscar_logs(post_id)
    ressonancia = media([i.similaridade for i in interacoes])
    freq_media = media([i.frequencia_user for i in interacoes])
    pico = max(interacoes, key=lambda x: x.impacto)
    return {
```

```
    "ressonancia": ressonancia,
    "frequencia_media": freq_media,
    "pico_expansao": pico.timestamp
}
```

📦 Banco de Dados — Tabela `painel_insights`

Campo	Tipo	Descrição
`post_id`	UUID	Post analisado
`user_id`	UUID	Autor do post
`indice_ressonancia`	Float	Compatibilidade média
`frequencia_media`	Float	Média das frequências dos receptores
`eco_coletivo`	Integer	Número de interações positivas geradas
`colapsos_detectados`	Integer	Número de reações negativas
`pico_expansao`	Timestamp	Horário de maior impacto

🎨 UI/UX do Painel

- **Gráfico de linha**: mostra evolução da frequência do post ao longo do tempo.
- **Mapa de calor**: impacto geográfico (regiões onde ressoou mais).
- **Resumo textual da IA**:
  - Exemplo: *"Seu post de ontem elevou 73 conexões e foi mais sentido no Sudeste."*

📌 Notas Técnicas

- Apenas **usuários Premium** têm acesso ao painel.
- Dados são atualizados em batch (a cada 6h) para otimizar performance.
- O painel **não mostra nomes de quem foi impactado**, apenas dados agregados para preservar privacidade.

- Logs de colapsos são tratados separadamente e enviados também para a moderação.

🔚 Fechamento

“O Painel de Insights Vibracionais dá ao usuário a visão do impacto invisível. O próximo passo é conectar isso ao **sistema de feedback sensorial em tempo real**.”

🔷 CAMADA 10 — SISTEMA DE FEEDBACK SENSORIAL EM TEMPO REAL

📍 Entrada

“No FriendApp, interagir não é apertar um botão — é gerar um pulso. O sistema de feedback sensorial substitui curtidas genéricas por respostas energéticas em tempo real.”

🎯 Objetivo da Camada

Implementar um mecanismo de **respostas energéticas em tempo real** que traduza cada interação em sinais visuais, auditivos e vibracionais, registrando impacto no emissor e na rede.

⚡ Tipos de Feedback

Tipo de Interação	Efeito no Feed
Curtida Vibracional	Pulso leve (cor/aura) aplicado ao post
Ressonância	Post ganha brilho extra se vários usuários reagirem em sincronia
Eco Sensorial	Post reverbera e pode aparecer em “Explorar” com selo de expansão
Feedback Negativo	Colapso → IA reduz alcance do post e registra no log

---

📊 Exemplo de Payload de Feedback

```
{
  "user_id": "u123",
  "post_id": "p456",
  "tipo_feedback": "ressonancia",
  "intensidade": 0.87,
  "timestamp": "2025-09-08T14:33:00Z"
}
```

🧠 Pipeline Técnico de Feedback

1. Usuário interage → evento registrado em tempo real.
2. IA calcula intensidade (`intensidade = similaridade + emoção detectada`).
3. Post atualizado no feed do autor e conexões em **menos de 300ms**.
4. Log registrado em `logs_sensorial` com impacto acumulado.

🔢 Pseudocódigo Simplificado

```
def processar_feedback(user_id, post_id, tipo):
    intensidade = calcular_intensidade(user_id, post_id)
    atualizar_post_visual(post_id, tipo, intensidade)
    registrar_log(post_id, user_id, tipo, intensidade)
```

📦 Banco de Dados — `feedback_sensorial`

Campo	Tipo	Descrição
`feedback_id`	UUID	Identificação única do feedback
`user_id`	UUID	Usuário que reagiu
`post_id`	UUID	Post reagido
`tipo`	Enum	curtida, ressonancia, eco, negativo
`intensidade`	Float	Valor entre 0.0–1.0
`timestamp`	Timestamp	Momento da interação

🎨 UI/UX de Feedback

- Post pulsa com cores dinâmicas ao receber interações.
- Se vários usuários interagem em sincronia → feed exibe efeito coletivo (onda de luz).
- Feedback negativo não é exibido publicamente → apenas reduz alcance do post.

📌 Notas Técnicas

- Feedbacks são processados via **WebSocket** para garantir latência baixa (<300ms).
- Logs alimentam o **Painel de Insights** (ver Camada 9).
- Feedback coletivo pode ser usado para impulsionar posts no modo **Explorar**.

🔚 Fechamento

“O sistema de feedback transforma interação em energia visível. O próximo passo é detalhar a **Diferenciação Free vs Premium** aplicada a este mecanismo, ampliando recursos para quem assina.”

🔷 CAMADA 11 — DIFERENCIAÇÃO FREE VS PREMIUM NO FEEDBACK E FEED SENSORIAL

📍 Entrada

“O FriendApp é inclusivo, mas o acesso completo ao Feed Sensorial e aos mecanismos de feedback só se revela no modo Premium. Esta camada define tecnicamente os limites e liberdades de cada perfil.”

🎯 Objetivo da Camada

Definir, de forma objetiva, quais funcionalidades do Feed e Feedback estão disponíveis para usuários Free e quais são exclusivas dos Premium.

⚙️ Comparação de Recursos

Recurso	Free	Premium
Ver postagens temporárias (Momentos)	✅ Sim	✅ Sim
Ver postagens permanentes	❌ Não	✅ Sim
Publicar postagens	❌ Apenas Momentos	✅ Momentos e Permanentes
Feedback Vibracional (curtidas)	❌ Somente visualizar	✅ Interagir e enviar
Feedback Ressonância Coletiva	❌ Não disponível	✅ Disponível em tempo real
Painel de Insights (Camada 9)	❌ Não disponível	✅ Exclusivo
Filtros de Feed (Intenção/Frequência)	❌ Não	✅ Personalizável
Firewall Configurável (Camada 4)	Básico (default)	Avançado (configurável)
Explicabilidade (“Por que vejo isso?”)	❌ Não	✅ Sim

🧠 Pseudocódigo — Controle de Acesso

```
def verificar_permissao(usuario, recurso):
    if usuario.tipo == "premium":
        return True
    elif usuario.tipo == "free":
        if recurso in ["momentos", "visualizar_basico"]:
            return True
        else:
            return False
```

📦 Banco de Dados — Tabela `user_plan`

Campo	Tipo	Descrição
`user_id`	UUID	Identificação do usuário
`tipo`	Enum	free, premium
`data_inicio_premium`	Timestamp	Quando começou plano Premium
`data_expiracao`	Timestamp	Fim do período Premium
`recursos_habilitados`	JSON	Lista de recursos ativos (dinâmico)

📊 Exemplo de Fluxo de Feedback (Free x Premium)

- Usuário **Free** toca em post → só vê contagem de ressonâncias.
- Usuário **Premium** toca em post → envia sua própria ressonância, que pulsa em tempo real no feed do autor.

📌 Notas Técnicas

- Todo controle é feito em **camada de API** (backend), impedindo bypass.
- UI do app já oculta botões Premium para usuários Free, mas back-end valida por redundância.
- Usuários Free recebem prompts sutis para upgrade ao tentar acessar recursos bloqueados.

🔚 Fechamento

“A diferenciação Free vs Premium garante sustentabilidade sem excluir: todos podem ver, mas apenas quem expande tem acesso total. O próximo passo é detalhar a **Segurança, Moderação e Autoapagamento Automático** do feed.”

🔷 CAMADA 12 — SISTEMA DE SEGURANÇA E AUTOAPAGAMENTO (DENÚNCIAS, BLOQUEIOS, COLAPSOS)

📍 Entrada

“No FriendApp, segurança não é censura, é proteção. Esta camada estabelece os mecanismos automáticos de denúncia, moderação e autoapagamento para manter o feed saudável e coerente.”

🎯 Objetivo da Camada

Definir regras e processos de **moderação inteligente** que eliminam conteúdos prejudiciais ou incoerentes, mantendo a liberdade do usuário sem comprometer o coletivo.

⚙️ Componentes do Sistema de Segurança

Componente Técnico	Função
IA Aurah Kosmos	Detecta conteúdos dissonantes antes da exibição
Denúncias de Usuários	Permite à comunidade sinalizar posts
Autoapagamento Vibracional	Remove posts que atingirem thresholds negativos
Painel Interno	Revisão manual em casos críticos
Shadowban Temporário	Reduz alcance de usuários reincidentes sem excluir sua conta

📡 Critérios de Autoapagamento

- **Score vibracional < 3.0** após 50 interações.
- **10+ denúncias** únicas em menos de 1h.
- Conteúdo classificado pela IA como **violência explícita ou discurso de ódio**.

🧠 Fluxo de Segurança

```
function avaliar_post(post):
    if post.score_final < 3.0:
        autoapagar(post)
    elif contar_denuncias(post) > 10:
        ocultar(post)
        enviar_para_revisao(post)
```

📦 Banco de Dados — Tabela `moderacao_posts`

Campo	Tipo	Descrição
`post_id`	UUID	Post moderado
`user_id`	UUID	Autor
`status`	Enum	ativo, ocultado, apagado
`motivo`	String	Razão da ação (score baixo, denúncias, IA detectou)
`denuncias_total`	Integer	Número de denúncias
`data_acao`	Timestamp	Quando a ação foi tomada

📊 Exemplo de Log de Ação

```
{
  "post_id": "p321",
  "user_id": "u999",
  "status": "autoapagado",
  "motivo": "score_baixo",
  "denuncias_total": 12,
  "data_acao": "2025-09-08T20:15:00Z"
}
```

🔐 Estratégias de Proteção Extra

- Posts autoapagados permanecem visíveis apenas para o autor.
- Usuários reincidentes (>3 vezes em 7 dias) recebem alerta e podem entrar em **modo de pausa sensorial** (feed mais leve).
- Moderadores têm acesso a logs completos para revisar casos de falso positivo.

📌 Notas Técnicas

- Denúncias são auditáveis, cada denúncia é validada para evitar abuso.
- Autoapagamentos são notificados ao usuário com explicação clara:

> “Sua postagem foi removida por score vibracional baixo. Deseja revisá-la e repostar?”

🔚 Fechamento

“O sistema de segurança garante a integridade do feed sem eliminar autenticidade. O próximo passo é detalhar o **Mapa de Impacto Energético**, que mostra como os posts se espalham pelo ecossistema.”

🔷 CAMADA 13 — MAPA DE IMPACTO ENERGÉTICO

📍 Entrada

“O impacto de uma postagem não se limita ao feed de um usuário. Cada interação gera ondas que se espalham pela rede. Esta camada define como medir, mapear e visualizar esse impacto.”

🎯 Objetivo da Camada

Construir um sistema que rastreia **a propagação vibracional** de um post, permitindo identificar:

- Onde ele repercutiu (geografia e conexões).
- Se expandiu ou colapsou o campo coletivo.
- Qual foi a intensidade e duração da sua reverberação.

⚙️ Estrutura Técnica do Mapa de Impacto

Elemento	Descrição Técnica
Impacto Local	Conexões diretas impactadas (primeiro círculo)
Impacto Regional	Repercussão por geolocalização (cidade/região)
Impacto Global	Expansão em outras regiões do ecossistema FriendApp
Ciclo Temporal	Gráfico de ressonância no tempo (início, pico, decaimento)
Polaridade	% de interações positivas vs negativas

📊 Exemplo de Payload — Impacto de Postagem

```
{
  "post_id": "p789",
  "impacto_local": 42,
  "impacto_regional": {
    "SP": 15,
    "RJ": 7,
    "MG": 5
  },
  "impacto_global": 87,
  "polaridade": {
    "positiva": 0.82,
    "negativa": 0.18
  },
  "ciclo_temporal": {
    "inicio": "2025-09-08T10:00:00Z",
    "pico": "2025-09-08T12:30:00Z",
    "fim": "2025-09-09T10:00:00Z"
  }
}
```

🧠 Pseudocódigo de Cálculo

```
def calcular_mapa_impacto(post_id):
    interacoes = buscar_logs(post_id)
    impacto_local = contar_conexoes(interacoes)
    impacto_regional = agrupar_por_regiao(interacoes)
    impacto_global = len(interacoes)
    polaridade = calcular_polaridade(interacoes)
    ciclo_temporal = calcular_timeline(interacoes)
    return {
        "impacto_local": impacto_local,
        "impacto_regional": impacto_regional,
        "impacto_global": impacto_global,
        "polaridade": polaridade,
```

```
    "ciclo_temporal": ciclo_temporal
}
```

📦 Banco de Dados — Tabela `mapa_impacto`

Campo	Tipo	Descrição
`post_id`	UUID	Post analisado
`impacto_local`	Integer	Número de conexões diretas impactadas
`impacto_regional`	JSON	Dados agregados por região
`impacto_global`	Integer	Total de usuários impactados
`polaridade`	JSON	% positivo vs negativo
`ciclo_temporal`	JSON	Dados de início, pico e fim da repercussão

🎨 Visualização no App (para Premium)

- **Mapa de calor**: mostra onde o post reverberou mais.
- **Gráfico de linha**: curva de intensidade no tempo.
- **Resumo da IA**:

> "Sua postagem alcançou 87 pessoas, elevou 82% delas e gerou expansão maior no Sudeste."

📌 Notas Técnicas

- Dados são atualizados em tempo quase real (batch de 15 minutos).
- Usuários Free não têm acesso ao mapa — apenas Premium.
- Mapas são agregados e **não expõem indivíduos**, apenas tendências.

🔚 Fechamento

"O Mapa de Impacto transforma métricas em significado, mostrando como a energia se espalha no ecossistema. O próximo passo é conectar este impacto ao **algoritmo de priorização do feed**."

🔷 CAMADA 14 — ALGORITMO DE PRIORIZAÇÃO NO FEED

📍 **Entrada**

“O FriendApp não entrega popularidade, entrega compatibilidade. Esta camada define como o algoritmo organiza as postagens no feed após calcular o Score de Ressonância.”

🎯 Objetivo da Camada

Explicar tecnicamente como o feed aplica regras de **ordenação final** para cada usuário, equilibrando:

- Compatibilidade pessoal.
- Impacto coletivo.
- Diversidade de conteúdos (para evitar bolha).
- Recência.

⚙️ Variáveis Consideradas

Variável	Peso (%)	Observação
Score de Ressonância	40	Resultado do cálculo da Camada 6
Impacto Coletivo	20	Baseado nos logs e mapa de impacto
Recência (Freshness)	20	Evita feed congelado, garante fluxo constante
Diversidade de Conteúdo	10	Garante variedade (texto, imagem, vídeo, áudio)
Randomização Segura	10	Injeta posts fora da bolha para ampliar percepção do usuário

---

🧮 Fórmula de Priorização

Prioridade(post)=(0.4∗ScoreRessonancia)+(0.2∗ImpactoColetivo)+(0.2∗Freshness)+(0.1∗Diversidade)+(0.1∗Randomizacao)Prioridade(post) =
  (0.4 * ScoreRessonancia) +
  (0.2 * ImpactoColetivo) +
  (0.2 * Freshness) +

(0.1 * Diversidade) +
(0.1 * Randomizacao)

$$\text{Prioridade(post)} = (0.4 * \text{ScoreRessonancia}) + (0.2 * \text{ImpactoColetivo}) + (0.2 * \text{Freshness}) + (0.1 * \text{Diversidade}) + (0.1 * \text{Randomizacao})$$

🧠 Pseudocódigo de Ordenação

```
def priorizar_posts(posts):
    for post in posts:
        post.prioridade = (
            0.4 * post.score_ressonancia +
            0.2 * post.impacto_coletivo +
            0.2 * post.freshness +
            0.1 * calcular_diversidade(post) +
            0.1 * randomizacao_segura()
        )
    return sorted(posts, key=lambda x: x.prioridade, reverse=True)
```

📦 Banco de Dados — Campo `prioridade_final`

Campo	Tipo	Descrição
`post_id`	UUID	Postagem ordenada
`user_id`	UUID	Usuário que recebe o feed
`prioridade_final`	Float	Valor final após cálculo
`modo_visualizacao`	Enum	para_voce, conexoes, explorar
`gerado_em`	Timestamp	Data/hora da ordenação

📊 Exemplo de Saída do Algoritmo

```
[
  {
    "post_id": "p001",
    "prioridade_final": 0.92,
    "motivo": "Alta compatibilidade + impacto coletivo"
```

```
  },
  {
    "post_id": "p002",
    "prioridade_final": 0.78,
    "motivo": "Recente + diversidade de conteúdo"
  }
]
```

📌 Notas Técnicas

- Posts de baixa prioridade ainda podem aparecer via randomização controlada, garantindo diversidade.
- O algoritmo evita que apenas **posts virais** dominem o feed.
- Cada feed é recalculado **a cada abertura** ou após mudanças significativas no estado vibracional do usuário.

🔚 Fechamento

“O algoritmo de priorização é o maestro do feed, equilibrando compatibilidade, impacto coletivo e diversidade. O próximo passo é detalhar como funciona o **Sistema de Caching e Performance**, que garante fluidez mesmo em escala global.”

🔷 CAMADA 15 — SISTEMA DE CACHING E PERFORMANCE

📍 Entrada

“O feed sensorial precisa ser rápido, fluido e global. Esta camada define como caching, pré-renderização e baixa latência são garantidos, mesmo com milhões de usuários simultâneos.”

🎯 Objetivo da Camada

Projetar um sistema de caching que equilibra **personalização em tempo real** com **escala global**, mantendo latência abaixo de 300ms por interação.

⚡ Estratégia de Caching

Componente	Função Técnica
Redis Cache	Armazena snapshots de feed personalizados com TTL curto
CDN Global (Cloudflare)	Distribui conteúdo estático (imagens, vídeos) de forma local
Pré-renderização	Pré-carrega os próximos 10 posts no scroll
Invalidadores Seletivos	Atualizam apenas posts novos/relevantes, sem recarregar tudo

📊 Configuração do TTL

- **Feed personalizado**: 3–5 minutos (ajustável por usuário ativo).
- **Posts recentes**: atualizados em tempo real via WebSocket.
- **Posts antigos**: servidos diretamente do cache até expirar.

🧠 Pseudocódigo de Caching

```
def gerar_feed_cache(user_id):
    if redis.exists(f"feed:{user_id}"):
        return redis.get(f"feed:{user_id}")
    else:
        feed = gerar_feed_personalizado(user_id)
        redis.set(f"feed:{user_id}", feed, ex=300) # TTL = 5 min
        return feed
```

📦 Banco de Dados — Tabela `feed_cache`

Campo	Tipo	Descrição
`user_id`	UUID	Usuário dono do feed

Campo	Tipo	Descrição
`feed_snapshot`	JSON	Lista de posts ranqueados
`ttl`	Integer	Tempo de vida em segundos
`cache_criado`	Timestamp	Quando o snapshot foi gerado

📈 Monitoramento de Performance

Indicador	Meta Técnica
Latência média	< 250ms
Cache hit rate	> 90%
Erros de carregamento	< 0.5%
Tempo de scroll infinito	< 150ms por novo bloco

🔍 Exemplo de Cache de Feed

```
{
  "user_id": "u123",
  "feed_snapshot": [
    {"post_id": "p001", "score": 0.92},
    {"post_id": "p002", "score": 0.85}
  ],
  "ttl": 300,
  "cache_criado": "2025-09-08T15:45:00Z"
}
```

📌 Notas Técnicas

- Cache é **invalidação seletiva**: só posts novos ou impactantes substituem o snapshot.
- WebSocket garante que posts críticos aparecem instantaneamente.
- Redis clusterizado suporta milhões de requisições simultâneas.

🔚 Fechamento

“O caching garante fluidez e escalabilidade. O próximo passo é detalhar o **Sistema de Logs de Performance e Failover**, para manter o feed resiliente mesmo em falhas.”

🔷 CAMADA 16 — LOGS DE PERFORMANCE E FAILOVER

📍 **Entrada**

“Não basta ser rápido — é preciso ser resiliente. Esta camada descreve como o FriendApp registra métricas de performance e aciona mecanismos de failover para garantir que o feed nunca pare.”

🎯 Objetivo da Camada

- Registrar métricas de latência, erros e uso de cache.
- Detectar gargalos em tempo real.
- Acionar mecanismos de failover quando um serviço falhar.

⚡ Métricas Monitoradas

Indicador	Meta Técnica
Latência média do feed	< 250ms
Taxa de erro de API	< 0.5%
Cache hit rate	> 90%
Uptime do feed service	99.98%
Tempo de render no app	< 1.2s

📊 Exemplo de Log de Performance

```
{
  "log_id": "pf001",
  "user_id": "u123",
  "latencia": 180,
```

```
  "cache_hit": true,
  "erro": false,
  "servidor": "feed-node-07",
  "timestamp": "2025-09-08T16:12:00Z"
}
```

🧠 Pseudocódigo de Monitoramento

```
def monitorar_requisicao(user_id, latencia, erro, cache_hit):
    log = {
        "user_id": user_id,
        "latencia": latencia,
        "erro": erro,
        "cache_hit": cache_hit,
        "timestamp": agora()
    }
    banco.logs_performance.insert(log)

    if latencia > 1000 or erro:
        acionar_failover()
```

🔄 Mecanismos de Failover

Falha Detectada	Ação Automática
API principal fora do ar	Redireciona para réplica secundária
Tempo de resposta > 3s	Ativa modo “low fidelity” (feed sem efeitos visuais)
Falha na IA de Curadoria	Exibe posts com ranqueamento básico do cache
Perda de logs	Armazena em fila Kafka até serviço voltar

📦 Banco de Dados — Tabela `logs_performance`

Campo	Tipo	Descrição
`log_id`	UUID	Identificação única do log

Campo	Tipo	Descrição
`user_id`	UUID	Usuário impactado
`latencia`	Integer	Tempo de resposta em ms
`erro`	Boolean	Se houve erro
`cache_hit`	Boolean	Se resposta veio do cache
`servidor`	String	Qual nó atendeu a requisição
`timestamp`	Timestamp	Data/hora do registro

📌 Notas Técnicas

- Logs de performance ficam disponíveis em **Datadog + Grafana** para dashboards.
- Alertas automáticos disparam no Slack da equipe se **erro >1% em 5 minutos**.
- Failover é testado a cada sprint com simulações de falha.

🔚 Fechamento

“Com logs de performance e failover ativo, o FriendApp garante que o feed continue funcionando mesmo em falhas. O próximo passo é conectar tudo ao **Sistema de Integrações e Dependências Cíclicas do Ecossistema**.”

🔷 CAMADA 17 — INTEGRAÇÕES E DEPENDÊNCIAS DO ECOSSISTEMA

📍 Entrada

“O Feed Sensorial não é isolado — ele é o coração que bombeia dados para todo o ecossistema FriendApp. Esta camada mapeia como o feed consome e entrega informações para outros módulos.”

🎯 Objetivo da Camada

Documentar as integrações técnicas do feed com os principais sistemas do FriendApp, garantindo **clareza para desenvolvedores** e evitando dúvidas sobre dependências.

⚙️ Dependências de Entrada (o que o feed consome)

Sistema Origem	Dados Fornecidos	Tipo de Integração
Cadastro Consciente	Vetor inicial do usuário (user_vector)	API + Banco
Teste de Personalidade Energética	Perfil energético e arquétipos	API + Logs
Mapa de Frequência	Frequência local/regional	Consulta periódica
Aurah Kosmos (IA)	Análise de intenção, sentimento e curadoria	Webhook + API
Sistema Premium	Flags de recursos habilitados	API + Cache

📤 Saídas do Feed (para outros sistemas)

Destino	Dados Enviados	Tipo de Entrega
Mapa de Frequência	Impactos locais de postagens	Logs + API
Sistema de Conexões Autênticas	Afinidade entre usuários baseada em posts	Webhook + Banco
Chat Vibracional	Contexto inicial de conversas (post origem)	API + Trigger
Eventos & Locais Parceiros	Posts relacionados a eventos e check-ins	API + Integração
Jogo da Transmutação	Posts usados como gatilhos de missões	API + Logs
Painel de Insights Premium	Dados de impacto dos posts	Consulta direta

🧠 Fluxo de Integração Simplificado

```
flowchart TD
  A[Usuário cria post] --> B[Aurah Kosmos analisa]
  B --> C[Feed Sensorial exibe]
  C --> D[Mapa de Frequência recebe impacto]
  C --> E[Chat Vibracional pode iniciar conversa]
```

```
C --> F[Conexões Autênticas atualizam afinidade]
C --> G[Jogo da Transmutação recebe gatilhos]
```

📦 Banco de Dados — Relacionamentos

- `user_vector` sincronizado com **Cadastro + Teste de Personalidade**.
- `impacto_regional` gravado em **Mapa de Frequência**.
- `afinidade_posts` alimenta **Conexões Autênticas**.
- `post_origem` usado como referência em **Chats**.

📌 Notas Técnicas

- Integrações são **assíncronas** via **Kafka/Webhooks** para não impactar latência do feed.
- Logs de saída têm **retentiva de 90 dias** para auditoria.
- Dependências críticas (Aurah Kosmos e Cadastro Consciente) têm **failover local** em cache para evitar indisponibilidade.

🔚 Fechamento

“O Feed Sensorial é o hub central de dados do FriendApp. Ele consome perfis, gera impacto, cria conexões e inicia conversas. O próximo passo é detalhar como a IA Aurah Kosmos atua como motor inteligente da curadoria.”

🔷 CAMADA 18 — PAPEL DA IA AURAH KOSMOS NO FEED SENSORIAL

📍 Entrada

“Aurah Kosmos é mais do que um filtro: é o motor inteligente que lê, classifica, rotula e ranqueia cada postagem. Esta camada explica como a IA atua tecnicamente dentro do pipeline do feed.”

🎯 Objetivo da Camada

Descrever em detalhes o **pipeline de curadoria da Aurah Kosmos**, desde a análise multimodal de conteúdo até a decisão de exibição, priorização ou bloqueio.

⚡ Funções da IA no Feed

Função	Descrição Técnica
Vetorização Multimodal	Converte texto, imagem, áudio e vídeo em vetores comparáveis
Score de Ressonância	Calcula compatibilidade entre usuário e post
Rotulagem Vibracional	Marca posts como leve, intenso ou colapso
Moderação Automática	Bloqueia conteúdos ofensivos, propõe content warnings
Explicabilidade	Gera mensagens “Por que estou vendo isso?”
Feedback Inteligente	Sugere ajustes de intenção ao usuário antes de publicar

🧠 Pipeline Técnico

```
flowchart TD
  A[Postagem Criada] --> B[Aurah Kosmos analisa multimodal]
  B --> C[Gerar post_vector]
  C --> D[Comparar com user_vector]
  D --> E[Calcular Score de Ressonância]
  E --> F{Compatibilidade}
  F -->|Alta| G[Priorizar no feed]
  F -->|Média| H[Exibir normal]
  F -->|Baixa| I[Ocultar ou sinalizar para revisão]
```

🔬 Pseudocódigo Simplificado

```
def processar_postagem(post, user_vector):
    post_vector = vetorizar_post(post)
    score = calcular_score(user_vector, post_vector,
                           post.intencao, post.impacto, post.freshness)
```

```
    if score >= 0.8:
        status = "prioritario"
    elif score >= 0.5:
        status = "normal"
    else:
        status = "baixo"

    rotulo = aurah.rotular(post)
    return {"score": score, "status": status, "rotulo": rotulo}
```

📦 Banco de Dados — Tabela `curadoria_aurah`

Campo	Tipo	Descrição
`post_id`	UUID	Post analisado
`post_vector`	Array[Float]	Representação numérica multimodal
`score_ressonancia`	Float	Valor calculado pela IA
`rotulo`	Enum	leve, intenso, colapso
`status_exibicao`	Enum	prioritario, normal, baixo
`explicacao`	String	Motivo da decisão, usado no “Por que vejo isso?”

📊 Exemplo de Saída

```
{
  "post_id": "p456",
  "score_ressonancia": 0.83,
  "rotulo": "leve",
  "status_exibicao": "prioritario",
  "explicacao": "Post ressoa com sua intenção de 'cura' e alta compatibilida
de"
}
```

📌 Notas Técnicas

- Aurah Kosmos usa embeddings pré-treinados + modelos refinados internamente.
- Toda decisão gera log auditável.
- A IA nunca deleta conteúdo sozinha — apenas **rotula e recomenda ações**.

🔚 **Fechamento**

"Aurah Kosmos transforma dados em decisões inteligentes, mantendo o feed pessoal, escalável e transparente. A próxima camada descreve como o feed garante **diversidade e prevenção de bolhas vibracionais**."

🔷 CAMADA 19 — DIVERSIDADE DE CONTEÚDO E ANTI-BOLHA VIBRACIONAL

📍 **Entrada**

"Um feed que só mostra o que você já sente cria conforto, mas também aprisiona. O FriendApp quebra essa lógica, introduzindo diversidade intencional e controles anti-bolha."

🎯 Objetivo da Camada

- Prevenir que o feed se torne uma **câmara de eco vibracional**.
- Garantir que o usuário veja conteúdos de diferentes intenções, formatos e frequências.
- Expor de forma segura a conteúdos contrastantes, que também podem gerar crescimento.

⚡ Estratégias Anti-Bolha

Estratégia	Descrição Técnica
Diversidade de Mídia	Garante que entre posts de texto, imagem, vídeo e áudio em cada sessão.
Injeção de Contraste	10–15% do feed vem de conteúdos fora da compatibilidade direta, mas seguros.
Modo Explorar (Camada 5)	Amplia exposição a novas conexões e frequências regionais.

Estratégia	Descrição Técnica
Controle de Polaridade	Balanceia posts de “cura” e “introspecção” com “alegria” e “expansão”.

🧠 Algoritmo de Diversidade

```
def aplicar_diversidade(feed):
    feed_diverso = []
    for i, post in enumerate(feed):
        feed_diverso.append(post)
        if i % 5 == 0:  # a cada 5 posts
            feed_diverso.append(selecionar_post_contraste())
    return feed_diverso
```

- `selecionar_post_contraste()` busca posts com **score médio** (0.4–0.6), que não seriam exibidos normalmente, mas ampliam horizontes.

📊 Exemplo de Feed com Diversidade

```
[
  {"post_id": "p001", "score": 0.92, "tipo": "texto"},
  {"post_id": "p002", "score": 0.87, "tipo": "imagem"},
  {"post_id": "p003", "score": 0.45, "tipo": "contraste_injetado"}
]
```

📦 Banco de Dados — Registro de Diversidade

Campo	Tipo	Descrição
`user_id`	UUID	Usuário que recebeu feed
`post_id`	UUID	Post exibido como diversidade
`tipo_diversidade`	Enum	contraste, mídia, polaridade
`injetado_por`	String	IA Aurah, Modo Explorar
`timestamp`	Datetime	Quando foi injetado

📌 Notas Técnicas

- Diversidade é aplicada no **ranking final** (Camada 14).
- Posts de contraste têm **explicabilidade clara**:

  > “Este post foi mostrado para expandir sua percepção, mesmo não sendo altamente compatível.”

- Logs de diversidade são usados para avaliar se a exposição está ajudando ou causando rejeição.

🔚 Fechamento

“Com diversidade e anti-bolha, o feed se mantém autêntico, mas também desafiador, ampliando horizontes. O próximo passo é detalhar o **Sistema de Transparência e Explicabilidade da IA**.”

🔷 CAMADA 20 — TRANSPARÊNCIA E EXPLICABILIDADE DA IA

📍 Entrada

“O usuário precisa confiar no que vê. Esta camada define como o FriendApp explica, em tempo real, por que uma postagem foi exibida, eliminando a sensação de caixa-preta algorítmica.”

🎯 Objetivo da Camada

- Mostrar justificativas claras para cada postagem exibida.
- Aumentar a confiança do usuário na IA Aurah Kosmos.
- Registrar logs de explicabilidade para auditoria.

⚙️ Estrutura de Explicabilidade

Elemento	Descrição Técnica
Critérios de Exibição	Similaridade, intenção, impacto coletivo, diversidade, recência

Elemento	Descrição Técnica
Mensagem Legível	Texto claro e curto para o usuário entender a razão
Logs de Justificativa	Registro técnico usado para auditoria interna

🧠 Pseudocódigo de Explicabilidade

```
def gerar_explicacao(post, user):
    criterios = []
    if post.similaridade > 0.8:
        criterios.append("alta compatibilidade")
    if post.intencao == user.meta:
        criterios.append("intenção alinhada")
    if post.impacto_coletivo > 0.7:
        criterios.append("impacto coletivo relevante")

    explicacao = f"Este post foi exibido por {', '.join(criterios)}."
    registrar_log_explicabilidade(post.id, user.id, criterios)
    return explicacao
```

📊 Exemplo de Payload de Explicação

```
{
  "post_id": "p123",
  "user_id": "u456",
  "criterios": ["alta compatibilidade", "impacto coletivo relevante"],
  "explicacao": "Este post foi exibido por alta compatibilidade e impacto col
etivo relevante.",
  "timestamp": "2025-09-08T17:12:00Z"
}
```

📦 Banco de Dados — Tabela `explicabilidade_feed`

Campo	Tipo	Descrição
`post_id`	UUID	Post analisado
`user_id`	UUID	Usuário que recebeu
`criterios`	JSON	Lista dos fatores que levaram à exibição
`explicacao`	String	Texto legível para o usuário
`timestamp`	Datetime	Momento da geração

🎨 UI/UX

- Ícone ℹ️ em cada post → ao clicar, abre card com explicação:

  > "Este post foi exibido por alta compatibilidade com sua intenção de 'cura' e impacto coletivo positivo."

- Feedback rápido: usuário pode marcar se a explicação fez sentido ou não.

📌 Notas Técnicas

- Logs de explicabilidade são guardados por **30 dias** para auditoria.
- Usuários Premium podem acessar uma visão detalhada no painel:
  - Ex.: "Score de compatibilidade: 0.87 / Impacto coletivo: 0.72".
- Explicações sempre usam **linguagem simples** → transparência sem complexidade técnica.

🔚 Fechamento

"A explicabilidade torna o feed transparente, confiável e auditável. O próximo passo é detalhar o **Sistema de Logs e Auditoria**, que garante rastreabilidade de todas as decisões da IA."

🔷 CAMADA 21 — SISTEMA DE LOGS E AUDITORIA DE DECISÕES DA IA

📍 Entrada

“Aurah Kosmos toma decisões a cada segundo: priorizar, ocultar, aplicar aviso. Para que haja confiança, cada decisão precisa ser registrada e auditável. Esta camada documenta esse processo.”

🎯 Objetivo da Camada

- Registrar todas as decisões da IA sobre postagens.
- Garantir **transparência técnica** e **auditoria retroativa**.
- Fornecer dados para evoluir o modelo e corrigir erros.

⚙️ Tipos de Decisões Registradas

Decisão da IA	O que é Registrado
Score de Ressonância	Valor calculado, vetores usados, variáveis aplicadas
Rotulagem Vibracional	Classificação: leve, intenso, colapso
Exibição/Prioridade	Se foi priorizado, exibido normal ou ocultado
Content Warning aplicado	Se overlay foi exibido e quantos usuários revelaram
Autoapagamento sugerido	Se IA recomendou remoção e motivo

🧠 Pseudocódigo de Registro

```
def registrar_decisao(post_id, user_id, score, rotulo, acao, motivo):
    log = {
        "post_id": post_id,
        "user_id": user_id,
        "score": score,
        "rotulo": rotulo,
        "acao": acao,
        "motivo": motivo,
        "timestamp": agora()
    }
    banco.logs_auditoria.insert(log)
```

📦 Banco de Dados — Tabela `logs_auditoria`

Campo	Tipo	Descrição
`log_id`	UUID	Identificação única do log
`post_id`	UUID	Post analisado
`user_id`	UUID	Usuário afetado
`score`	Float	Score calculado
`rotulo`	Enum	leve, intenso, colapso
`acao`	Enum	exibido, priorizado, ocultado, warning, apagado
`motivo`	String	Justificativa da decisão
`timestamp`	Datetime	Momento da decisão

📊 Exemplo de Log de Auditoria

```
{
  "log_id": "lg001",
  "post_id": "p111",
  "user_id": "u222",
  "score": 0.41,
  "rotulo": "colapso",
  "acao": "oculto",
  "motivo": "score abaixo do threshold configurado",
  "timestamp": "2025-09-08T18:30:00Z"
}
```

📌 Notas Técnicas

- Logs de auditoria são armazenados por **1 ano**.
- Moderadores podem consultar logs por post, usuário ou período.
- Dados alimentam dashboards para detectar **erros de classificação da IA**.
- Qualquer ação automática da IA precisa ter **justificativa registrada**.

🔚 Fechamento

“O sistema de auditoria garante que a IA nunca seja uma caixa-preta: cada decisão fica registrada, transparente e revisável. O próximo passo é detalhar a **Camada de Segurança e Privacidade dos Dados**, essencial para proteger os usuários.”

🔷 CAMADA 22 — SEGURANÇA E PRIVACIDADE DOS DADOS

📍 Entrada

“O feed sensorial lida com vetores, logs e interações íntimas. Para manter confiança, cada dado precisa ser protegido por design, desde a captura até o armazenamento.”

🎯 Objetivo da Camada

- Garantir **conformidade legal** (LGPD/GDPR).
- Proteger dados pessoais e vetoriais de usuários.
- Evitar vazamentos, acessos não autorizados e usos indevidos.

⚡ Medidas Técnicas de Segurança

Área	Medida Técnica
Armazenamento	Criptografia em repouso (AES-256) para vetores e logs
Trânsito	Criptografia TLS 1.3 em todas as APIs do feed
Anonimização	Vetores armazenados sem referência direta ao conteúdo original
Controle de Acesso	RBAC (Role-Based Access Control) para moderadores e admins
Logs Sensíveis	Acesso restrito com mascaramento de dados pessoais
Auditoria	Rastreamento de todas as consultas realizadas no feed

---

🧠 Pseudocódigo — Armazenamento Seguro

```
def salvar_post_vector(user_id, post_vector):
    encrypted_vector = AES256.encrypt(post_vector)
```

```
    db.insert({
        "user_id": user_id,
        "vector": encrypted_vector,
        "created_at": agora()
    })
```

📦 Banco de Dados — Tabela `user_vectors_secure`

Campo	Tipo	Descrição
`vector_id`	UUID	Identificação do vetor
`user_id`	UUID	Usuário dono do vetor
`vector_enc`	Blob	Vetor criptografado (AES-256)
`created_at`	Timestamp	Data de criação
`expires_at`	Timestamp	Data de expiração para retenção de segurança

📌 Políticas de Retenção

- Vetores expiram em **12 meses** se não forem atualizados.
- Logs sensoriais mantidos por **90 dias** para análise de impacto.
- Logs de auditoria de IA (decisões) armazenados por **1 ano**.

🛡️ Privacidade do Usuário

- Usuário pode **solicitar exclusão de seus dados** (direito de esquecimento).
- Posts apagados têm vetores removidos do banco em até 24h.
- Nenhum dado é compartilhado com terceiros sem consentimento explícito.

📊 Monitoramento

- Ferramentas: **Datadog, Grafana, SIEM** para alertas de intrusão.
- Alerta automático se houver acesso não autorizado.
- Revisões trimestrais de segurança (pentests).

🔚 Fechamento

“Segurança e privacidade são fundações. Com dados protegidos e compliance ativo, o feed pode expandir sem comprometer a confiança. O próximo passo é documentar a **Camada de Observabilidade e Relatórios Internos para Equipes**.”

🔷 CAMADA 23 — OBSERVABILIDADE E RELATÓRIOS INTERNOS PARA EQUIPES

📍 Entrada

“Para manter o feed saudável e eficiente, não basta calcular scores. É essencial monitorar métricas, gerar relatórios e oferecer painéis claros para as equipes técnicas e de moderação.”

🎯 Objetivo da Camada

- Fornecer **dashboards internos** de monitoramento de métricas de performance, impacto e segurança.
- Garantir relatórios periódicos para as áreas de **Engenharia, Produto e Moderação**.
- Facilitar auditorias e tomadas de decisão baseadas em dados.

⚙️ Principais Dashboards

Dashboard	Métricas Monitoradas
Performance do Feed	Latência média, cache hit rate, erros de API, uptime
Impacto Vibracional	Média de score dos posts, % de colapsos, expansão coletiva
Moderação e Segurança	Nº de content warnings aplicados, posts autoapagados, denúncias
Engajamento Sensorial	Nº de ressonâncias, ecos sensoriais, tempo médio no feed
Premium vs Free	Acesso a funcionalidades, upgrades gerados, uso do painel Premium

---

📊 Exemplo de Relatório Periódico

```
{
  "periodo": "2025-09-01 a 2025-09-07",
  "usuarios_ativos": 125000,
  "latencia_media": 210,
  "expansoes_geradas": 32000,
  "colapsos_detectados": 2800,
  "denuncias": 412,
  "premium_conversoes": 670
}
```

🧠 Pseudocódigo — Geração de Relatórios Semanais

```
def gerar_relatorio(periodo):
    dados = coletar_logs(periodo)
    relatorio = {
        "usuarios_ativos": contar_usuarios(dados),
        "latencia_media": media(dados.latencia),
        "expansoes_geradas": soma(dados.expansoes),
        "colapsos_detectados": soma(dados.colapsos),
        "denuncias": soma(dados.denuncias),
        "premium_conversoes": soma(dados.upgrades)
    }
    salvar_relatorio(relatorio, periodo)
    return relatorio
```

📦 Banco de Dados — Tabela `relatorios_feed`

Campo	Tipo	Descrição
`relatorio_id`	UUID	Identificação do relatório
`periodo`	String	Ex.: “2025-09-01 a 2025-09-07”
`usuarios_ativos`	Integer	Nº de usuários que abriram o feed

Campo	Tipo	Descrição
`latencia_media`	Float	Tempo médio de resposta
`expansoes_geradas`	Integer	Posts que elevaram a frequência coletiva
`colapsos_detectados`	Integer	Posts que reduziram a frequência coletiva
`denuncias`	Integer	Nº de denúncias recebidas
`premium_conversoes`	Integer	Nº de upgrades gerados via feed

📌 Notas Técnicas

- Dashboards em **Grafana/Metabase**, conectados ao banco e aos logs via **Elasticsearch**.
- Relatórios automáticos enviados por e-mail/Slack toda segunda-feira.
- Times de Engenharia usam KPIs de latência, Moderação usa KPIs de segurança, Produto usa engajamento e conversão Premium.

🔚 Fechamento

“A observabilidade garante que cada decisão seja mensurável e cada métrica auditável. O próximo passo é documentar a **Camada de Escalabilidade Global**, mostrando como o feed opera em múltiplas regiões do mundo.”

🔷 CAMADA 24 — ESCALABILIDADE GLOBAL E OPERAÇÃO MULTI-REGIÃO

📍 Entrada

“O feed sensorial precisa atender milhões de usuários em diferentes países, com latência mínima. Esta camada documenta a arquitetura multi-região para garantir escalabilidade e resiliência global.”

🎯 Objetivo da Camada

- Descrever como o feed opera em **infraestrutura distribuída**.
- Garantir **baixa latência (<300ms)** independente da região do usuário.
- Evitar single point of failure com arquitetura resiliente.

⚡ Estratégias de Escalabilidade

Estratégia	Descrição Técnica
Multi-Cloud	Hospedagem em Google Cloud + AWS (redundância)
Regiões Ativas	Clusters independentes em América, Europa e Ásia
Replicação de Banco	Postgres + Firestore replicados por região com sincronização
Balanceamento Global	Cloudflare + DNS inteligente para direcionar usuários
Particionamento de Dados	Sharding por região/usuário para evitar sobrecarga
Failover Automático	Se uma região cair, tráfego redirecionado em segundos

🧠 Pseudocódigo de Direcionamento Multi-Região

```
def roteamento_global(user_location):
    if user_location in ["América do Sul", "América do Norte"]:
        return "cluster_americas"
    elif user_location in ["Europa", "África"]:
        return "cluster_europe"
    else:
        return "cluster_asia"
```

📊 Exemplo de Arquitetura

- Usuário no Brasil → Feed servido pelo **cluster América do Sul**.
- Usuário na França → Feed servido pelo **cluster Europa**.
- Se cluster Europa cair → Tráfego redirecionado automaticamente para América do Norte.

📦 Banco de Dados Distribuído — Estrutura

Campo	Tipo	Descrição
`region`	String	Região do cluster (americas, europe, asia)

Campo	Tipo	Descrição
`user_id`	UUID	Usuário pertencente à região
`feed_cache`	JSON	Snapshot local do feed
`sync_status`	Boolean	Se já foi replicado globalmente
`updated_at`	Timestamp	Última sincronização

📌 Notas Técnicas

- Cache e logs são **regionais** → só dados agregados são sincronizados globalmente.
- Reduz custo e latência mantendo dados sensíveis próximos do usuário.
- Estratégia **Active-Active**: todas as regiões estão ativas, sem dependência central.
- Testes de caos simulam falhas mensais para validar resiliência.

🔚 Fechamento

“Com arquitetura multi-região, o FriendApp garante fluidez global, sem depender de um único ponto. O próximo passo é detalhar a **Camada 25 — Integrações Cíclicas do Ecossistema (Seção Final)**, que fecha o manual conectando o feed ao restante da plataforma.”

🔷 CAMADA 25 — INTEGRAÇÕES CÍCLICAS DO ECOSSISTEMA (SEÇÃO FINAL)

📍 Entrada

“O Feed Sensorial não é apenas uma funcionalidade — ele é o coração que se conecta a cada sistema do FriendApp. Esta camada final mapeia todas as integrações cíclicas, garantindo clareza total para desenvolvedores.”

🎯 Objetivo da Camada

- Documentar **dependências cruzadas** (entrada e saída de dados).

- Garantir que nenhuma funcionalidade fique isolada.
- Dar aos devs um **mapa único de conexões** do feed.

⚙️ Sistemas dos quais o Feed Depende

Sistema Origem	Dados Fornecidos	Tipo de Integração
Cadastro Consciente	user_vector inicial e dados de perfil	API + Banco
Teste de Personalidade	Arquétipos e preferências energéticas	API + Logs
Aurah Kosmos (IA)	Vetorização, score, moderação	Webhook + API
Mapa de Frequência	Frequência local/regional	Consulta periódica
Sistema Premium	Flags de recursos avançados	API + Cache

📤 Sistemas que Recebem Dados do Feed

Destino	Dados Enviados	Tipo de Entrega
Mapa de Frequência	Impactos locais/regionais de postagens	Logs + API
Conexões Autênticas	Afinidade entre usuários com base em posts	Banco + Trigger
Chat Vibracional	Contexto inicial de conversas (post origem)	API + Trigger
Eventos & Locais Parceiros	Posts relacionados a experiências e check-ins	API + Integração
Jogo da Transmutação	Postagens usadas como gatilhos de desafios	API + Logs
Painel de Insights Premium	Dados de impacto e ressonância	Consulta direta

🧠 Fluxo Global do Feed

```
flowchart TD
  U[Usuário publica post]
  U --> A[Aurah Kosmos analisa]
  A --> F[Feed Sensorial exibe]
  F --> M[Mapa de Frequência atualiza]
  F --> C[Conexões Autênticas reforçam afinidade]
```

```
F --> H[Chat Vibracional inicia conversa]
F --> E[Eventos/Parceiros recebem eco]
F --> J[Jogo da Transmutação registra gatilho]
F --> P[Painel Premium gera insights]
```

📦 Banco de Dados — Estrutura de Integrações

Tabela	Descrição
`feed_integra_entradas`	Registra origens de dados usados pelo feed
`feed_integra_saidas`	Registra sistemas que receberam dados do feed
`feed_fluxo_logs`	Histórico de cada evento cíclico

📌 Notas Técnicas

- Todas as integrações são **event-driven** (Kafka, Webhooks) → feed não trava se outro sistema estiver lento.
- Logs de integração armazenados por **90 dias**.
- Dependências críticas (Aurah, Cadastro, Premium) têm fallback local em cache.

🔚 Fechamento Final

“O Feed Sensorial é o centro nervoso do FriendApp. Ele consome dados do perfil e da IA, processa em tempo real e retroalimenta todo o ecossistema com impacto, conexões, insights e jogos. Com esta seção final, o manual está completo, pronto para execução em escala global.”